PLATAFORMA AUTOMATIZADA DE DETECCIÓN DE MALWARE

núm. 1363

Memoria del proyecto de Ingeniería Superior en Informática.

Presentado por Ferran Pichel Llaquet
codirigido por Miguel Ángel Domínguez
y dirigido por Marc Moreno Berengué.

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El firmante, Marc Moreno Berengué, profesor del Departamento de Microelectrónica y Sistemas Electrónicos de la Universidad Autónoma de Barcelona.
CERTIFICA:

Que la presente memoria ha sido realizada bajo su dirección por Ferran Pichel Llaquet.

Bellaterra, 16 de Septiembre de 2009

Firmante: Marc Moreno Berengué

Bellaterra, 16 de Septiembre de 2009

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El firmante, Miguel Ángel Domínguez de la empresa **Internet Security Auditors, S.L.**.
CERTIFICA:

Que el proyecto correspondiente a la presente memoria ha sido realizado bajo su dirección mediante convenio firmado con la **Universidad Autónoma de Barcelona**.

Asimismo, la empresa tiene conocimiento y aprueba el contenido que se detalla en esta memoria.

Y para que así conste, firma el presente documento.

Firmante: Miguel Ángel Domínguez

Bellaterra, 16 de Septiembre de 2009

Dedicado a Ángel Puigventós y Mario Ballano por crear las bases que han permitido el desarrollo de este proyecto.

Agradecimientos

Agradezco a todo el equipo de **Internet Security Auditors, S.L.** por brindarme la oportunidad de planificar, diseñar y desarrollar todo el proyecto. Con esta responsabilidad he podido aprender a llevar a cabo las tareas de forma autónoma y encontrar las soluciones a todos los problemas detectados antes y durante la implementación, además de aprender a realizar proyectos de software complejos como el descrito en el presente texto.

Agradecer de forma especial a Ángel Puigventós, Nuria Colinas y Juan Galiana las aportaciones en ideas y estrategias para afrontar algunos de los problemas de planificación e implementación. Además de sugerir multitud de funcionalidades adicionales que no se encontraban en el diseño inicial, gracias a sus aportaciones continuas a lo largo del desarrollo se ha logrado un diseño fácilmente modificable y escalable con el que se ha permitido adaptar rápidamente las nuevas características de las que debía disponer.

Índice general

Índice de figuras

Capítulo 1

Introducción

En este capítulo se presentan las motivaciones que han llevado a la realización del proyecto, el contexto sobre el que se sitúa y los objetivos que se deben cumplir para su éxito.

1.1. Contexto Actual

La seguridad en Internet ha ido evolucionando durante los últimos años. A principios de este siglo y finales del pasado la seguridad en los servidores era altamente deficiente, por ello eran objetivo de ataques de distinta índole. El objetivo podía ser muy variado, tanto la información que éste contenía como la auto-propaganda del propio atacante, entre otras.

Con la evolución de las nuevas tecnologías y su asentamiento como plataforma principal de acceso a la información, tanto gobiernos como entidades se han preocupado en mayor medida del problema que suponía una seguridad tan deficiente. Llevando a cabo notables medidas de protección para evitar muchos de los ataques ya conocidos.

En consecuencia, los atacantes han modificado su objetivo y lo que antes eran intrusiones a servidores de grandes entidades se han convertido en infecciones masivas a usuarios. Muchos de ellos, aunque familiarizados con las nuevas tecnologías, desconocen aspectos de la seguridad digital como pueden ser la comunicación entre cliente y servidor cifrada, jerarquía de certificados... o incluso la propia estructura de la red. Este hecho les da una gran ventaja a los atacantes, que intentan aprovechar cualquier descuido para infectar el entorno del usuario y robar información sensible. Precisamente a raíz de este desco-

nocimiento existe un notable número de usuarios que pulsarán sobre el botón *Aceptar* a cualquier aviso de seguridad sin pensárselo dos veces, o ni si quiera una.

Existen multitud de ataques que se utilizan actualmente para conseguir acceder al sistema del usuario y obtener así las credenciales para acceder a la banca electrónica u otros servicios sensibles. Aunque también son utilizados por grandes entidades para recabar información acerca de los hábitos de consumo de la población y otro tipo de datos que puedan ayudar a mantener una dinámica de negocio más eficiente en un futuro.

El más común de los ataques, y que crece en número cada año, es la utilización de software malicioso (ing: *Malware*[1]).

Debido a la naturaleza dinámica que forma actualmente el entorno Web de la red, puede ocurrir que una infección se realice únicamente durante un corto periodo de tiempo, o que dependa de otros aspectos y sólo infecte bajo unas condiciones predefinidas: una versión concreta del navegador o de cualquier otro software, la utilización de alguna librería concreta. Además, los ataques son infinitamente variados y complejos, de manera que realizar una detección basada en trazas o heurísticas, que es como lo hacen actualmente los antivirus, no es posible.

Para detectar de forma eficaz esta clase de ataques de *Malware* se debe realizar un análisis continuo de la aplicación Web y comprobar que no se producen anomalías Realizar este trabajo de forma manual es algo totalmente desaconsejable, para ello existen aplicaciones que de forma automática se conectan a una página especificada y analizan todas los eventos que se llevan a cabo en el sistema durante la carga y una vez visualizada la página. Al terminar, se muestran los resultados.

1.2. Motivaciones del Proyecto

Este proyecto nace de las nuevas necesidad detectadas durante el último año en la actual Plataforma de Detección de *Malware*. Su cometido es realizar un análisis de *Malware* de un portal web facilitado por el cliente. El proceso se realiza en dos fases principales, las cuales son ejecutadas por distinto software específico para cada acción:

[1] *def*: Software creado para realizar acciones maliciosas en un sistema. Primero infecta al usuario y se instala de forma clandestina en el sistema para obtener información, ésta dependerá de los intereses del autor del software, o simplemente mantener el acceso al sistema infectado. Algunos son virus, troyanos, puertas traseras...

Fase de Crawling: Proceso mediante el cual se obtiene el listado de *URLs*[2] que forman el dominio a analizar.

Fase de Analizador: Tomando como objeto a analizar las URLs recolectadas en la fase anterior, se lanza el sistema analizador o *HoneyClient* para cada una de ellas interpretando y almacenando los resultados obtenidos.

Algunos aspectos clave del diseño como la escalabilidad del software, gestión centralizada de errores, *logs* detallados de las acciones realizadas, inclusión de varios clientes en el mismo sistema o facilidad de manipulación de datos, no se tuvieron en cuenta en su primera versión y suponen un coste importante de tiempo en supervisión y mantenimiento básico de muchos aspectos automatizables. Éstas son deficiencias que este proyecto debe cubrir en su totalidad de una forma eficiente y robusta.

Adicionalmente se incorporarán algunas mejoras funcionales, centralizando varios aspectos del análisis en el mismo gestor, relacionando resultados de Antivirus, Analizador y *Crawler* de forma coherente y cohesionada, se evita así generar resultados independientes y tener que relacionarlos manualmente por marcas de tiempo u otros métodos.

1.3. Objetivos

El proyecto consiste en el desarrollo de un software capaz de gestionar la ejecución de los diferentes procesos de *Crawling* y *HoneyClient*, recopilando los resultados de forma coherente y almacenarlos en una base de datos. Adicionalmente deberá ser capaz de responder de forma eficaz ante el mayor número de contratiempos posibles y reaccionar correctamente en el proceso habitual de análisis. Como puede ser enviar alertas a los clientes en caso de detectar *Malware* en alguno de los recursos.

El software debe ser capaz de gestionar el análisis de varios dominios, además de cubrir todas y cada una de las deficiencias detectadas en la versión actual del sistema. Adicionalmente se deberán añadir funcionalidades y modificar el diseño para permitir centralizar la información en un mismo contenedor, como una base de datos, y relacionada

[2] *def*: Proviene del inglés: *Uniform Resource Location* que significa Localizador Uniforme de Recurso y corresponde a la cadena de caracteres que forma la dirección con la que se accede a un recurso web concreto. La sintaxis es la siguiente:

protocolo://máquina/directorio/archivo

entre sí de forma coherente aunque las fuentes de información sean diferentes: *crawler*, antivirus, *honeyclient*.

El diseño deberá ser escalable y permitir implementar futuras mejoras y ampliaciones de forma cómoda. ésta máxima debe aplicarse tanto al software desarrollado en Python como la base de datos que sustentará toda la información.

1.4. Conceptos

En este apartado se presentan algunos conceptos importantes para la comprensión de este texto.

1.4.1. HoneyClient

HoneyClient es un software destinado a detectar y recolectar las acciones que se realizan en un sistema cliente cuando se accede a algún *site*[3] remoto. La esencia de su comportamiento ha sido heredada de sus antecesores, los *HoneyPots*[4]. Lo que éstos realizan a nivel de servidor, el *HoneyClient* lo hace en el sistema del cliente. Hay algún que otro proyecto por la red destinado a esta clase de software, en la bibliografía se indican algunos de éstos.

Existen varias categorías, tanto *HoneyClients* como *HoneyPots*:

- **Alta-Interactividad:** Se utiliza un sistema real en el que se instala un agente que controla todos los eventos que se producen en el sistema y los reporta de un modo seguro con el fin de ser analizados.

- **Baja-Interactividad:** Se emula únicamente un servicio/cliente concreto en lugar de todo el sistema, programando a conciencia el error para controlar el origen de la explotación. De forma análoga a los de *Alta-Interactividad* se reportan los resultados para su posterior estudio.

[3] *def*: Sinónimo de página web

[4] *def*: Servidores señuelo utilizados para estudiar el comportamiento de un atacante. Se configuran para que parezcan servidores reales, incluso con datos falsos para ofrecer mayor realismo al atacante, que se ha convertido en presa. Sin embargo, todas las acciones son registradas para un posterior estudio y realizar planes de prevención de intrusiones a servidores en producción que pueden llevar a consecuencias desastrosas.

La solución más eficaz es la utilización de un *HoneyClient* de *Alta-Interactividad*, éste se trata realmente de un entorno real, consiguiendo un entorno más adecuado para que el código malicioso realice sus acciones. Además, es posible detectar ataques que afectan a otras aplicaciones y no únicamente al navegador, éstas pueden ser reproductores multimedia, flash, visualizadores de ficheros PDFs... Incluso ataques desconocidos o *0-day's*[5], ya que al no tratarse de un sistema pre-diseñado para un error concreto, el código malicioso tendrá acceso a todo el sistema.

1.4.2. Crawling

Se denomina *Crawling* a la acción de acceder de forma recursiva a todos los enlaces de una página Web con el fin de obtener los nombres de cada uno de los recursos a los que se puede acceder desde la misma.

La plataforma deberá gestionar la ejecución de un proceso de *crawling* con el fin de obtener todos los recursos que componen cada uno de los portales web a analizar. La gestión de estos procesos y sus resultados deberá realizarse correctamente.

1.4.3. XML

Se trata de un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Las siglas vienen del inglés *Extensible Markup Language* que significa: Lenguaje de Marcas Extensible. El aspecto y estructura que define es el utilizado por muchos lenguajes extendidos como por ejemplo *HTML*[6].

XML únicamente define las características sintácticas del lenguaje, es decir, su estructura, pero no es propiamente uno, como sí lo es en cambio *HTML*.

Un pequeño ejemplo de código *XML* se encuentra en la *Figura-1.1*.

Se puede observar la utilización de etiquetas y atributos para definir el elemento. Este formato es muy versátil, y se puede utilizar tanto para almacenar datos como implementar protocolos de comunicación. En la *Figura-1.2* y *Figura-1.3* se pueden observar ejemplos de utilización de *XML* en los ámbitos expuestos.

[5] *def*: Programas que aprovechan vulnerabilidades aun no conocidas o que acaban de ver la luz con cualquier finalidad, la que su creador prefiera.

[6] *def*: Las siglas provienen del inglés *HyperText Markup Language* que significa Lenguaje de Marcas de Hipertexto

```
<parent>
    <child1 attr1="value1">
        optional txt field
    </child1>
</parent>
```

Figura 1.1: Ejemplo de código *XML*

```
<client>
    <operation name="suma">
        <var name="variable1" valor="2" />
        <var name="variable2" valor="3" />
    </operation>
</client>
```

Figura 1.2: Ejemplo de petición utilizando *XML*

El único inconveniente de este formato es el notable incremente del tamaño ocupado por la información debido a los nombres de atributo y etiquetas. Haciendo que la comunicación entre un cliente y servidor sea más pesada, y ocupando mas espacio en el disco cuando se almacena la información utilizando este formato, a diferencia de, por ejemplo, una base de datos, cuya función es minimizar el espacio ocupado.

1.4.4. Virtualización

Permite utilizar varios sistemas operativos en una sola máquina de forma simultánea. Esta clase de software emula una máquina entera, incluyendo memoria RAM, disco duro, dispositivos externos... Con lo que es posible instalar cualquier sistema operativo en ella y conseguir emular cualquier entorno de usuario, afrontando la infección desde un espectro más amplio y detectando mayor número de software malicioso.

1.4.5. Sniffer

Software destinado a la recolección de los datos que se transmiten por la red. Con el fin de analizarlos y, en medida de lo posible, comprenderlos.

Existen multitud de herramientas que realizan esta funcionalidad, sin embargo, en el proyecto se ha implementado un objeto que realiza esta funcionalidad. Para ello se ha utilizado un módulo de Python con el fin de integrarlo en el software y comunicarse de forma más cómoda con el resto de la plataforma.

```
<network range="192.168.0.0/24">
    <host addr="192.168.0.2" state="up">
        <services total="2">
            <service name="telnet" port="23" banner="Telnet server" />
            <service name="http" port="80" banner="Apache" />
        </services>
    </host>
</network>
```

Figura 1.3: Ejemplo de almacenaje de información utilizando *XML*

1.5. Contenido de la Memoria

La memoria se ha estructurado en cinco grandes bloques, el primero es en el que se encuentran estas lineas. Su cometido es el de introducir el tema del proyecto y familiarizar al lector con los conceptos que se utilizan y desarrollan posteriormente en el trabajo.

Después se encuentra el capítulo en el que se expone el planning del proyecto y se describen las herramientas utilizadas, se presenta también una tabla en la que se puede observar de forma desglosada todo el proyecto estructurado por fases y la dedicación en cada una de ellas.

El cuerpo del trabajo corresponde al siguiente capítulo, en el que se explica y comenta todo el proceso llevado a cabo para el diseño de todo el proyecto y su posterior implementación, obviando detalles de la base de datos por cuestiones de confidencialidad y seguridad en la aplicación. Ya que podrían realizarse ataques complejos con el fin de alterar la información que ésta contiene o provocar errores en la plataforma u otro software que utilize la base de datos.

En el capítulo de pruebas se exponen algunos de los tests realizados al software y su resultado. Comentando además el motivo por el que se han realizado cada una de las pruebas.

El último apartado corresponde a las conclusiones a las que se ha llegado una vez finalizado el proyecto, exponiendo una valoración personal y unas lineas de aplicaciones o servicios futuros que se pretenden desarrollar sobre la plataforma.

Capítulo 2

Metodología de trabajo y entorno de diseño

En este capítulo se describe la metodología de trabajo realizada para el diseño e implementación de la plataforma.

2.1. Metodología de trabajo

El trabajo se ha desarrollado de forma individual, tanto la implementación como el diseño han sido hechos por un único individuo, al igual que la planificación. Desde **Internet Security Auditors, S.L.** se ha controlado el cumplimiento del planning y los resultados obtenidos en cada uno de los hitos correspondientes. Además de facilitar un acceso cómodo para desarrollar, permitiendo utilizar remotamente el entorno.

Todo el proceso se ha llevado a cabo en horas extra-laborales, estableciendo una media de 4 horas diarias durante 5 días a la semana.

Se empezó diseñando en un inicio la arquitectura general de la plataforma y las funcionalidades que debía requerir. Después se han ido detallando cada uno de los módulos a medida que se abordaba su diseño e implementación, al mismo tiempo se iban añadiendo funcionalidades no enumeradas al inicio, ocasionando algún que otro retraso en el cumplimiento del planning inicial.

2.1.1. Planificación

Seguidamente se presenta la planificación en formato de tabla ya que su desarrollo ha sido secuencial y no tiene mucho sentido mostrar el diagrama de Gantt. Cada día corresponde a 4 horas de trabajo, y la documentación se ha ido realizando a medida que avanzaba el proyecto, dejando para la última fase el agruparla y darle formato.

Plataforma automatizada	173,38 días?	12/01/09 09:00	10/09/09 12:00
Preparación	3,63 días	12/01/09 09:00	15/01/09 16:00
Análisis del software HoneyClient	2 días	12/01/09 09:00	13/01/09 19:00
Gestión de Riesgos	0,5 días	15/01/09 10:00	15/01/09 16:00
Módulo 1 crawling/BD	42 días	10/03/09 09:00	07/05/09 09:00
Análisis	0,5 días	10/03/09 09:00	11/03/09 10:00
Diseño	1 día	11/03/09 10:00	12/03/09 10:00
Adaptación DB	1 día	11/03/09 10:00	12/03/09 10:00
Implementación	2,5 días	12/03/09 15:00	23/03/09 11:00
Test	2 días	23/03/09 11:00	30/03/09 12:00
Validación	1 día	30/03/09 12:00	31/03/09 12:00
Corrección	1 día	01/04/09 09:00	03/04/09 11:00
Documentación	2,5 días	03/04/09 11:00	08/04/09 12:00
Creación Entorno y Máquinas Virtuales	3 días	10/04/09 09:00	21/04/09 12:00
Gestión General de Errores	0,5 días	22/04/09 09:00	23/04/09 10:00
Módulo 2 HoneyClient/BD	18,13 días	25/05/09 09:00	18/06/09 10:00
Análisis	1 día	25/05/09 09:00	27/05/09 11:00
Diseño	1 día	27/05/09 11:00	01/06/09 10:00
Adaptación BD	0,5 días	01/06/09 10:00	02/06/09 11:00
Implementación	1,5 días	02/06/09 11:00	08/06/09 11:00
Test	1 día	08/06/09 11:00	11/06/09 10:00
Validación	1 día	11/06/09 10:00	12/06/09 10:00
Corrección	0,5 días	12/06/09 10:00	15/06/09 11:00
Documentación	1 día	15/06/09 11:00	18/06/09 10:00
Módulo 3 Gestión de Alertas	16 días	18/06/09 10:00	10/07/09 10:00
Análisis	1,5 días	18/06/09 10:00	24/06/09 10:00
Diseño	0 días	24/06/09 10:00	24/06/09 10:00
Adaptación BD	0,5 días	24/06/09 10:00	25/06/09 11:00
Implementación	1,5 días	25/06/09 11:00	01/07/09 11:00
Test	1 día	01/07/09 11:00	06/07/09 10:00
Validación	0 días	06/07/09 10:00	06/07/09 10:00
Corrección	0,5 días	06/07/09 10:00	07/07/09 11:00
Documentación	1 día	07/07/09 11:00	10/07/09 10:00
Módulo 4 Gestión Central	20 días	10/07/09 10:00	07/08/09 10:00
Análisis	0,5 días	10/07/09 10:00	13/07/09 11:00
Diseño	2 días	13/07/09 11:00	20/07/09 12:00
Adaptación BD	0,5 días	21/07/09 09:00	22/07/09 10:00
Implementación	1,5 días	22/07/09 10:00	28/07/09 10:00
Test	1 día	28/07/09 10:00	30/07/09 12:00
Validación	0 días	30/07/09 12:00	30/07/09 12:00
Corrección	1 día	31/07/09 09:00	04/08/09 11:00
Documentación	1 día	04/08/09 11:00	07/08/09 10:00
Módulo 5 Puesta a Punto	14,25 días	07/08/09 10:00	27/08/09 12:00
Análisis. Revisar funcionalidades e interacción entre módulos	0,5 días	07/08/09 10:00	10/08/09 11:00
Implementación mejoras detectadas	1 día	10/08/09 11:00	13/08/09 10:00
Redefinir listas de inclusión exclusión	1 día	13/08/09 10:00	17/08/09 12:00
Test	1 día	18/08/09 09:00	20/08/09 11:00

Validación	0 días	20/08/09 11:00	20/08/09 11:00
Corrección	1 día	20/08/09 11:00	25/08/09 10:00
Documentación	1 día	25/08/09 10:00	27/08/09 12:00
PRUEBAS DE ACEPTACIóN	5 días	27/08/09 12:00	03/09/09 12:00
Finalizar documentación	6 días	04/09/09 12:00	10/09/09 12:00

En la tabla se puede observar la división de las tareas en los módulos que componen la aplicación. Exceptuando el módulo de *Puesta a Punto*, el cual sirve para preparar la plataforma con el fin de que funcione en el entorno de producción, realizando pruebas sobre los dominios reales y el comportamiento ante contratiempos.

La base de datos se ha ido completando a medida que avanzaban los módulos funcionales, añadiendo los datos necesarios para cada uno de ellos manteniendo siempre el esquema principal de distribución de los datos.

Es importante señalar que la última fase se alargó algo más de lo esperado debido a contratiempos que no permitían la dedicación deseada y una mala estimación de la duración de las pruebas. Aunque se ha cumplido de forma bastante satisfactoria. Al ser un producto que evoluciona con el tiempo, se pretende seguir implementando mejoras en un futuro, por lo que aun no está finalizado por completo. Únicamente se ha implementado una base sólida sobre la que elaborar mejoras que es capaz de funcionar tal y como lo hace la plataforma actual pero de forma mucho más robusta y automatizada.

2.2. Entorno de diseño

En esta sección se describen algunas de las herramientas utilizadas y el entorno de trabajo sobre el que se ha desarrollo del proyecto.

2.2.1. Python

Lenguaje de programación interpretado, de alto nivel y muy versátil. La versión utilizada ha sido la 2.6. En el paquete base ya se incorporan muchos módulos que aportan infinidad de funcionalidades al lenguaje, permitiendo programar cualquier tipo de aplicación en pocas lineas. En caso de requerir alguna funcionalidad adicional se puede descargar de la red un módulo de forma gratuita o programar uno propio.

Al tratarse de un lenguaje interpretado permite explorar de forma cómoda las características del lenguaje. Sin embargo, tiene el problema de que hay errores de programación

que no se producen hasta que se ejecuta explícitamente la sentencia errónea.

El diseño del producto se ha enmarcada en el paradigma de la orientación a objetos, y Python proporciona una comodidad más que suficiente para plasmar el diseño de clases del papel al código, incluso soporta herencia múltiple de manera correcta sin provocar errores y sin necesidad de *Interficies* a diferencia de Java. Además libera memoria automáticamente mediante unos recolectares autónomos de basura que detectan los objetos que ya no son referenciados en el software.

2.2.2. MySQL

MySQL es un Sistema Gestor de Base de Datos (SGDB) gratuito que se puede descargar directamente desde su página web. Además se encuentra accesible en muchas distribuciones Linux en el repositorio propio de software.

Proporciona una cómoda instalación y posterior configuración. Se incluyen infinidad de herramientas que facilitan todo el proceso para iniciar un servidor en cualquier sistema y migrar los datos entre diferentes servidores. Además, su rendimiento es más que suficiente para la cantidad de datos que se deberán manejar en el proyecto.

MySQL puede utilizar varios motores de almacenaje distintos, en la implementación se ha utilizado *InnoDB* ya que permite hacer uso de *Foreign Keys*[1], muy necesarias para asignar una relación fuerte entre identificadores de diferentes entidades o tablas, aportando cohesión a los datos.

2.2.3. Linux

La plataforma debe integrarse lo mejor posible con el sistema operativo. Interactuando directamente con las herramientas del sistema. Linux es el nombre que recibe el núcleo del sistema o *kernel*, éste, conjuntamente con el operativo que incorpora muchas herramientas *GNU*[2] recibe el nombre de *GNU/Linux*. Existen varias distribuciones de éste núcleo en las que se implementa conjuntamente con sistema similar a *Unix*.

Además de por su filosofía abierta y libre, los sistemas GNU/Linux se caracterizan por permitir un control total al usuario administrador, que puede configurar y modificar

[1] *def*: Permiten establecer relaciones de claves primarias entre tablas diferentes.

[2] *def*: El proyecto *GNU* fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre: el sistema *GNU*.

cualquier parte del sistema sin infligir ninguna ley.

Permite una configuración sencilla de los servidores más extendidos además de una ayuda más que notable incorporada en el sistema base por si aparecen dudas durante el desarrollo o configuración. En la ejecución del proyecto ha sido extremadamente útil para la instalación de los servidores y software necesario para el desarrollo de la plataforma gracias también a su repositorio de aplicaciones que agilizan mucho la búsqueda de herramientas *on-line*.

Su sistema de *logs*, llamado *syslog*, permite a las aplicaciones enviar cadenas de texto que se mostrarán y tratarán del mismo modo que lo haría con cualquier otro servicio del sistema. Bastará únicamente con modificar una linea del fichero de configuración en la que se especifica el nombre del archivo de *log* que se desea crear y algunos parámetros más.

GNU/Linux permite un control total utilizando como interfaz la propia terminal, desde la que se puede ejecutar cualquier acción sobre el sistema. Esta característica facilita en gran medida el acceso remoto al sistema para el desarrollo del proyecto. Además, para la escritura del código se ha utilizada el editor *VIM*[3], el cual se ejecuta bajo un entorno de terminal por lo que se ha podido desarrollar el grueso del proyecto de forma remota.

2.2.4. VMware

Software de virtualización que permite crear diferentes máquinas virtuales en un mismo sistema. En la web oficial[4] se encuentra todo un abanico de aplicaciones destinados a la virtualización. Existen tanto versiones gratuitas como de pago y cada una de ellas tiene sus funciones.

Seguidamente se presenta un listado de algunos de los productos juntamente con las funcionalidades que permiten:

[3] *web*: Página oficial del editor *VIM*: http://www.vim.org/
[4] *web*: Suite VMware - http://www.vmware.com

Producto	**Descripción**
VMware Server	Permite alojar múltiples máquinas virtuales y garantizar el acceso a ellas remotamente mediante una autenticación basada en credenciales (usuario/contraseñoa). Es la herramienta perfecta para centralizar todas las máquinas virtuales.
VMware Player	Permite ejecutar una máquina virtual previamente creada. La única restricción es precisamente ésta, no se permiten crear máquinas virtuales utilizando *VMware Player*, en cambio, sí contiene todas las funcionalidades para ejecutar e interactuar con el escritorio virtual.
VMware Workstation	Uno de los productos más completos de VMware, permite realizar todas las funciones descritas y además el sistema ha sido optimizado para una ejecución más rápida y eficiente. Consecuentemente la interacción con el sistema es mucho más fluida que el resto. Como inconveniente se encuentra el hecho de que se trata de un software de pago, a diferencia de los dos primeros.

Para el desarrollo del proyecto se ha utilizado *VMware Server* y *Qemu* como proveedores de escritorios virtuales. De esta manera, el sistema analizador puede utilizar multitud de sistemas sin complicaciones. Para poseer un espectro de sistemas víctima lo más amplio posible, se deberá crear el mayor número de máquinas virtuales con diferentes sistemas operativos y versiones de navegador y complementos.

2.2.5. Qemu

Qemu es otro software de virtualización gratuito. Se puede descargar de su página oficial[5] para múltiples plataformas La razón de utilizar dos software de virtualización diferentes radica en la problemática de que existe software malicioso capaz de detectar un entorno de virtualización, en cuyo caso no se ejecutan para no ser detectados.

Al proveer a la plataforma de dos entornos de virtualización diferentes se consigue proporcionar una alternativa que puede no ser detectada por el software y, consecuentemente, éste será detectado de forma habitual.

[5] *web*: Qemu - http://www.qemu.org/

2.2.6. Software de Crawling

Existen múltiples herramientas de *crawling* que realizan la detección de recursos de un dominio. La plataforma generalmente utiliza un sistema externo para el análisis, pero también existen scripts concretos para algunos dominios debido a su complejidad.

Se deberá disponer de ambos crawler en el sistema y la plataforma deberá gestionar todos los resultados de forma homogenia y correctamente. Centralizando los resultados de ambos y almacenando la información en un único formato para una mayor comodidad en la consulta y modificación.

Un ejemplo de software destinado al proceso de *crawling* y que se encuentra disponible en toda distribución Linux es *Wget*[6] y puede conseguirse de su página oficial. Éste programa permite descargarse todo tipo de ficheros de cualquier Web, incluso tiene la funcionalidad de ejecutarse en modo recursivo y navegar por la totalidad del portal.

2.2.7. Sistema de Detección - HoneyClient

El sistema de detección de eventos que se utiliza en la actualidad consta de dos partes diferenciadas: un agente encargado de detectar cualquier evento del sistema de la máquina virtual y un gestor para controlar todo el proceso e ir enviando las URLs a analizar al agente.

En base a una lista de URLs facilitada, este software se encarga de navegar por las direcciones de la lista y analizar el sistema durante el proceso. Además guarda un log con la información retornada, almacenando también las eventos detectados asociados a una dirección concreta.

Estructura

El diagrama que se puede observar en la *Figura-2.1* muestra la estructura del sistema de detección.

Se comprende la estructura anidada de la arquitectura necesaria para el funcionamiento del sistema. Se requiere un sistema operativo, Linux por ejemplo, con un servidor de virtualización instalado correctamente para que el analizador pueda utilizarlo.

[6] *web*: Página oficial de *Wget* -http://www.gnu.org/software/wget/

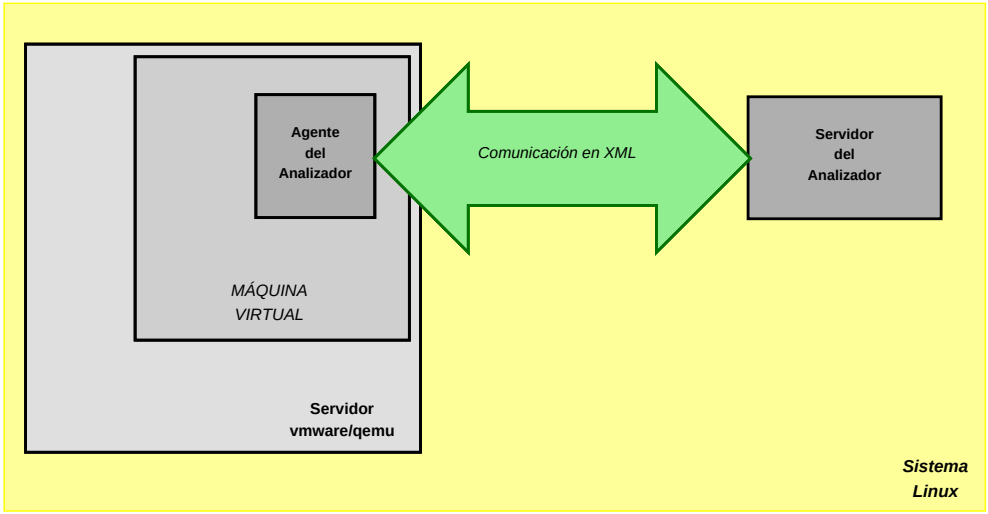


Figura 2.1: Estructura del Sistema de Detección bajo entorno Linux

Listas de Inclusión y Exclusión

El agente instalado en la máquina virtual detecta todos los eventos del sistema generando una lista de eventos de gran magnitud. Para evitar problemas relacionados con demasiada densidad de información, se realiza un filtrado de resultados.

Debido al gran número de acciones que se llevan a cabo en un sistema operativo, el software se abastece de tres listas de inclusión/exclusión: una destinada a los eventos relacionados con procesos, otra con el registro y finalmente la última destinada a los ficheros. De esta manera se permite especificar qué eventos no se desean notificar y cuales si, el reconocimiento de las acciones se basa en el uso de expresiones regulares que pueden ser de inclusión o exclusión. Se debe tener en cuenta que la lista de inclusión tiene preferencia en el proceso de filtrado. Éstas listas son enviadas al cliente cada vez que la máquina virtual es inicializada, ambas acciones las realiza el servidor.

Funcionamiento

Actualmente el sistema se ejecuta un total de 24 veces al día, es decir, realiza 24 ciclos diarios. Cuando un cliente facilita un dominio para incorporárlo al sistema de análisis se le asigna una prioridad para indicar la cantidad de procesos diarios a realizar:

Prioridad	Ciclos diarios
1	24
2	12
3	8
4	6

Cuando finalizan los 24 ciclos diarios, se envía un informe al cliente con los resultados, indicando si se ha detectado malware en alguno de los dominios y especificando los ciclos realizados.

Cada ciclo, después del proceso de *crawling* se inicia el *HoneyClient*, el servidor carga un conjunto de *URL*s de un fichero y seguidamente inicia la máquina virtual especificada en la configuración asegurándose que el agente allí instalado responde correctamente. Al confirmar el estado, el gestor envía las listas de exclusión e inclusión.

El gestor se encarga de dividir las *URL*s especificadas en bloques de cantidad personalizable, cinco actualmente, y transferirlas al agente. Adicionalmente gestiona la ejecución de la máquina virtual y la reinicia en caso de detectar anomalías o eventos no filtrados.

Una vez el agente ha sido inicializado se le envía el bloque de direcciones a analizar, y éste se encarga de ejecutar las diferentes instancias del navegador y cargar la dirección correspondiente a cada uno de ellos. El agente detecta todas y cada una de las acciones que ocurren en el sistema auditado, filtrando mediante las listas de inclusión/exclusión los eventos y retornando el resultado al servidor.

Mientras, el servidor controla el comportamiento del agente, comprobando su estado mediante peticiones *Ping*. Cuando el gestor recibe el resultado de un bloque de direcciones se prosigue con el siguiente. Este proceso se repite hasta que se finaliza la lista de *URL*s especificada.

El canal de comunicación se establece utilizando la arquitectura cliente/servidor, siendo el agente el cliente y servidor el gestor. La comunicación se transmite por la red creada entre la máquina virtual y el servidor, utilizando en todo momento *XML* para los mensajes.

Protocolo

El diagrama de la *Figura-2.2* muestra de forma simplificada la comunicación entre el agente y el Servidor. Existen más operaciones, pero lo habitual es que se realice esa clase de comunicación con alguna variación.

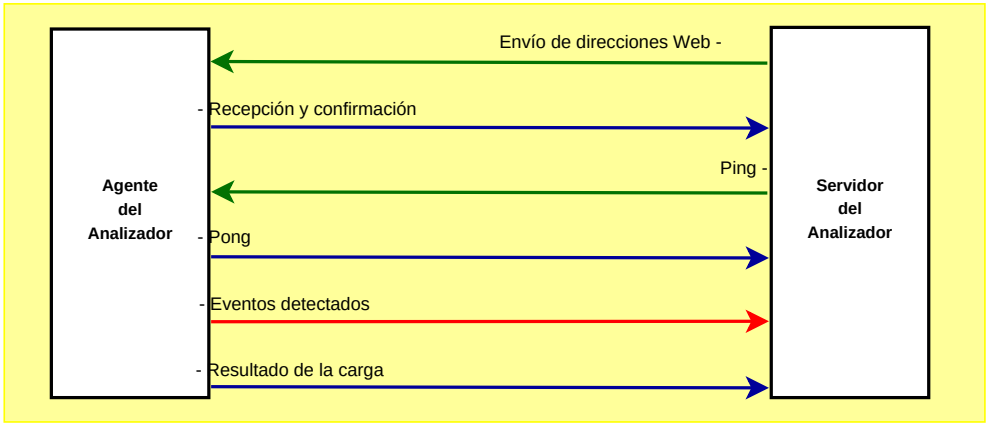


Figura 2.2: Ejemplo de comunicación entre el Agente y el Servidor

La flecha coloreada en rojo indica un evento de sistema no excluido por las listas de exclusión o incluido por las de inclusión. Cuando el sistema detecta un evento de este tipo significa que ha podido haber una acción maliciosa, y se prosigue a identificar la causa. El proceso es el siguiente:

1. Se reinicia la máquina virtual a su estado inicial mediante una imágen del disco previamente creada.
2. Se envían las listas de exclusión/inclusión.
3. Condicional:
   a) Si únicamente se analizaba una dirección, se marca como maliciosa y finaliza el proceso.
   b) Si se analizaba un grupo de *URL*s, se subdivide en dos grupos.
4. Se manda uno de los grupos al Agente para que lo analice
5. Condicional:
   a) En caso de detectar algún evento en el sistema, se vuelve al primer paso.
   b) Si no se detecta ningún evento, se envía el grupo restante al Agente para que lo analice y se repite el proceso.

Deficiencias

Desgraciadamente, existen deficiencias en el proceso seguido por este software. Según el procedimiento descrito, si existe una acción maliciosa, se espera que ésta sea repetida

en cada una de las siguientes iteraciones de carga de la página para poder discernir que dirección la ha provocado exactamente. Sin embargo, esto no ocurre siempre. Existen multitud de circunstancias por las que el ataque únicamente se realizará una vez, y cuando se vayan dividiendo las URLs en subgrupos para dar con la causante el evento no se repita, en consecuencia, el servidor no almacena la información de la infección. Como resultado, no aparecerá esa acción maliciosa en el log del analizador y pasará desapercibida.

Otro punto negro del sistema es que en alguna ocasión ha dejado de funcionar sin causa aparente, o incluso rara vez el gestor puede llegar a enviar infinidad de peticiones *Ping* sin recibir respuesta alguna. Este último problema provoca que en el momento de ver los resultados, no estén finalizados, además de impedir la ejecución continua.

2.2.8. Sistema de Antivirus

El software Antivirus utilizado es totalmente independiente al resto de programas. La función del antivirus es capturar todo el tráfico de la red e ir analizando utilizando trazas y heurísticas los paquetes enviados y recibidos en busca de virus. En caso de detectar algún código vírico, se genera una entrada en el fichero de *log*.

Debido al funcionamiento independiente del software antivirus, será necesario sincronizar las detecciones de código malicioso con el sistema de detección de eventos. Así se podrán asociar con exactitud los eventos maliciosos detectados por el *HoneyClient* con el código vírico identificado por el antivirus.

Capítulo 3

Diseño e Implementación

En este capítulo se describe el diseño e implementación del proyecto. Comentando de forma sintetizada los procedimientos llevados a cabo. Se obvian algunos detalles irrelevantes para la comprensión del proyecto por motivos de confidencialidad.

3.1. Arquitectura General

Para entender mejor la funcionalidad de cada uno de los diferentes módulos y cómo interactuarán entre ellos, se presenta en la *Figura-3.1* un diagrama con un esquema que describe el comportamiento general de la aplicación.

A continuación se definen los pasos enumerados en el diagrama.

1. El gestor recoge los datos iniciales de la base de datos.
2. El objeto de la base de datos retorna varias listas de dominios, una por cliente.
3. El gestor envía todos los dominios al gestor de *crawling*.
4. El gestor de *crawling* inicia el proceso de *crawling* para cada uno de ellos de forma eficiente y controlando cada uno de los procesos. Para ello crea varios nodos a los que envía la información referente al dominio.
5. El nodo de *crawling* termina su ejecución y envía el objeto que almacena los resultados al gestor de *crawlers*.
6. El gestor va recopilando los datos del gestor de *crawlers* hasta que éste termina.

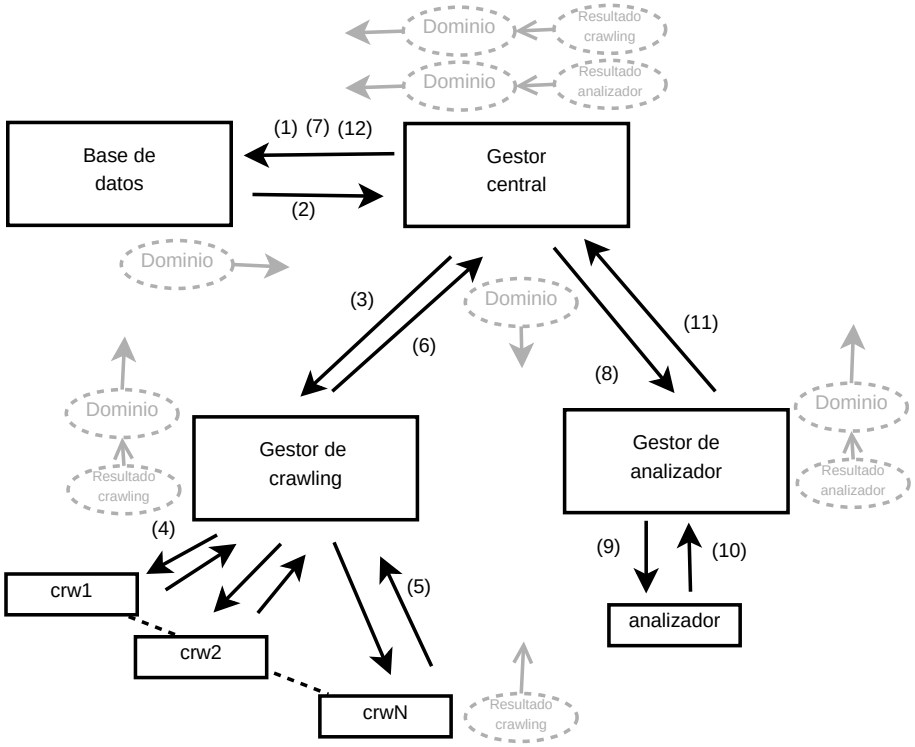


Figura 3.1: Esquema de la comunicación entre los diferentes módulos.

7. El gestor almacena los resultados del *crawler* de cada uno de los dominios a medida que éstos finalizan.

8. El gestor inicia el proceso de análisis. Para ello envía los dominios al gestor del analizador.

9. El gestor analizador carga un nodo de *HoneyClient* y le facilita los dominios.

10. El módulo analizador (*HoneyClient*) lanza el proceso y retorna los dominios terminados a medida que van finalizando su análisis.

11. El gestor del analizador recolecta los resultados de los dominios y los envía al gestor central.

12. El gestor central almacena los resultados en la base de datos y cuando todos los dominios terminan se recopila un nuevo ciclo de análisis.

Todo este procedimiento es el que se sigue en cada ciclo del sistema. Cuando finalmente se completan los 24 ciclos diarios, se envía el informe resumen al cliente.

3.2. Base de Datos

En este apartado se presentan las características generales de la base de datos diseñada e implementada para soportar los datos del software.

3.2.1. Análisis de las Funcionalidades

La base de datos necesaria para el desarrollo del proyecto debe cumplir una serie de requisitos con el fin de proporcionar las funcionalidades requeridas y la escalabilidad propuesta. Se trata del núcleo de la aplicación y dónde quedará constancia de todo lo sucedido en el proceso de análisis y las alertas enviadas - además del *syslog* -. La información debe estructurarse de forma correcta para evitar consultas excesivamente complejas.

La base de datos debe permitir una cómoda gestión de los datos necesarios para esta plataforma. Las relaciones tiene que ser robustas para evitar que datos incoherentes se queden aislados.

Al tratarse de un producto destinado a trabajar de forma dinámica, con borrados e inserciones continuas, se deberá minimizar el tamaño de las tablas destinadas a almacenar la información más volátil. De esta manera se optimiza el tiempo de ejecución de las sentencias.

Por otra parte todos los datos referentes a clientes, dominios, usuarios, alertas, resultados, filtros personalizados de *HoneyClient*, configuración de *crawler*.... es decir, todos los datos, han de almacenarse coherentemente y de forma relacionada entre si.

La naturaleza de los datos a almacenar sugiere una estructura de árbol, tomando como raíz el cliente. Éste estará relacionado con todos los dominios que le correspondan, además tendrá unos filtros de eventos propios y una lista de direcciones de correos electrónicos correspondientes a los destinatarios a los que notificar en caso de detectar alguna alerta durante el análisis.

Se diferencian dos grandes bloques, en uno se almacena la información referente a la configuración de la plataforma, dominios, máquinas virtuales... y en el otro se organizarán los resultados obtenidos para consultarlos posteriormente, este segundo módulo será mucho mas estático que el primero, ya que estos datos no deben ser modificados, en principio.

3.2.2. Diseño

En la *Figura-3.2* se puede observar el diagrama esquematizado para comprender mejor la estructura de la base de datos diseñada, aunque faltan elementos y detalles, se entiende la división por bloques de los datos.

No se ha realizado un esquema más detallado por cuestiones tanto de claridad como legibilidad al ser un tamaño considerable.

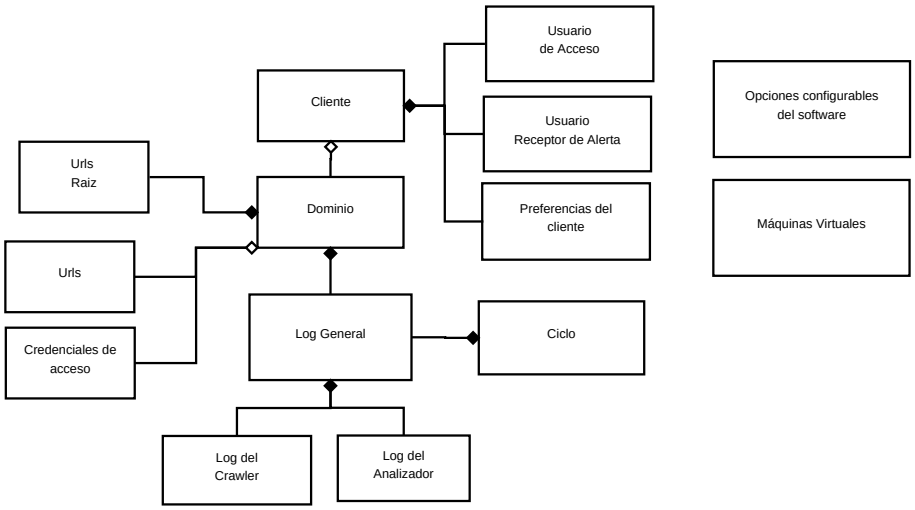


Figura 3.2: Diseño esquematizado de la base de datos

De la figura se desprenden algunas de las características del diseño. Cada cliente podrá configurar sus propias preferencias del software modificando los parámetros que serán cargados durante la ejecución. En estas preferencias se incluyen básicamente los filtros a aplicar a los eventos de registro, procesos y .sistema de ficheros.

Los clientes podrán facilitar además una lista de destinatarios de correo electrónico (o otros medios en un futuro) a los que enviar las alertas e informes de resultados. Por otra parte están los usuarios de acceso a la aplicación. Estos se utilizarán como credenciales en las consultas y deberán asociarse con los datos del cliente al que representan.

Cada cliente sera dueño de tantos dominios como requiera. Permitiendo configurar para cada uno de ellos las opciones del *crawler* (en caso de requerirlas), el módulo de *crawling* a utilizar, la periodicidad de escaneo, puerto, nombre... Y tal y como se observa en el diagrama, cada uno de ellos estará formado por varios recursos. Algunos de éstos, los llamados raíces, proporcionan el punto de entrada para analizar el portal, se utilizarán además en el módulo de *crawling* para detectar modificaciones significativas en la web e inicializar el proceso. Además, pueden configurarse portales estáticos sobre los que no se

realizará *crawling*, únicamente se analizarán los recursos raíz, que este caso serán todos los que forman el portal. Adicionalmente, si el dominio tiene autenticación, se pueden especificar las credenciales en la tabla *Credenciales de acceso*, actualmente se permite únicamente la autenticación *Basic*, en un futuro se deberá añadir autenticación mediante formularios.

Como ya se ha comentado anteriormente, el funcionamiento del software es mediante ciclos. Actualmente se realizan un total de 24 ciclos diarios, uno cada hora. Dependiendo de la periodicidad del dominio, se analizará: 24, 12, 8 o 6 veces. Para poder relacionar los ciclos con los resultados de cada proceso se ha creado la relación que se observa en el diagrama entre la entidad *Log General* y *Ciclo*.

Nótese que existen varias entidades destinadas a almacenar información de resultados. En el *Log General* se almacenarán las marcas de tiempo que delimitarán la duración del *crawler*, juntamente con el analizador. En el *Log del Crawler* el tiempo de inicio y fin del proceso de *crawling*. Lo mismo ocurre en el gestor del analizador, donde únicamente se almacenan las marcas de tiempo de dicho proceso junto con los resultados del análisis. Finalmente en la tabla *Ciclo* se establecen los tiempos de inicio y fin de todo un ciclo en el que se analizan multitud de dominios.

Mantener un control de tiempo tan detallado permite mejorar el rendimiento, optimizando el orden de los dominios en base a la duración estimada del proceso, tomando como muestra los resultados previos.

De manera independiente al resto de la información se almacena también en la base de datos la configuración de las máquinas virtuales y las opciones del software. Éstas configuraciones podrán modificarse cuando sea necesario, añadiendo más máquinas virtuales para ampliar el espectro de infección.

Todos los datos se relacionan mediante identificadores numéricos que se generan automáticamente. Cada dominio y cliente mantienen así una relación inequívoca con sus componentes y resultados.

3.3. Gestor de Crawling

En este apartado se describe el procedimiento llevado a cabo para el diseño y posterior implementación del módulo destinado a la gestión de los procesos de *crawling*.

3.3.1. Análisis de las Funcionalidades

Este módulo debe permitir lanzar varias instancias de *crawling*, gestionándolas de forma autónoma y transfiriendo los resultados al gestor principal.

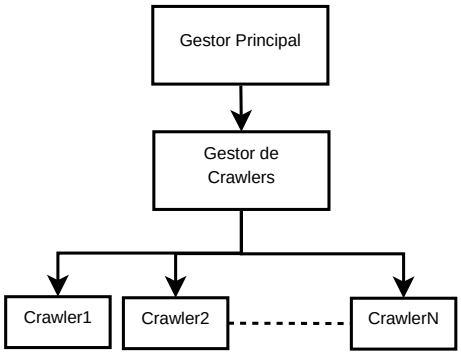


Figura 3.3: Diseño esquematizado del gestor de crawlers

Actualmente, la gran mayoría de dominios utiliza una aplicación externa de *crawling* cuyos resultados deberán gestionarse minuciosamente desde la plataforma. Adicionalmente, existen dominios que requieren funcionalidades específicas y cuyos *crawlings* se procesan utilizando pequeños scripts creados específicamente para este fin.

Debido a estas funcionalidades la aplicación deberá ser capaz de gestionar ambos tipos de *crawlers* y permitir añadir tantos como sean necesarios en un futuro. Tanto en forma de módulo Python como mediante pequeños scripts individuales para cada dominio.

Los resultados retornados deberán organizarse de forma correcta y permitir una consulta rápida y precisa de los mismos. En la *Figura-3.3* se observa un diagrama esquematizado de la arquitectura del gestor de *crawlers*.

3.3.2. Diseño

Para permitir una gestión independiente de los diferentes componentes del proceso de *crawling* como son el gestor y cada una de las instancias que se ejecutan de forma concurrente, se ha optado por la utilización de *Threads*[1] para cada uno de ellos.

Con el fin de optimizar el tiempo de ejecución se ha diseñado un pequeño algoritmo que, en base a los último diez tiempos de duración del crawling, permite una ejecución continua y los dominios más grandes nunca llegan a colapsar la cola de ejecución.

[1] *def*: Traducido como hilo de ejecución, permite a un mismo proceso realizar múltiples tareas concurrentemente y mantener un control preciso sobre cada una de ellas.

Principalmente se trata de ordenar de mayor a menor los *N* dominios según su tiempo de crawling. Seguidamente se calcula la media de todos los tiempos. Se añaden *N-4* dominios con una duración más larga y las *4* posiciones restantes se eligen del otro extremo de la cola, con lo que su duración es la mínima. Cuando un *crawling* finaliza se comprueba si la duración ha sido mayor que el tiempo medio, en ese caso se escoge el dominio del inicio de la lista - mayor duración -, en caso contrario el del final - menor duración - y así para cada proceso de *crawling* finalizado.

Diagrama

En el diagrama expuesto en la *Figura-3.4* se muestran únicamente los objetos que afectan de forma directa al gestor de crawlers y su gestión. Se ha marcado en rojo el gestor principal y en amarillo el de crawlers. Se observa también la arquitectura en árbol ya comentada.

Para permitir una compatibilidad total con las futuras ampliaciones se han desarrollado clases abstractas que facilitarán una integración más cómoda y completa. En estas clases se declaran los métodos que posteriormente deberían ser definidos en dichos módulos. De lo contrario se lanzará una excepción y la ampliación no funcionará correctamente.

En el gráfico también se ha hecho referencia al uso de *Threads*, gracias a la librería utilizada que proporciona el propio núcleo de Python, *threading.Thread*. Se apreció también el uso de herencias para optimizar el proceso de implementación, minimizando el número de lineas programadas al reaprovechar el código.

Resultados de Crawler

El gestor de *crawling* únicamente recibirá un tipo de objetos, que serán los resultados del *crawler*. Independientemente del módulo utilizado para el proceso, el gestor debe recibir la información estructurada siguiendo una pauta para poder tratarla de forma genérica.

Se ha diseñado un objeto para almacenar los resultados, de manera que será independiente del *crawler* utilizado. Dicho objeto puede observarse de forma simplificada en la *Figura-3.4*, únicamente se muestran los métodos más utilizados.

Los datos almacenados tienen un formato de array con un mínimo de tres posiciones, donde se almacena:

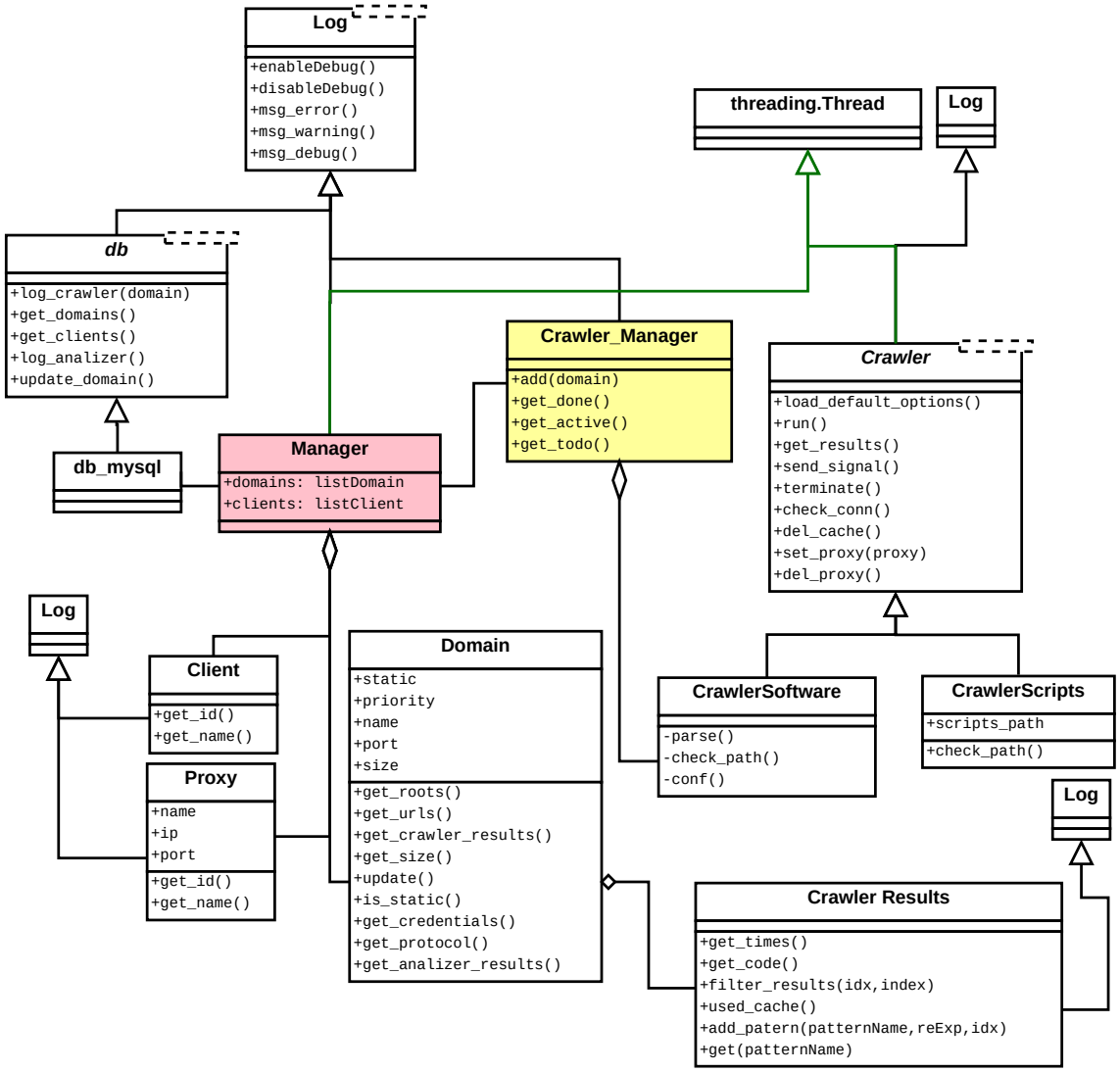


Figura 3.4: Diseño de clases del gestor de crawlers

```
[ longitud , recurso ,  <, opcionales > ]
```

La longitud del recurso se indica en primer lugar, seguido de la *URL* o dirección. Finalmente se facilitan los datos opcionales que pueda proporcionar el sistema de *crawling* y que pueden utilizarse, por ejemplo, para filtrar los resultados.

A continuación se presenta un ejemplo de contenido para estos campos:

```
[ 230 , 'http://www.dominio.com/recurso1.html' , 200 , 'new' ]
[ 340 , 'http://www.dominio.com/recurso2.html' , 300 , 'error' ]
[ 279 , 'http://www.dominio.com/recurso2.html' , 300 , 'no-modified' ]
```

Seguidamente se describen los métodos presentados en el esquema:

- **get_times()**: Retorna una lista con las marcas de tiempo de inicio y fin del proceso.
- **add_pattern(patternName,regExp,idx=0)**: Añade un patrón de clasificación de los resultados. Para ello se especifica en los parámetros el nombre del patrón, la expresión regular a cumplir y la posición del *array* que se evaluará.

  Una muestra de uso destinada al filtrado de los resultados expuestos en el ejemplo presentado sería:

  ```
  add_pattern('new','^new$', 3)
  add_pattern('errors','^error$',3)
  add_pattern('untouched','^no-modified$')
  ```

  En este caso se han creado tres patrones de filtrado. Para obtenerlos bastará con llamar al siguiente método.
- **get(patternName)**: Este método retorna los resultados que sean acordes con el patrón especificado. Siguiendo con el ejemplo propuesto, en este caso existirán los siguientes:
  - *'new'* - nuevos recursos.
  - *'errors'* - recursos con errores de carga: 404 (*Not Found*), 403 (*Forbidden*) ...
  - *'untouched'* - recursos que no han cambiado desde la última ejecución del crawler.
- **get_code()**: Retorna el código con el que ha finalizado la ejecución del *crawler*, 0 en caso de éxito y negativo en caso contrario.

- **filter_results(regExp,index=0)**: Retorna los resultados filtrados mediante una expresión regular facilitada en los parámetros. Permite realizar búsquedas sin necesidad de crear un patrón para ello.
- **cache_used()**: Devuelve *True* si el proceso de *crawling* se ha realizado utilizando cache y *False* en caso contrario.

Este diseño permite mucha flexibilidad de uso, el funcionamiento del filtrado de resultados puede ser configurado de cualquier manera y por lo tanto facilitará la integración con todo tipo de módulos y software de *crawling* futuro.

3.3.3. Implementación

La implementación del diseño ha sido finalizada con éxito utilizando código Python. Para ejecución de binarios externos se ha utilizado el módulo *subprocess*[2], el cual es proporcionado por el propio núcleo del paquete del lenguaje.

Como curiosidad a destacar, el módulo *subprocess* no permite esperas temporales, es decir, ejecutar un binario y esperar un máximo de N segundos, para ello se ha tenido que modificar dicho módulo añadiendo el parámetro *timeout* al método *call*. Esto es necesario para evitar que los scripts de crawling no se queden ejecutándose de forma indefinida. Las modificaciones realizadas permiten la ejecución de un código como el siguiente:

```
try:
    subprocess.call(['cmd','args'],timeout=N)
except:
   # Acciones a realizar si el proceso se retrasa
```

En el código expuesto, *N* corresponde a los segundos que se desean esperar como máximo. Y el código contenido en el bloque *except* se ejecutará únicamente cuando la llamada al subproceso necesite más tiempo del especificado en el parámetro *timeout*.

3.4. Gestor de HoneyClient

En este apartado se describe el procedimiento llevado a cabo para el diseño y posterior implementación del módulo destinado a la gestión de HoneyClients.

[2] *def*: Módulo de Python que brinda varias funcionalidades para la ejecución de binarios externos.

3.4.1. Análisis de las Funcionalidades

Las funcionalidades que debe realizar este módulo son muy variadas. En cuanto a la ejecución del software *HoneyClient*, se deberá mantener al detalle la comunicación entre el servidor y el cliente. Detectando cualquier posible error de dicho software y actuando en consecuencia.

Los eventos se deberán detectar en tiempo de ejecución para mantener un sistema de alertas más preciso y eficaz. Se deberá sincronizar la ejecución con el proceso de Antivirus para saber que *URL*s se están visitando y conocer así las que intentan acceder a código vírico. Discriminando en mayor medida estas alertas para no desbordar al que recibe los avisos.

Todo este proceso debe hacerse de forma automática sin ningún tipo de intervención humana.

3.4.2. Diseño

Este módulo ha sido el más laborioso de diseñar y probar. Debido a la complejidad de controlar de forma precisa la ejecución del software de análisis con el fin de evitar posibles interrupciones del servicio. Al tratarse de un proceso externo, el gestor únicamente lo lanza y controla su comportamiento.

Para poder llevar un control minucioso sobre la ejecución del analizador, se ha diseñado un sistema de *sniffing* de manera que se captura todo el tráfico entre el cliente y el servidor. Analizándolo en tiempo real y almacenando los datos incluso antes de que la ejecución termine, a medida que van apareciendo.

Se han diseñado varios objetos que funcionan mediante eventos y uno de ellos es precisamente el *sniffer*. Éste objeto es un *sniffer* genérico que permite aplicar diferentes filtros y adjuntar otro objeto en él. Cuando un paquete es capturado se llama al método *update* del objeto adjuntado, en este caso *Analizer Sniffer*. Éste servirá para procesar los datos y transformarlos a objetos *XML*. Del mismo modo, *Analizer Sniffer* mantiene objetos asociados a los que enviará los datos una vez procesados, dependiendo de si el destinatario es el cliente o el servidor se llamará a *update_client* o *update_server*, permitiendo un mejor control del protocolo utilizado. El mismo sistema se ha utilizado para gestionar la información proveniente del Antivirus, el cual notifica mediante el método *notify* al *update* del objeto asociado cuando se detecta un virus circulando por la red.

Se ha decidido independizar la gestión del proceso externo, *Analizer Api*, de la interpretación del protocolo, *Analizer Protocol*. De esta manera se gana comodidad al implementar y desarrollar modificaciones o correcciones en la gestión del protocolo y/o ejecución. Al tratarse de un software externo, es probable que en un futuro su protocolo varíe, con este diseño se deberá modificar únicamente uno de los objetos.

Diagrama

En la *Figura-3.5* se puede observar el diagrama de clases diseñado para el gestor y sus componentes, en rasgos generales es similar al gestor de *crawling*, con sus nodos analizadores y el gestor. No se muestran los objetos ni métodos necesarios para la gestión de alertas, se ha preferido dedicar una sección entera a este submódulo por cuestiones de legibilidad y claridad del diagrama. Por el mismo motivo, se han coloreado de color verde los objetos y clases que heredan de *threading.Thread* y eliminado los métodos que ya han aparecido en figuras anteriores, de esta manera se presenta el diagrama de forma mucho más clara e intuitiva.

Las relaciones que se observan en lineas discontinuas corresponden a la comunicación basada en eventos ya comentada, nótese que únicamente aparecen en funciones tipo *update*/*notify*.

Resultados del Analizador

Para cada dominio se almacenan los resultados de cada una de sus *URL*s, de esta manera se consigue poder reanudar la ejecución en caso de error del software o sistema. Bastará con reiniciar la ejecución y analizar únicamente las *URLs* para las que no existe resultado aun.

Recordando el algoritmo utilizado por el software cuando se detectan acciones maliciosas, es posible que aparezcan *falsos positivos* cuando una acción no se repite en los siguientes análisis. Gracias a la utilización del sniffer se podrá interceptar dicho evento.

Para diferenciar los posibles falsos positivos del malware confirmado se ha creado el método *is_confirmed_malware*, al que se puede preguntar si una *URL* es maliciosa o no. Ésta diferenciación deberá tenerse en cuenta también en el proceso de envío de alertas.

El objeto contenedor de estos resultados también indica cuando ha finalizado el análisis de un dominio. La razón es que el software aun no ha cesado su ejecución cuando el

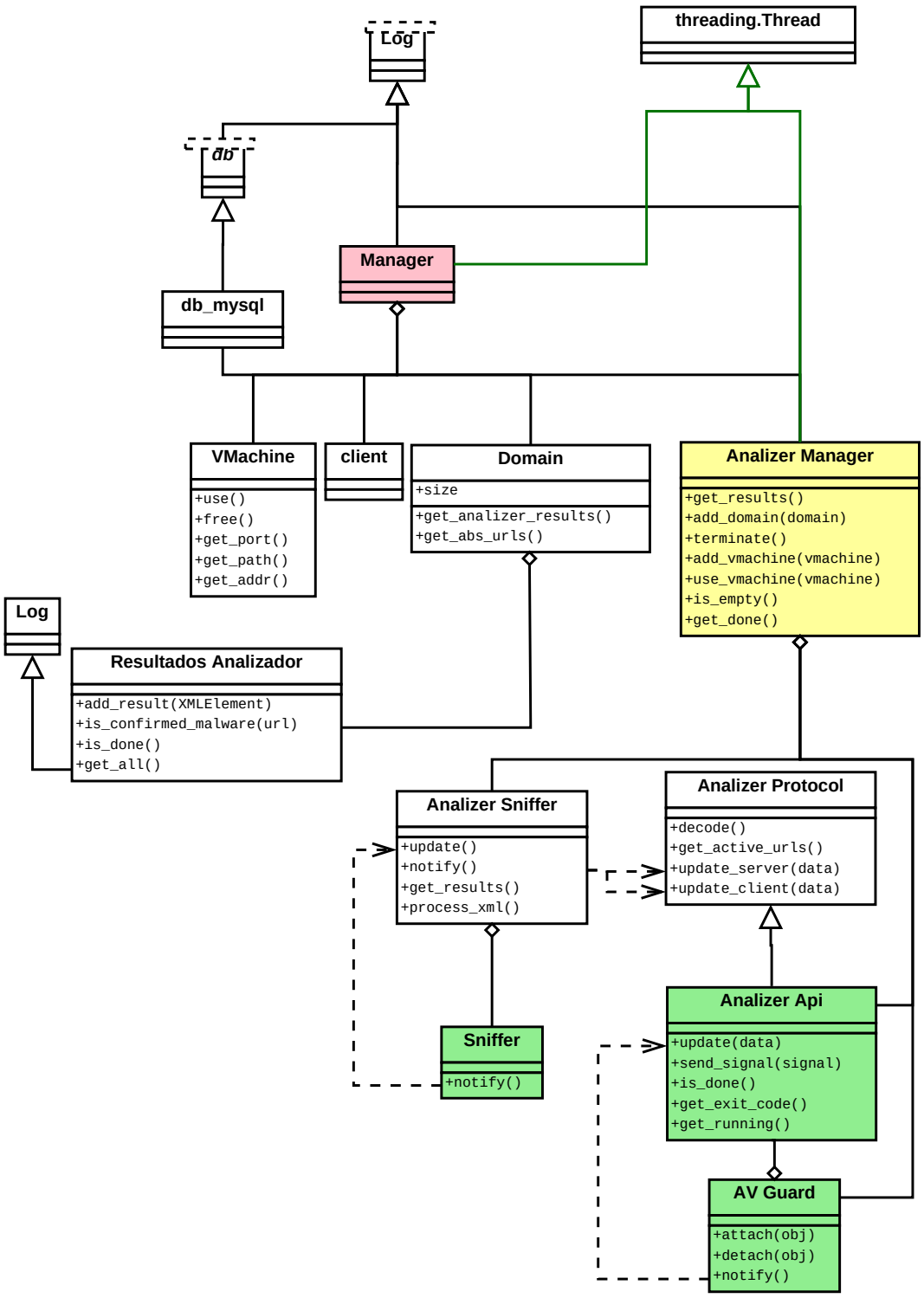


Figura 3.5: Diseño del gestor de HoneyClients

dominio ha finalizado, a diferencia del *crawling* en el que si lo hacía.

A continuación se definen los métodos expuestos:

- **add_result(XMLElement)**: Añade a los resultados el elemento *XML* facilitado.
- **is_confirmed_malware(url)**: Retorna True si la *URL* en cuestión ha sido la única analizada cuando se ha producido el evento del sistema.
- **is_done()**: Retorna *True* si los resultados ya están completos y *False* en caso contrario.
- **get_all()**: Retorna todos los resultados almacenados en el objeto como un *hash* de listas, donde cada *URL* finalizada corresponde a una clave.

3.4.3. Implementación

La implementación se ha finalizado exitosamente. Para interpretar correctamente el protocolo se ha utilizado un módulo llamado *xml.etree.ElementTree*, el cual permite la construcción de objetos *XML*, proporcionando comodidad para tratar y consultar la información.

Éste es proporcionado por el propio núcleo del sistema y permite una cómoda consulta de los datos mediante métodos intuitivos y sencillos de utilizar. El objeto *Analizer Sniffer* se encarga de ensamblar los fragmentos recolectados por el *sniffer* y construir un objeto *XML* utilizando la librería. Finalmente se envían los datos al *Analizer Protocol* para interpretarlos.

Para la función de *sniffer* se ha utilizado el módulo *pcap* de Python, éste no era facilitado por el propio núcleo del lenguaje y se ha descargado del paquete *python-libpcap* de *Debian*[3].

En un inicio el software parecía permanecer a la espera hasta que aparecía un nuevo paquete, pausando así toda la ejecución del sistema. La solución fue llamar al siguiente método:

```
pcapObject.setnonblock(1)
```

[3] *def*: Distribución de Linux, su web es: www.debian.com

Esta opción permite seguir ejecutando la aplicación aunque no lleguen paquetes ya que deshabilita el bloqueo.

Otra de las deficiencias detectadas consistía en el envío infinito de peticiones *Ping* del servidor al cliente sin ser respondidas. Este caso lo detecta directamente el objeto *Analizer Api*, mediante un sistema de control de los tiempos de respuesta entre cliente y servidor, acto seguido reinicia el proceso facilitando únicamente las *URL*s que aun no han sido analizadas, optimizando así el tiempo de ejecución.

3.5. Alertas

El sistema de alertas está muy ligado al gestor de *HoneyClients*, aunque al ser una funcionalidad diferente se comenta en una sección a parte.

3.5.1. Análisis de las Funcionalidades

El objetivo de las alerta es permitir conocer al cliente y al administrador del sistema cualquier tipo de acciones maliciosas detectadas en el portal lo antes posible y sin intermediarios.

Debido al procedimiento seguido por el software a la hora de concretar la *URL* autora de las acciones maliciosas, se deberá diferenciar entre los resultados que puedan ser *Falsos Positivos*, acciones detectadas cuando se analizan multitud de recursos, y los que realmente puedan ser *Posible Malware*, acciones detectadas cuando únicamente se analiza una *URL*.

Las alertas deberán enviarse a los destinatarios, éstos dependerán del dominio al que pertenezca la *URL* que genere la alerta.

3.5.2. Diseño

El resultado ha sido un diseño modular, en el que hay un gestor de alertas del que hereda *Analizer Protocol* y permite la funcionalidad de envío de correos electrónicos. Cuando se detecta un evento en el Analizador, éste rápidamente genera una instancia de alerta. éstas instancias son nodos independientes, los cuales esperan un tiempo prudencial, y envían la alerta.

Cada nodo espera un tiempo prudencial para reconocer cuando se trata de eventos que posiblemente serán *Falso Positivo* o bien *Posible Malware*. Por la naturaleza del proceso del software utilizado, puede ocurrir que las acciones maliciosas se lleven a cabo únicamente al analizar múltiples *URL*s y no posteriormente al concretar la causante.

Cuando se genera un evento en múltiples *URL*s, suponiendo 5, se generan tantas alertas como direcciones se estén analizando, 5 alertas en este caso. Cuando el software detecta eventos en el sistema, divide los recursos analizados en dos grupos y repite el proceso para cada uno de ellos. En caso de no aparecer ningún evento de *malware* con el primer grupo. Las 5 alertas se mantienen aun a la espera. Seguidamente se procesa el siguiente grupo. Si en este caso sí se detecta *malware*, se generan tantas alertas como *URL*s tenga el grupo y el gestor elimina las 5 anteriores. Si no se hubiera detectado *malware* en este último análisis, las 5 alertas seguirían activas y se enviarían al destinatario como *Posible Falso Positivo*. Siguiendo con el procedimiento, cuando finalmente se registraran eventos en una única *URL*, se eliminarían todas alertas anteriores y se generaría un único nodo, en este caso se trataría de un *Posible Malware*.

Diagrama

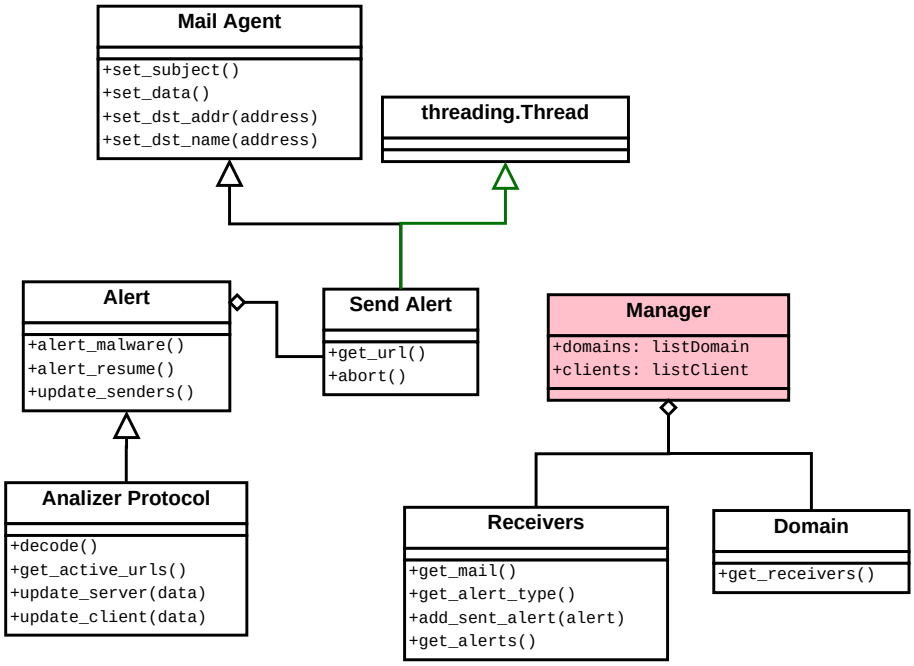


Figura 3.6: Diseño del sistema de alertas

En la *Figura-3.6* se puede observar el diagrama de clases del diseño propuesto e implementado. Como en el resto de éstos se ha marcado de color verde la herencia de *threading.Thread*.

El gestor principal mantiene una lista de todos los receptores de la base de datos y los distribuye por los dominios que se deben analizar. De esta manera cada dominio mantiene una lista con los receptores que tiene asignados para las alertas generadas y facilita la consulta de la información necesaria para el envío de correos.

Adicionalmente, cada receptor mantiene una lista con las alertas que le han sido enviadas, el gestor principal se encarga de ir almacenando estos datos en la base de datos automáticamente a medida que avanza la ejecución y se ejecutan las iteraciones del bucle principal.

3.5.3. Implementación

La implementación ha finalizado correctamente y se ha respetado el diseño expuesto. Para el envío de correos se ha utilizado el módulo *smtplib*[4]. incluido en el propio núcleo del sistema Python. Para que el lector se haga una idea de la comodidad que ofrecen los módulos al interactuar con servicios comunes, seguidamente se presenta una porción de código que permite enviar un correo electrónico utilizando una comunicación cifrada mediante *TLS*[5].

```
s = smtplib.SMTP(server_ip,port)
s.starttls()
s.login(user,password)
s.sendmail(mail_orig,mail_dest,data)
s.close()
```

Tan simple como eso.

3.6. Gestor General

En esta sección se describe el proceso llevado a cabo para el diseño del gestor general de la aplicación.

[4] *def*: Módulo de Python que facilita la interacción con los servidores de envío de correo electrónico.
[5] *def*: Sus siglas provienen del inglés *Transport Layer Security* que significa Seguridad de la Capa de Transporte y se trata de un protocolo criptográfico que proporciona comunicaciones seguras por una red.

3.6.1. Análisis de las Funcionalidades

El gestor principal deberá asegurarse que la ejecución transcurre con toda normalidad. Inicializando en cada ciclo los datos de todas las partes. Con el fin de distribuir correctamente los dominios por el conjunto de máquinas virtuales utilizables, se deberá diseñar un algoritmo que realice dicha distribución correctamente antes de comenzar la ejecución de los ciclos.

El funcionamiento básico del sistema se basa en la realización de 24 ciclos diarios, ejecutándose cada uno de ellos en una máquina virtual específica. Y eliminando la *cache* de algunos dominios al azar en cada uno de estos ciclos. Toda esta gestión deberá realizarse en este módulo.

Actualmente se envía al cliente un informe diario con el resultado del análisis, éste informe deberá generarse también automáticamente cuando finalicen los ciclos diarios. Detallando errores de conexión y/o qué acciones maliciosas se han llevado a cabo en el sistema durante el análisis.

3.6.2. Diseño

Para mantener un control exacto del proceso que se lleva a cabo en cada instante se ha diseñado una máquina de estados con un total de 5.

- **Inicio Crawling**: Se inicializan todas las variables. Si se trata del primer ciclo se calcula la distribución de dominios entre las máquinas virtuales. Seguidamente se cargan los dominios a analizar y se ordenan. Finalmente arranca el proceso de *crawling*.

- **Actualizar Crawling**: Se actualiza el estado del gestor de *crawlers* y se espera a que finalicen. Almacenando los resultados de cada dominio a medida que se completa el proceso.

- **Inicio Analizador**: Se arranca el analizador, inicializando las variables necesarias para el control del mismo: máquina virtual, *sniffer*, *analizer_api*...

- **Actualizar Analizador**: Se actualizan los datos del gestor del analizador, conociendo los recursos que se procesan en cada instante, los dominios que han ido finalizando, el estado en el que lo han hecho...

- **Report**: Si se han realizado todos los ciclos se recopilan los datos almacenados y se genera el informe que se enviará a los destinatarios, cliente. En caso contrario se sigue con el siguiente ciclo.

Además de estos estados perfectamente diferenciados, en cada iteración se actualiza la información referente a las alertas. El núcleo de la aplicación también funciona con un *thread* independiente, de esta manera se permite utilizarlo desde el intérprete cómodamente ya que no realiza una función bloqueante. Se permite así arrancar toda la plataforma y realizar llamadas en tiempo real a cada uno de los objetos del sistema, empezando por el gestor general.

Diagrama

En la *Figura-3.7* se muestra el diagrama del diseño del gestor principal, en el que se engloban los demás bloques descritos anteriormente. En este diagrama únicamente se han mostrado algunas de las clases que componen la aplicación por cuestiones de espacio y legibilidad.

Se han marcado con recuadros los objetos pertenecientes a cada uno de los grandes gestores, *crawling* y analizador. También se han mostrado los métodos creados en el gestor principal para una consulta del estado en tiempo real.

Algoritmo de distribución

El algoritmo de distribución sigue los pasos siguientes:

1. Se calcula el total de Máquinas Virtuales disponibles.
2. Se calcula las veces que se ejecutara el conjunto de máquinas virtuales durante los 24 ciclos, en caso de no ser exacto se adapta añadiendo al final la máquina virtual del inicio, por ejemplo.
3. Para cada prioridad:
    a) Se calcula el total de ciclos que deberán realizarse.
    b) Se calcula las veces que aparecerá en cada ciclo de máquinas virtuales, en caso de resultar un valor decimal se calcula también el residuo.

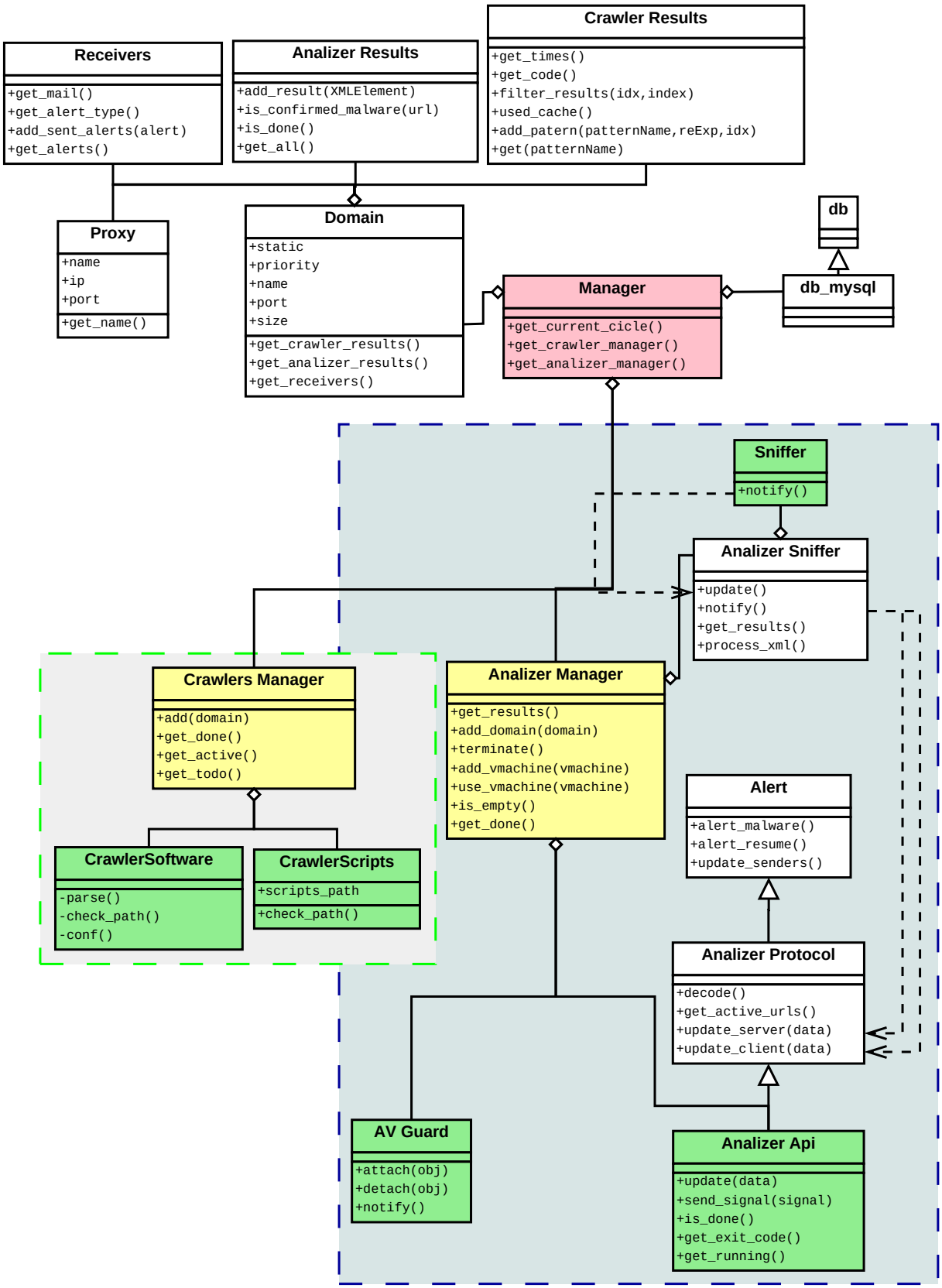


Figura 3.7: Diagrama general de la aplicación *(simplificado)*

c) Se distribuye de forma uniforme entre los ciclos de las máquinas virtuales tantas veces como se haya calculado.

d) En caso de haber un residuo, éste se distribuye por cualquier ciclo que no esté ocupado por esta prioridad, tomando preferencia aquellos que contengan una máquina virtual aun no ha asignada a la prioridad.

Se debe tener en cuenta que no se asignarán las prioridades de forma consecutiva, ya que un buen servicio de vigilancia debe hacerse distribuido en el tiempo, o de lo contrario se agotan los análisis en pocas horas y el dominio se mantiene una importante parte del día vulnerable a ataques y sin revisar.

A continuación se muestra una distribución retornada por este algoritmo:

```
TotalCiclos = 24
MaquinasVirtuales = [ ’A’, ’B’, ’C’, ’D’ ’E’, ’A’ ]
prioridad: 1: 6 veces por cada grupo de MaquinasVirtuales
prioridad: 2: 3 veces por cada grupo de MaquinasVirtuales
prioridad: 3: 2 veces por cada grupo de MaquinasVirtuales
prioridad: 4: 1 vez por cada grupo de MaquinasVirtuales

Ciclo: Prioridades de los dominios a analizar
’0’: [1, 2, 3, 4],
’1’: [1],
’2’: [1, 2, 4],
’3’: [1, 3],
’4’: [1, 2],
’5’: [1],
’6’: [1],
’7’: [1, 2, 3, 4],
’8’: [1],
’9’: [1, 2],
’10’: [1, 3],
’11’: [1, 2],
’12’: [1, 2],
’13’: [1],
’14’: [1, 2, 3, 4],
’15’: [1],
’16’: [1, 2],
’17’: [1, 3],
’18’: [1, 3]
’19’: [1, 2],
’20’: [1],
’21’: [1, 2, 3, 4],
```

```
'22': [1],
'23': [1, 2, 4],

Prioridades: MaquinasVirtuales sobre las que se ejecuta
'2': ['A', 'C', 'E', 'B', 'D', 'A', 'C', 'E', 'A', 'D', 'A', 'B']
'3': ['A', 'D', 'B', 'E', 'C', 'A', 'D', 'A']
'4': ['A', 'B', 'C', 'D', 'E', 'A']}
```

La distribución expuesta combina correctamente todas las prioridades con el conjunto de máquinas virtuales disponibles. Además, la distribución en el total de ciclos también ha resultado bien, el espacio que existe entre ellos corresponde al adecuado para la prioridad (*p.e.* prioridad 3 no se ejecuta 8 veces seguidas, si no que se distribuye entre el total).

3.6.3. Implementación

La implementación se ha realizado correctamente, respetando en todo momento el diseño expuesto y se han añadido las funcionalidades requeridas. Los métodos de consulta de los distintos gestores y el ciclo actual permite navegar por la totalidad de los datos. Bastará con ejecutar el interprete de Python una vez inicializada la plataforma para controlar en todo momento el estado de ésta.

Capítulo 4

Pruebas y resultados

En este capítulo se describe el conjunto de pruebas realizadas sobre la plataforma y sus resultados.

4.1. Entorno de verificación

Para el desarrollo de las pruebas, **Internet Security Auditors, S.L.** ha facilitado un entorno con las características siguientes:

- Sistema Operativo Linux con kernel 2.6.
- Servidor MySQL instalado y funcionando.
- Acceso a la red.
- Acceso remoto al entorno.
- Entorno de honeyclient instalado.
- Máquinas virtuales accesibles.
- Software de crawling instalado .
- Cuenta de correo para utilizarla como origen en las alertas.

4.2. Descripción de las pruebas

Debido al considerable tamaño de la plataforma, las pruebas se han ido realizando en cada uno de los módulos por separado y finalmente al unirlos se han hecho algunas más para comprobar el funcionamiento del conjunto.

4.2.1. Pruebas de Gestor de Crawling

Se han realizado múltiples pruebas con el fin de asegurarse que el gestor de *crawling* era robusto y no dejaba de funcionar ante cualquier adversidad. Al mismo tiempo que se ha comprobado la gestión correcta de todos los datos y la transmisión, al gestor principal, de los resultados.

Prueba: Se han añadido varios scripts de *crawling* incorrectos para comprobar el correcto funcionamiento del submódulo destinado a gestionar los scripts.

Resultado: Como resultado, se muestra un mensaje por el sistema de *logs* de la aplicación y devuelve un número negativo como código de finalización. Consecuentemente, queda constancia en la base de datos y en el registro del sistema, tal y como se muestra a continuación.

```
Sep  1 17:03:42 host [warning]: (script) Could not find script at:
/scripts/script_www.dom1.com. Aborting!
```

Prueba: Se han añadido dominios inexistentes o con problemas de conexión para comprobar el resultado devuelto por el gestor y el módulo de *crawling*.

Resultado: El resultado ha sido satisfactorio ya que permite diferenciar entre errores de resolución de nombres y errores de conexión. Devolviendo en ambos casos un valor negativo de distinta magnitud.
En el siguiente ejemplo de sentencias *log* se observa el mensaje descriptivo del error que ha ocurrido y, en caso de ser un valor sin descripción se muestra el valor de retorno.

```
Sep  9 20:43:13 host [warning]: (crawler) could not connect to
domain "www.dominio1.com" at port 80
Sep  9 22:00:49 host [warning]: (crawler) could not resolve
domain "www.dominio2.com"
Sep  9 22:01:40 host [warning]: (crawler) error on domain
"www.dominio3.com" returned: -9
```

Prueba: Se ha lanzado la señal *kill*[1] desde el sistema operativo a los procesos de *crawling* lanzados desde el gestor con el fin de que este hecho fuera almacenado en la base de datos y en el log del sistema.

Resultado: Como resultado, el módulo de crawler devuelve el valor de la señal que se le ha enviado desde el sistema operativo. La información se almacena en la base de datos y en los *logs* del sistema.

```
Sep  9 23:18:32 host [warning]: (crawler) error on domain
"www.dominio1.com" returned: -9
Sep  9 23:18:33 host [debug]: (db_mysql) logging crawler:
www.dominio1.com
```

Prueba: Se ha comprobado que el uso del *proxy* se realice correctamente, añadiendo uno y especificando su utilización en el gestor de crawlers.

Resultado: Como resultado el gestor indica a cada nodo de *crawling* que lanza que debe aplicar el *proxy* en la conexión. És el nodo el que se encarga de configurarlo en el *script* o software externo de *crawling*.

Prueba: Se ha comprobado que se utilicen correctamente las credenciales necesarias para el acceso al dominio. Para ello se han añadido varios que requerían autenticación *Basic*.

[1] *def*: El comando *kill* sirve para matar un proceso, al que se le envía la señal especificada por el valor valor del parámetro, -9 en el caso del ejemplo.

Resultado: El dominio mantiene almacenadas las credenciales que el gestor le ha asignado. Consecuentemente, el nodo de *crawling* es capaz de utilizarlos en sus peticiones. Como resultado se accede a los recursos especificando la siguiente sintaxis:

```
http://usuario:password@www.dominio.com/dir/recurso.html
```

Prueba: Se han añadido dominios estáticos, es decir, que no requieren el proceso de *crawling*, con el fin de comprobar si se lanzaría el procedimiento.

Resultado: El resultado ha sido el esperado y al detectar que se trata de un dominio estático el gestor lo retorna sin crear ningún nodo de *crawling* adicional y mandando el siguiente mensaje al sistema de *logs*:

```
 Sep 11 00:16:27 host [debug]: (crawler_manager) Static
domain: "www.dominio2.com" crawling aborted!
```

Prueba: Se ha intentado configurar un dominio con un módulo de *crawling* inexistente para comprobar que el gestor detectara el error.

Resultado: Como resultado se muestra una sentencia en el *log* del sistema con el siguiente aspecto:

```
 Sep 11 00:16:27 host [debug]: (crawler_manager) crawler
'crawler_inexistente' not supported! [www.dominio1.com]
```

En la linea se observa tanto el dominio afectada como el *crawler* especificado en la configuración, de esta manera tanto la corrección como la detección del error se convierten en un proceso trivial.

Prueba: Se ha comprobado que se genere una linea en el *log* cuando el tamaño de un recurso raíz varíe para avisar de modificaciones sobre el dominio.

Resultado: El resultado ha sido el esperado y al detectar la variación del tamaño se genera la siguiente linea en la que se muestra el dominio y la raíz que ha variado:

```
Sep  9 15:41:26 host [error]: (domain) root "/mapa" size
mismatch at 'www.dominio.com'
```

Todos los sucesos inesperados que han ocurrido durante las pruebas, tanto provocados

como circunstanciales se han almacenado correctamente, o en la base de datos o en el sistema de *logs* del mismo modo que se ha observado en las pruebas realizadas.

4.3. Pruebas del gestor del Analizador

Las pruebas a este gestor han sido algo más complejas, ya que algunos de los errores que se pretenden subsanar se producen muy raras veces y son muy difíciles de reproducir.

Prueba: Se han verificado los controles implementados en el gestor que evitarán iniciarlo sin una correcta configuración previa para evitar posibles errores.

Resultado: Como resultado la aplicación devuelve un mensaje de error tanto en el *stdout* como en el sistema de logs. Consecuentemente el administrador percibe que algo no funciona como debe y puede actuar en consecuencia. Por ejemplo:

```
Sep  3 17:29:21 host [error]: (analizer_manager) no VMachines
assigned to analizer manager, aborting!
```

Prueba: Se ha verificado que el procesamiento de los mensajes *XML* enviados entre cliente y servidor se realice adecuadamente. Para ello se han comprobado los resultados devueltos por el módulo *sniffer* con el debug del software *HoneyClient* y comprobar así que la reconstrucción era correcta.

Resultado: Como resultado se ha detectado que en algunos casos el proceso de transformar los datos a objetos *XML* en Python producían retrasos importantes en la interpretación del protocolo. Por esta razón se han ignorado algunos mensajes carentes de información útil para el control de eventos, como puede ser el envío de las listas de exclusión e inclusión. Realizada esta pequeña modificación el resultado es totalmente satisfactorio y se permite un control total del protocolo seguido.

Prueba: Se ha provocado el error descrito en la sección de deficiencias del software *HoneyClient*[2]. El objetivo era comprobar que se detectaba correctamente por la plataforma y se reiniciaba el proceso en el punto que estaba cuando ocurrió el error.

Resultado: El resultado es totalmente satisfactorio, la plataforma detecta justo el instante en el que se produce el error y actúa en consecuencia sin problema alguno durante el proceso de recuperación.

[2] *def*: El servidor permanece indefinidamente enviando peticiones *Ping* sin recibir ningún tipo de respuesta y sin realizar ninguna acción, bloqueando así la ejecución

A continuación se presenta el log que se imprime cuando ocurre se produce este error:

```
Sep 11 00:16:27 host [debug]: (analizer_results) www.d1.com: 2/2
Sep 11 00:16:27 host [debug]: (analizer_results) www.d2.com: 5/13
Sep 11 00:16:27 host [debug]: (analizer_results) www.d2.com: 6/13
Sep 11 00:16:27 host [debug]: (analizer_results) www.d2.com: 7/13
Sep 11 00:16:27 host [debug]: (analizer_results) www.d2.com: 8/13
Sep 11 00:16:27 host [debug]: (analizer_api) scan FINISHED malicious: 0
Sep 11 00:16:27 host [debug]: (db_mysql) logging analizer:  www.d1.com
Sep 11 00:16:27 host [debug]: (analizer_api) scan request
Sep 11 00:16:27 host [debug]: (analizer_api) type: start
Sep 11 00:16:27 host [debug]: (analizer_api) scan request validated
Sep 11 00:17:33 host [warning]: (analizer_api) ping response is taking so
long or client innactivity! analizer should be rebooted!
Sep 11 00:17:38 host [warning]: (analizer_api) proccess killed!
Sep 11 00:18:10 host [debug]: (analizer_api) domain www.d1.com
analizer_results is done (skipped)
Sep 11 00:18:11 host [debug]: (analizer_api) domain www.d2.com
analizer_results is done (added)
Sep 11 00:18:11 host [debug]: (analizer_api) urls to analize: 5
Sep 11 00:18:11 host [debug]: (analizer_api) 'current_state' error,
something happened. State '2' should be '0' or '3'
Sep 11 00:18:30 host [debug]: (analizer_api) scan request id: 2024961154
Sep 11 00:18:30 host [debug]: (analizer_api) type: start
Sep 11 00:18:31 host [debug]: (analizer_api) scan request validated,
id: 2024961154
Sep 11 00:19:23 host [debug]: (analizer_api) type: finish
Sep 11 00:19:24 host [debug]: (analizer_results) www.d2.com: 9/13
Sep 11 00:19:24 host [debug]: (analizer_results) www.d2.com: 10/13
```

Se observa claramente el funcionamiento. Después de procesar los resultados de *www.d1.com* y *www.d2.com*, el gestor almacena los datos referentes a *www.d1.com* que ya ha terminado. Seguidamente el analizador se bloquea por el fallo ya descrito y el gestor procede a reiniciar el software. Cuando vuelve a cargar las *URL*s a analizar, no agrega las que ya han sido analizadas anteriormente, en consecuencia la recuperación no supone una gran pérdida de tiempo de análisis.

Prueba: Se han generado a conciencia algunos eventos maliciosos en el sistema para comprobar que el *sniffer* capturaba correctamente todos los eventos y el proceso de envío de alarmas se comportaba como debía.

Resultado: Como resultado se reciben multitud de correos electrónicos en los que se indica correctamente si se trata de un *Posible Falso Positivo* o un *Posible Malware*. En ambos casos se detallan a la perfección las acciones realizadas sobre el sistema.

A continuación se muestran los datos que se envían en estos correos, se presentan sin el formato adecuado únicamente para entender el tipo de datos que se recogen:

```
Subject: Posible Falso Positivo en 'www.dominio.net'

The next url: "http://www.dominio.net:80/dir/SConsNoticia?ID_IDIOMA=ca
&ope=1&clau=NO01056&total=14&member=6" may be infected! Domain:
www.dominio.net

Resultado Analizador:
  malicious: 0
  program: internet explorer
  time: 7/9/2009 17:2:3.341
  visited: 1

Traza retornada:

  processId: 2672
  process: C:\Archivos de programa\Real\RealPlayer\realplay.exe
  object: HKCU\Software\Microsoft\Internet Explorer\Main\FullScreen
  time: 7/9/2009 16:55:17.999
  action: SetValueKey
  type: registry

  processId: 2672
  process: C:\Archivos de programa\Real\RealPlayer\realplay.exe
  object: HKCU\Software\Microsoft\Internet Explorer\Main\Window_Placement
  time: 7/9/2009 16:55:17.999
  action: SetValueKey
  type: registry

  processId: 2672
  process: C:\Archivos de programa\Real\RealPlayer\realplay.exe
  object: C:\Documents and Settings\Administrador\Datos de programa\
Real\RealPlayer\browserrecord.swf
  time: 7/9/2009 16:55:18.827
  action: Write
  type: file
```

En el mensaje de correo expuesto se observa el nivel de detalle que se obtiene del software *HoneyClient* y como és correctamente procesado y asociado con el dominio al que pertenece mediante la plataforma automatizada. En este caso se trata de un *Posible Falso Positivo*, si fuera un *Posible Malware*, se modificaría el *subject* del correo, pero el aspecto seguirá siendo el mismo.

Prueba: Se ha comprobado que el sistema de alertas reaccione de forma diferente para los usuarios que desean recibir una alerta por cada recurso infectado en tiempo real, o una alerta global del dominio una vez finalizado.

Resultado: La plataforma se comporta como debe y envía correctamente el resumen de acciones detectadas en un dominio y para cada recurso, dependiendo de cual sea el caso del receptor. Seguidamente se muestra el correo resumen del domínio:

```
Subject:Resumen Resultados www.dominio.net


Resultado Crawler:
    new: 10
    untouched: 2
    exit code: 0

Resultado Analizador:

    urls: 10
    time: 73 sec
    posibleMalware: 0 urls
    posibleVirii: 0 urls
    posibleBlackListDomains: 0 urls
    falsesPositives: 5
    TotalSentAlerts: 5

Traza retornada:

URL: "/directorio/index.php?lang=es"

  processId: 2672
  process: C:\Archivos de programa\Real\RealPlayer\realplay.exe
  object: HKCU\Software\Microsoft\Internet Explorer\Main\FullScreen
  time: 7/9/2009 16:55:17.999
  [..]
```

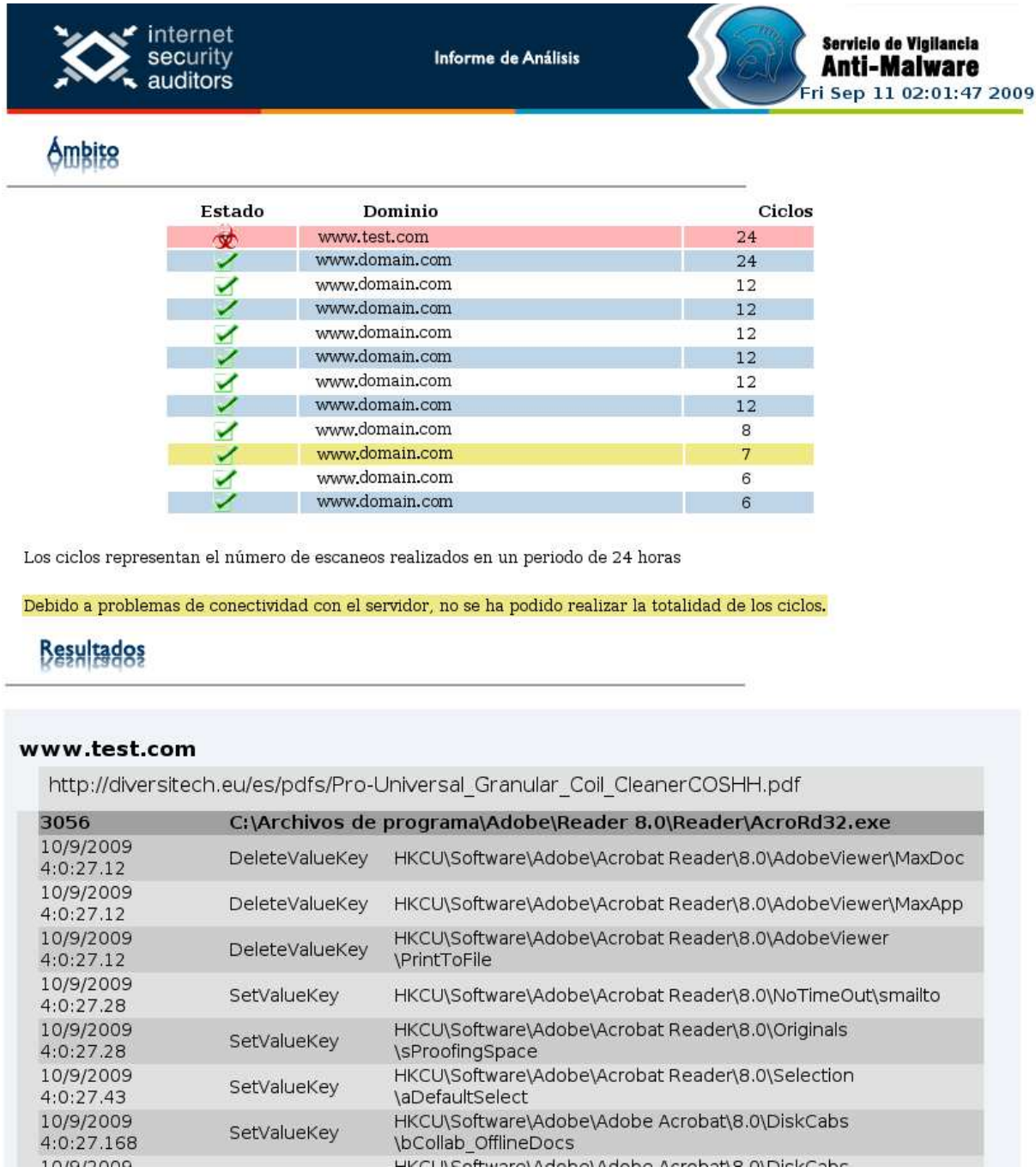
Prueba: Se ha comprobado que la asignación de *URL*s procesadas por el analizador sea correspondida con los dominios analizados de forma correcta.

Resultado: Cuando la plataforma detecta una petición de análisis para varios recursos, el módulo del analizador asocia esas *URL*s al dominio al que pertenecen, en caso de no ser posible dicha asociación se muestra un mensaje de error en el sistema de *logs* como el siguiente:

```
Sep  9 19:46:16 host [warning]: (analizer_api) unknown domain for url:
http://www.dominio.net:80/dir2/AriaMat?ID=ca&term=La%20pura
```

4.4. Pruebas del gestor principal

Después de comprobar el correcto funcionamiento de todos los módulos anteriores, le toca el turno a al gestor principal. Seguidamente se describen las pruebas realizadas y los resultados obtenidos.


internet security auditors

Informe de Análisis

Servicio de Vigilancia Anti-Malware

Fri Sep 11 02:01:47 2009

Ámbito

Estado	Dominio	Ciclos
	www.test.com	24
	www.domain.com	24
	www.domain.com	12
	www.domain.com	12
	www.domain.com	12
	www.domain.com	12
	www.domain.com	12
	www.domain.com	12
	www.domain.com	8
	www.domain.com	7
	www.domain.com	6
	www.domain.com	6

Los ciclos representan el número de escaneos realizados en un periodo de 24 horas

Debido a problemas de conectividad con el servidor, no se ha podido realizar la totalidad de los ciclos.

Resultados

www.test.com

http://diversitech.eu/es/pdfs/Pro-Universal_Granular_Coil_CleanerCOSHH.pdf

3056	C:\Archivos de programa\Adobe\Reader 8.0\Reader\AcroRd32.exe	
10/9/2009 4:0:27.12	DeleteValueKey	HKCU\Software\Adobe\Acrobat Reader\8.0\AdobeViewer\MaxDoc
10/9/2009 4:0:27.12	DeleteValueKey	HKCU\Software\Adobe\Acrobat Reader\8.0\AdobeViewer\MaxApp
10/9/2009 4:0:27.12	DeleteValueKey	HKCU\Software\Adobe\Acrobat Reader\8.0\AdobeViewer \PrintToFile
10/9/2009 4:0:27.28	SetValueKey	HKCU\Software\Adobe\Acrobat Reader\8.0\NoTimeOut\smailto
10/9/2009 4:0:27.28	SetValueKey	HKCU\Software\Adobe\Acrobat Reader\8.0\Originals \sProofingSpace
10/9/2009 4:0:27.43	SetValueKey	HKCU\Software\Adobe\Acrobat Reader\8.0\Selection \aDefaultSelect
10/9/2009 4:0:27.168	SetValueKey	HKCU\Software\Adobe\Adobe Acrobat\8.0\DiskCabs \bCollab_OfflineDocs

Figura 4.1: Informe final que se enviará al cliente.

Prueba: Se ha comprobado la correcta finalización de los gestores de *crawling* y analizador. Para ello se ha realizado una modificación en el código que muestra el contenido de cada uno de estos justo antes de empezar el siguiente proceso.

Resultado: El resultado ha sido satisfactorio, los objetos se encuentran totalmente vacíos cuando finalizan su ejecución, con lo que no se producen fugas de memoria ocupada.

Prueba: Se han añadido los dominios de producción para comprobar el correcto análisis de todos ellos y la gestión de los resultados.

Resultado: El proceso funciona a la perfección, realizando primero el *crawling* a los dominios y ejecutando el analizador posteriormente. Los resultados se obtienen en tiempo real y se almacenan cuando termina la ejecución de cada uno de los dominios.

```
Sep 17 16:34:45 host [info]: (main_manager) ciclo: 8
Sep 17 16:34:47 host [debug]: (db_mysql) logging crawler: www.test.com
Sep 17 16:34:50 host [debug]: (analizer_api) domain www.test.com (added)
Sep 17 16:34:50 host [debug]: (analizer_api) urls to analize: 10
Sep 17 16:35:30 host [debug]: (analizer_api) scan request id: 3608943
Sep 17 16:35:30 host [debug]: (analizer_api) type: start
Sep 17 16:35:30 host [debug]: (analizer_api) scan request validated, id: 3608943
Sep 17 16:36:10 host [debug]: (analizer_api) scan FINISHED malicious: 0
Sep 17 16:36:10 host [debug]: (analizer_results) www.test.com: 1/10
[..]
Sep 17 16:36:46 host [debug]: (analizer_results) www.test.com: 10/10
Sep 17 16:36:46 host [debug]: (analizer_api) scan FINISHED malicious: 0
Sep 17 16:36:47 host [debug]: (db_mysql) logging analizer: www.test.com
Sep 17 16:36:48 host [debug]: (analizer_manager) killing completed analizer!
Sep 17 16:36:50 host [info]: (main_manager) ciclo: 9
```

Prueba: Se ha comprobado la sincronización de las alertas enviadas con la base de datos con el fin de mantener la información relacionada a cuando y qué alertas se han enviado.

Resultado: La plataforma almacena todos los datos correctamente en cada iteración del gestor principal. El cual espera un segundo entre cada una de ellas.

Prueba: Se ha comprobado la generación correcta de los datos necesarios para generar el informe, aunque los ciclos ejecutados sean variables, además de verificar una correcta construcción y posterior envío del mismo.

Resultado: Cuando llega la hora límite o se terminan los ciclos a realizar, se genera en la base de datos una marca que relaciona los últimos ciclos con el informe que se deberá generar para el cliente. De esta manera quedan segmentados los resultados dependiendo del informe al que pertenezcan.

Los informes generados que se envían tiene el aspecto mostrado en la *Figura-4.1*. Los nombres de los dominios han sido modificados por cuestiones de confidencialidad exceptuando *www.test.com*. El recurso listado no corresponde al dominio *www.test.com*, pero para realizar las pruebas se ha creado un dominio cuyas raíces son páginas con *malware* o en las que se cargan varios *plug-ins*.

En este caso, todos los eventos que se pueden ver en el informe son inofensivos, ya que se trata de el proceso de carga del *plug-in* de *Adobe Reader* para lectura de *PDF*s. El siguiente paso sería filtrar estos resultados para que no aparezcan en el siguiente informe.

Prueba: Se ha comprobado que los filtros añadidos en la configuración filtren realmente los eventos detectados del sistema y estos no aparezcan en el informe.

Resultado: Cuando se recolectan los datos para el informe, se cargan también los filtros existentes y se utilizan al mostrar los resultados. De esta manera nunca aparecen en el informe final, sin embargo, siguen almacenados en la base de datos por si fueran necesarios

Prueba: Se ha comprobado que la plataforma pueda ejecutarse de forma indefinida con totalidad autonomía.

Resultado: El resultado ha sido totalmente satisfactorio ya que la plataforma se ha ejecutado ininterrumpidamente durante varios días, recuperándose automáticamente de cada error ya detectado en la versión anterior

Prueba: Se ha comprobado que la plataforma no bloquee el sistema al consumir todos sus recursos. Para ello se ha visualizado la carga de memoria y *CPU* del proceso.

Resultado: Mediante el comando de *top*[3] se ha listado el proceso en cuestión y este ha sido el resultado:

```
PID   USER      PR  NI  VIRT  RES  SHR S %CPU %MEM
22461 user1     15   0  149m  62m  50m S   60  1.2
```

Se observa una carga de memoria de un 1.2 % lo que no supone nada importante. En cuanto a la carga de la *CPU*, en cambio, si que es notablemente mayor, sin embargo, no supone ningún problema para seguir utilizando el sistema de forma habitual, partiendo de la base que se trata de un servidor dedicado y no un sistema de escritorio. Se puede decir que el resultado es bastante bueno teniendo en cuenta la gran cantidad de operaciones internas que realiza y la densidad de datos con las que se trabaja.

4.5. Resumen de los resultados y valoraciones generales del conjunto

Por las pruebas que se han realizado hasta el momento, la plataforma se comporta tal y como se esperaba. Aunque no se trate del software perfecto, lo cierto es que realiza su función correctamente y supone una gran mejora sobre la plataforma anterior. Se ha conseguido solucionar la totalidad de las deficiencias, y en caso de que algún error no controlado ocurriera, el mensaje generado en el sistema de *logs* facilitará en gran medida la búsqueda del fallo y su arreglo.

La automatización implementada tiene en cuenta todos los problemas posibles que puedan surgir en el software gestionado, *crawling y analizador*, o, almenos, los que han sido detectados hasta la fecha. Además de haber reducido notablemente la intervención humana en toda la gestión de este sistema.

Los algoritmos utilizados para distribuir las máquinas virtuales entre los diferentes dominios de manera que todos sean ejecutados en cada una de ellas es más eficiente que el anterior. Ya que podía ocurrir que no se ejecutara un dominio por todas las máquinas virtuales.

La integración de todos los datos en la base de datos ha supuesto una gran comodidad

[3] *def*: Muestra los procesos del sistema y algunos detalles sobre su ejecución.

en cuanto a la consulta de los resultados. Se mantiene un control milimétrico de los diferentes procesos a los que se somete un dominio durante el análisis, así como un histórico de los ciclos e informes que se han realizado. Además se permite el acceso a la información desde infinidad de tecnologías, por ejemplo, para añadir un dominio al proceso se podrá utilizar una aplicación en lugar de conectarse al servidor por *ssh* y configurar infinidad de ficheros como se hacía en la versión anterior. Se podrán realizar *backups* remotos fácilmente, enviar los datos del *syslog* a sistemas del cliente... En definitiva, muchas posibilidades de acceso totalmente independientes entre si.

En resumen, la centralización de todo lo que conlleva un análisis de este tipo en una única plataforma era algo necesario y esta solución así lo hace. Además permite el control en tiempo real de forma remota gracias al intérprete de Python, por ejemplo, para lanzar toda la plataforma de forma habitual bastará con el siguiente código:

```
$ python
Python 2.6.1 (r261:67515, Mar 12 2009, 23:47:00)
[GCC 4.1.3 20080704 (prerelease) (Debian 4.1.2-25)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from main_manager import *
>>> hc = main_manager()
>>> hc.start()
>>>
```

Después ya se permite utilizar el objeto *hc* para consultar los datos como el ciclo por el que va actualmente o que máquina virtual se está utilizando, su nombre... Se tiene acceso a toda la aplicación:

```
>>> hc.cicle
20
>>> vm = filter(lambda x: x.in_use() == True, hc.vms)
[<vmachine.vmachine instance at 0x8499f2c>]
>>> vm.name
'IE6-WinXP'
>>>
```

Gracias a esta interfaz no ha sido necesario implementar un terminal de control interactivo para el administrador, Python lo ha proporcionado.

Capítulo 5

Conclusiones

El proyecto ha resultado tal y como se esperaba. Ha concluido como una base sólida sobre la que el servicio de detección de *malware* se ejecutará de forma automatizada en un entorno de producción que ya es lo que se esperaba.

El control minucioso de toda acción realizada por el software de detección ha permitido subsanar todas las deficiencias detectadas en la versión anterior del producto. El diseño realizado facilita la manipulación de los puntos conflictivos de forma cómoda y sin necesidad de reescribir multitud de código en diferentes ficheros y lenguajes.

La información recopilada por esta plataforma ayuda a un seguimiento preciso de todas las acciones ocurridas y el momento en el que se han producido. Además el hecho de centralizar toda la información en una base de datos permite implementar multitud de aplicaciones de consulta cómodamente, ya que únicamente deberán comunicarse con la base de datos de manera independiente a como lo hace la plataforma desarrollada en Python.

Aunque no aparezca en la memoria por cuestiones de confidencialidad, se han documentado también unas guías de uso interno para los administradores de la plataforma. De manera que se les permita actuar cuando se requieran modificaciones o ampliaciones con la información adecuada y precisa que se necesita.

5.1. Experiencia Personal

Ha sido una experiencia muy grata el poder contar con al confianza de **Internet Security Auditors, S.L.** para el desarrollo de un producto que facilitará notablemente el

funcionamiento del servicio ofrecido.

Ha sido muy enriquecedor el poder planificar, diseñar e implementar el producto de forma individual, ya que ha sido todo un reto el cuadrar fechas y encontrar soluciones a todos los problemas y necesidades requeridos por el proyecto, algunos detectados desde el inicio y otros desarrollados sobre la marcha. Es importante decir que han habido desviaciones del planning inicial por diferentes motivos, algunos personales, otros académicos y otros, errores de planificación debidos a mi inexperiencia para planificar un proyecto de tal magnitud en tiempo. Por ejemplo, las pruebas sobre el módulo analizador llevaron bastante más tiempo del supuesto en un inicio. En este punto tengo que añadir que la empresa me ha dado total libertad de dimensionamiento, con lo que podía retrasarme sin problema si era necesario.

Me alegra haber elegido el lenguaje Python como herramienta para implementar la plataforma, ya que, aun sin haberlo utilizado nunca antes, ha facilitado mucho el desarrollo de varias soluciones y funcionalidades requeridas gracias al sistema de módulos. Además, la facilidad de programación ha permitido plagiar a la perfección el diseño de clases realizado a mano directamente al código.

5.2. Evolución Futura

Gracias a esta nueva plataforma, se permitirá realizar ampliaciones de forma más cómoda que anteriormente. El manejo de la información se debe llevar a cabo mediante una aplicación web que interactuará de forma directa con la base de datos. Dejando la información necesaria para que el software de la plataforma implementado en Python recoja los datos y los utilice en las posteriores ejecuciones.

Cuando el producto se encuentre en producción se pondrá en funcionamiento una página web sobre la que se podrán consultar todos los datos en tiempo real y navegar por el historial de ciclos realizados. Conociendo en cada uno de ellos como ha transcurrido el proceso de crawling y analizador y el código de finalización de los mismos.

En un futuro, no se descarta desarrollar aplicaciones orientadas a la navegación mediante dispositivos móviles, utilizando para ello tanto aplicaciones implementadas para el propio sistema operativo del dispositivo o bien una web diseñada para tal fin.

Bibliografía

[1] Documentación oficial de Python, Septiembre 2009.
<http://docs.python.org/>

[2] Documentación oficial de MySQL, Abril 2009 - *en inglés*.
<http://dev.mysql.com/doc/>

[3] Manuales del sistema Linux, Septiembre 2009.
<http://linuxmanpages.com/>

[4] Software de virtualización VMWare, 2009 - *en inglés*.
<http://www.vmware.com/>

[5] Página con multitud de información sobre *HoneyPot*s, 2009 - *en inglés*.
<http://www.honeypots.net/>

[6] Página oficial de proyectos *HoneyNet*, 2009 *en inglés*.
<https://projects.honeynet.org/>

[7] Página oficial de MITRE, Proyecto *HoneyClient* - *en inglés*.
<http://www.honeyclient.org/>

[8] Software de virtualización *Qemu*, 2009 - *en inglés*.
<http://www.qemu.org/>

[9] Software de crawling *GNU/Wget*, 2 de Julio de 2007 - *en inglés*.
<http://www.gnu.org/software/wget/>

[10] Software de crawling *HTTrack*, 2009 - *en inglés*.
<http://www.httrack.com/>

Firmante: Ferran Pichel Llaquet

Bellaterra, 16 de Septiembre de 2009

Resum

Es descriu el disseny i posterior implementació de la nova plataforma d'automatització del servei ofert per Internet Security Auditors, S.L. destinada a l'anàlisi de dominis d'Internet amb la finalitat de detectar possibles infecciones que afectin a usuaris de la web. El sistema actual conté algunes deficiències, de manera que aquest text presenta una nova versió, la qual aporta millores molt significatives com ara una gestió més òptima, o un disseny renovat i escalable de la informació i els diferents processos. Així mateix es dota al sistema d'un control d'errors centralitzat, amb enviament d'alàrmes en temps real, i una agrupació i centralització dels resultats.

Resumen

Se describe el diseño e implementación de la nueva plataforma de automatización del servicio ofrecido por Internet Security Auditors, S.L. destinado a analizar dominios de Internet con el fin de detectar posibles infecciones que afecten a los usuarios de la web. El sistema actual tiene algunas carencias, de manera que este texto presenta una nueva versión, que aporta mejoras muy significativas como son una gestión más óptima, con un diseño renovado y escalable de la información y los diferentes procesos. Asimismo se dota al sistema de un control de errores centralizado, con envío de alarmas en tiempo real, y una agrupación y centralización de los resultados.

Abstract

This paper describes the design and implementation of the new automatic platform service offered by Internet Security Auditors, SL. It is designed to analyze Internet domains in order to detect possible infections that could affect the user's system while browsing the web. The current system has some shortcomings and this paper presents a new version which provides significant improvements such as optimal management, with a renewed design in the management of the information and processes. It also gives the system a centralised error handling, with a real-time alarm delivery, and results in grouping and pooling.