Universidad Zaragoza
1542

Trabajo Fin de Grado

Integración de metodologías de seguridad en entornos de desarrollo web

Integration of security methodologies in web development environments

Autor

Samuel Garcés Marín

Director/es

Enrique Manuel De Meer Alonso

Ponente

José Luis Salazar Riaño

Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación.

Curso 2017/2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Samuel Garcés Marín,

con nº de DNI 73417144V en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Integración de metodologías de seguridad en entornos de desarrollo web,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 19 de Junio de 2018

Fdo: Samuel Garcés Marín

RESUMEN

Este trabajo de Fin de Grado se ha realizado en colaboración con la empresa Instrumentación y Componentes S.A. donde se propone la integración de metodologías de seguridad en el departamento de desarrollo de software y más concretamente en los proyectos relacionados con desarrollo web.

El trabajo aborda el análisis de los ciclos de desarrollo del departamento, el estudio de las soluciones implementadas por otras empresas y posteriormente, la implementación e integración de una solución a medida de las necesidades actuales, dejando abierta la puerta para futuras mejoras.

Tras conocer el funcionamiento interno del departamento se comienza a buscar formas de poder integrar análisis de seguridad que no entorpezcan la labor de los desarrolladores.

Esto ha llevado al desarrollo de una plataforma de seguridad continua, tolerante a fallos, extensible por medio de módulos, escalable, tanto horizontalmente como verticalmente, y segura de cara a la exfiltración de información.

Tabla de contenido

Índice de Figuras

Índice de Tablas

Índice de Diagramas

1.Introducción

Este trabajo de Fin de Grado se ha llevado a cabo en la empresa Instrumentación y Componentes S.A. (Inycom) y el departamento de desarrollo de esta.

Inycom es una empresa tecnológica con 35 años de actividad empresarial y consta de diferentes sectores, que son: tecnologías de la información y comunicación (TIC), instrumentación analítica para laboratorio, equipamiento de diagnóstico para entornos hospitalarios e instrumentación electrónica para test y medida.

El proyecto busca satisfacer la creciente necesidad por parte de la empresa de dar salida a proyectos web con un mayor grado de seguridad, aumentando el valor añadido de los mismos de cara a los clientes.

En definitiva, su objetivo es poder aplicar nuevas metodologías de seguridad e implementarlas en proyectos que llevan años de recorrido a sus espaldas y que no permiten modificaciones en sus ciclos de entrega, mejorar la técnicas de seguridad actuales; así como preparar el terreno para la integración de seguridad desde la concepción en futuros proyectos.

Como recomendación, se insta al lector a leer primero la lista de términos propios (ANEXO A: Términos Propios), utilizados en el contexto del desarrollo de este TFG, para evitar confusiones con sus homónimos utilizados en un contexto más general.

1.1 Motivación

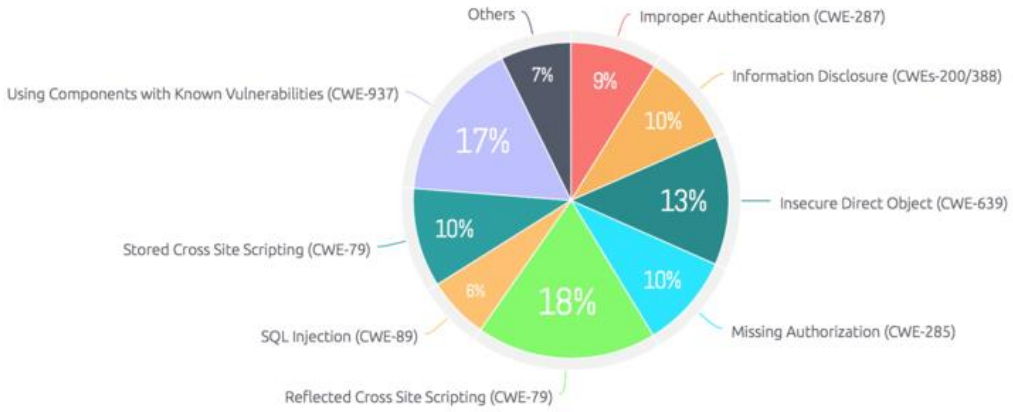


Figura 1 Distribución de vulnerabilidades (Vantagepoint, 2016)

Hoy en día la cantidad de ciberataques dirigidos hacia páginas web ha aumentado, debido principalmente a la modularidad de las páginas web, estando formadas por muchos componentes y librerías de código abierto, siendo este, el caso de Inycom. Si tomamos como norma [1] que una página web tiene una

media de entre 20 y 30 errores cada 1000 líneas de código, podemos hacernos una idea de la cantidad de posibles vulnerabilidades, siendo las más comunes [2] [3] las inyecciones de código SQL, el *Cross Site Scripting* (XSS) o el uso de componentes vulnerables, como se puede ver en la Figura 1.

Estos componentes normalmente son de código abierto y creados por varias personas o empresas haciendo que cualquier desarrollador pueda encontrar una vulnerabilidad o explotar las vulnerabilidades publicadas utilizando métodos, también llamados *exploits*, obtenidos de páginas como *Exploit Database,* Figura 2.

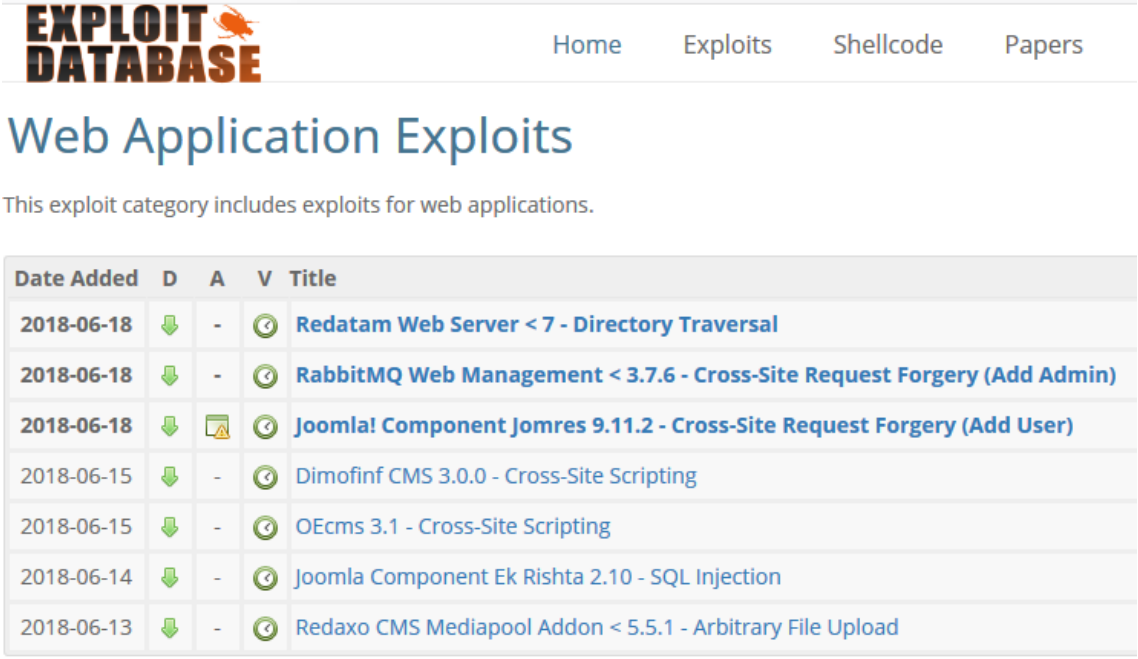


Figura 2 Publicación diaria de exploits

Es por ello por lo que el departamento de desarrollo de software de la empresa ve la necesidad de incorporar nuevas técnicas de seguridad en sus procesos de desarrollo, teniendo en cuenta la limitación impuesta al tener que trabajar al ritmo que marcan los clientes y con tiempos de entrega finales muy reducidos, debiéndose conseguir un compromiso en cuanto a seguridad, fiabilidad y funcionalidad, sin aumentar en exceso los costes de desarrollo.

Intentar integrar nuevas metodologías para mejorar la seguridad en estos proyectos, que hacen uso, en muchos casos, de desarrollos en cascada, puede llevar a retrasos indeseados asociados a la modificación de la metodología de trabajo, si algo funciona hay que tener cuidado al modificarlo; además, por las características de los mismos, no necesitan modelos de entrega continua o integración continua [4] [5], también llamados CI/CD, que hoy en día es lo habitual. Diseñaremos una mejora en el proceso de desarrollo que incorpore la seguridad desde el primer momento.

Por otro lado, los proyectos más recientes tienen sus propios entornos de desarrollo, preproducción y producción en la nube, principalmente en *Amazon Web Services*. Estos servicios en la nube tienden a facilitar la labor a la hora de desplegar las infraestructuras, pudiendo aliviar la carga de trabajo del departamento de sistemas y delegarla en el de desarrollo, lo que permite agilizar estos procesos de despliegue y entrega, acortando los tiempos de días a horas. Esto añade una problemática extra, en lo que a seguridad se refiere, pues los desarrolladores no están especializados en la administración de sistemas, haciendo que una mala configuración (Figura 4) o la no instalación de parches, pueda ser explotada por atacantes [6].

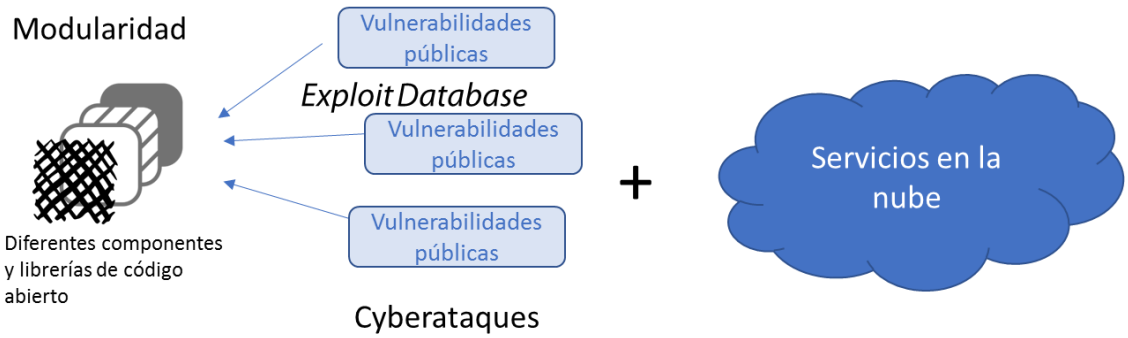


Figura 4 Vulnerabilidades en componentes y servicios

Como resultado del proyecto, se presentará una plataforma pensada exclusivamente para seguridad, que permita crear modelos de amenazas para cada proyecto, así como la realización de análisis de seguridad automatizados, informando debidamente a los responsables para actuar en consecuencia; todo ello por medio de una interfaz gráfica fácil de utilizar, como se puede apreciar en la Figura 3.

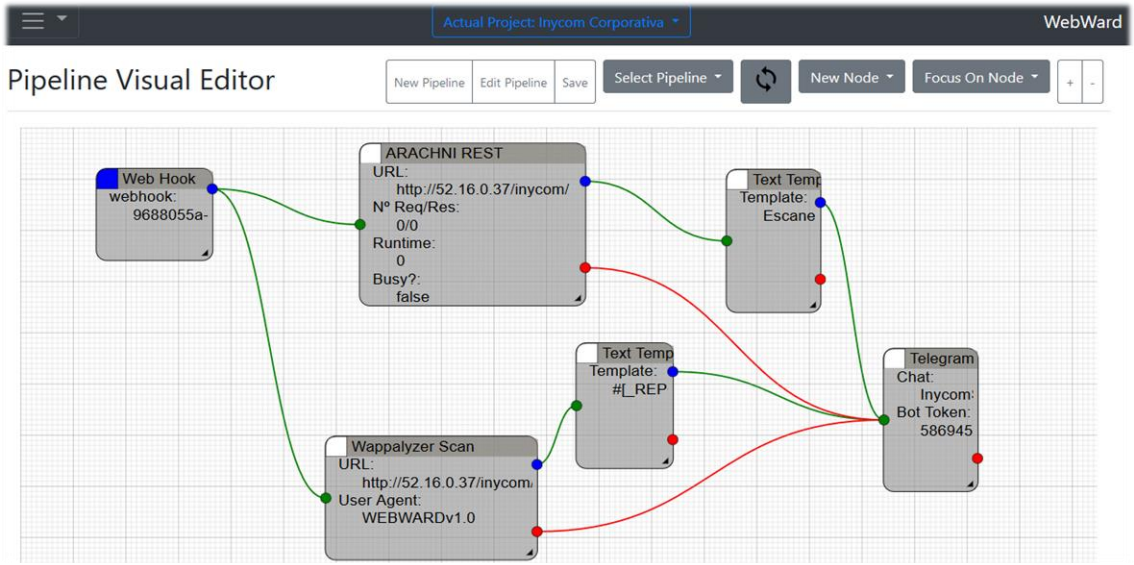


Figura 3 Aspecto final de la plataforma

1.2 Objetivos

Las diferentes tareas que se van a realizar asociadas a este trabajo fin de grado son:

- El estudio de las necesidades de seguridad informática de cada uno de los proyectos, el planteamiento de posibles necesidades futuras y la búsqueda de una solución que satisfaga cada una de ellas o pueda llegar a hacerlo con el tiempo.
- Investigar sobre técnicas de intrusión y vulnerabilidades web y extraer información para su posterior uso.
- Inventariado de las aplicaciones web, actualmente en desarrollo, y modelado de las más importantes amenazas, incluyendo la lista de componentes de las que se hace uso para conocer si una aplicación es vulnerable o no.
- Realización de una plataforma común a todos los proyectos para el control y monitorización de la seguridad en el desarrollo de aplicaciones web.

1.3 Planificación del proyecto

Para empezar la realización del proyecto en sí, hubo que documentarse sobre los ataques y vulnerabilidades más comunes en entornos web [2]. También se estudió las diferentes herramientas que se utilizan actualmente para la detección de vulnerabilidades [7].

Conforme se iba conociendo más sobre el funcionamiento tanto del proceso de desarrollo de cada proyecto por el departamento, que se detallará más adelante en el apartado 3. Ciclo de Desarrollo, como de las herramientas de análisis, se iban perfilando las necesidades y las soluciones que debían ser tomadas, entre las que destacaban *OWASP ZAP*, *Arachni* y *Wappalyzer* que se detallarán más adelante.

Se propuso, entonces, realizar una intranet que permitiera conectar a jefes de proyecto y analistas de seguridad para coordinar esfuerzos, permitiendo a estos últimos conocer el funcionamiento de cada aplicación y, a los jefes de proyecto, facilitarles la integración de análisis de seguridad automatizados [8], siendo esta la solución finalmente adoptada.

2. Estado del Arte

Es necesario primero introducir una serie de conceptos que permitan al lector tener una visión completa sobre toda la complejidad y problemática que conlleva el desarrollo de una web, así como las formas más comunes para explotar las posibles vulnerabilidades que puedan tener y las contramedidas que se están adoptando actualmente.

2.1 Modelado de amenazas

El modelado de amenazas (threat modeling) es un enfoque para el análisis de seguridad de una aplicación web que permite identificar, cuantificar y localizar los riesgos asociados a dicha aplicación, asegurando que está siendo desarrollada pensando en la seguridad desde el momento de su concepción, otorgando al analista de seguridad una perspectiva general de toda la aplicación, así como los puntos de entrada a la misma y los riesgos asociados a cada uno de ellos.

Hay varios modelos a seguir, como el modelo *STRIDE*, *DREAD* o *Trike* [9], habiéndonos centrado principalmente en la metodología seguida por *Open Web Application Security Project* [10], que es más sencilla de seguir, pudiendo después aplicar el modelo que deseemos.

El modelo *OWASP* divide todo el proceso en tres partes bien diferenciadas, de las que haremos uso durante el proceso de desarrollo de un proyecto:

- **Descomponer la aplicación:** Entender cómo la aplicación interacciona con agentes externos creando para ello casos de uso para conocer cómo un usuario (y atacantes) pueden utilizar la aplicación, identificar los recursos en los que pueda estar interesado un posible atacante e identificar permisos de acceso que la aplicación da a agentes externos. Esta información puede servir para generar diagramas de flujo de datos.
- **Determinar y clasificar amenazas:** Buscaremos amenazas para cada una de las interacciones de la aplicación y las clasificaremos basándonos en aspectos como el daño que podría causar a la empresa o clientes, la facilidad de ser explotada, la dificultad para mitigar la amenaza, etc.
- **Determinar contramedidas y formas de mitigación:** En este paso vamos a definir la estrategia a seguir en caso de encontrar una amenaza y la forma de reaccionar cuando una amenaza se hace realidad. Entre las opciones que podemos elegir están: eliminar el componente vulnerable, aceptar el riesgo, informar a los usuarios del riesgo o no hacer nada.

A la hora de utilizar una implementación, nos basaremos en el modelo SDL [11]propio de la empresa Microsoft por medio de una herramienta llamada *Microsoft Threat Modeling Tool* [12].

2.2 Metodologías de desarrollo ágil

El desarrollo ágil es un concepto que en los últimos años ha demostrado su utilidad y que va ganando cada vez más fuerza. Estas metodologías tienen en cuenta la evolución del proyecto a lo largo del tiempo, de forma que sus requisitos y soluciones también evolucionan con él.

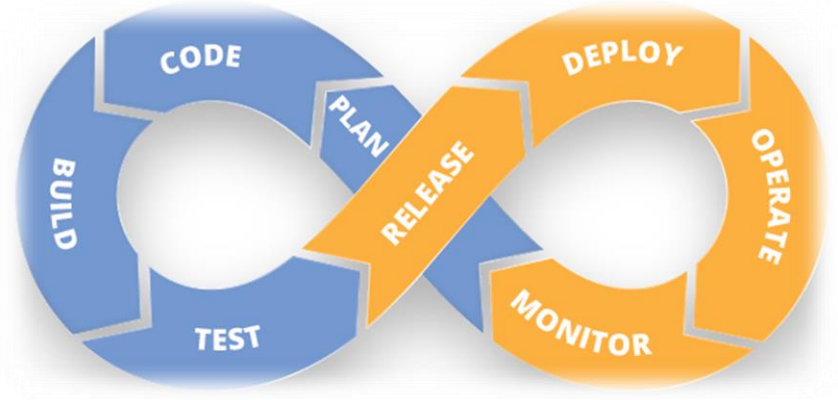


Figura 5 Ciclo DevOpsSec

En una metodología no ágil, como por ejemplo de ciclo en cascada, hay una serie de fases que se ejecutan secuencialmente como puede ser la planificación de requisitos, el diseño, la implementación, la verificación y el mantenimiento, siendo en esta última fase cuando ya se le enseña el proyecto al cliente. Estas fases siguen una secuencia en contraposición a las metodologías ágiles cuyas fases siguen una secuencia cíclica como la mostrada en la Figura 5, de forma que continuamente se cambian los requisitos, el diseño, la implementación etc. Al final de cada ciclo, también llamado iteración, se entregan funcionalidades completas de forma que el usuario es capaz de usarlo desde el principio sin que el proyecto entero esté completo y pudiendo adaptar cada ciclo al estado actual de los requisitos.

Para la implementación de metodologías ágiles y más concretamente *DevOps* [5], se hace hincapié en la automatización de tareas repetitivas, por medio de ciclos CI/CD [13], como la reducción de ficheros de código, la optimización de imágenes, la prueba de componentes de una aplicación por medio de *test unitarios* o el despliegue automatizado. Esta automatización se puede llevar al terreno de la seguridad informática, pues hay multitud de tareas que pueden ser automatizadas dejando tiempo al analista de seguridad para hacer análisis manuales, que son insustituibles.

2.3 Entornos de desarrollo

En todo el proceso de desarrollo, se hace uso de varios entornos donde realizar pruebas:

- **Producción**: En este entorno está la página web con la que interactúa el usuario final.
- **Preproducción**: Este entorno es el paso previo a producción, en él normalmente se configura todo, de forma que sea lo más parecido posible al entorno de producción. Se comprueba que en el servidor estén instalados todos los programas necesarios para que el servidor web funcione. En páginas web

basadas en gestores de contenidos (CMSs) estos entornos son usados como entornos de desarrollo, pues el contenido es creado sin líneas de código y a través del propio gestor con ayuda de *plugins* y módulos.

- **Desarrollo**: entorno donde se comprueba que el código compila y que pasa todos los test unitarios. Este entorno suele estar alojado en la propia máquina del desarrollador, pero puede ser replicado utilizando el propio código fuente de la página web a través de herramientas como *Jenkins* [14].

2.4 Implementación de la seguridad

Actualmente la mayoría de las empresas que incorporan seguridad en sus procesos de desarrollo hacen uso de metodologías ágiles e incorporan ciclos CI/CD añadiendo a éstos análisis de seguridad, siendo *Jenkins* una de las principales herramientas para tal fin.

Alguna de las posibilidades que nos brinda esta herramienta pasa por desplegar un entorno de pruebas para test unitarios (*unit testing*), hacer pruebas con la base de datos, desplegar la infraestructura en producción, etc. Para ello cuenta con multitud de *plugins* que permiten personalizar por completo el comportamiento ante una actualización de un proyecto, por ejemplo, para realizar un análisis de seguridad.

En los proyectos actuales de la empresa, rara vez se hace uso de estos ciclos CI/CD o de herramientas como Jenkins y en los que sí, se hace de forma parcial o por requerimiento de los clientes. Es por ello que no podremos utilizar directamente Jenkins para realizar los análisis de seguridad y deberemos buscar o crear una herramienta a la medida de nuestros proyectos.

Por otro lado, a la hora de realizar análisis/escaneos de seguridad se utilizan herramientas especializadas siendo una de las más extendidas el proyecto estrella de OWASP: *Zed Attack Proxy* también llamado ZAP [15] . Esta herramienta es una de las más potentes de código libre actualmente en el mercado, pero su automatización resulta complicada para evitar falsos positivos y el consumo de recursos resulta excesivo. Es por ello por lo que utilizaremos ZAP como sistema de escaneo manual y para los escáneres automáticos alternativas como *Arachni* [16].

Arachni es un escáner de seguridad escrito en el lenguaje de programación *ruby,* que muestra una tasa de falsos positivos muy baja evitando mostrar como amenazas algo que realmente no lo és. El soporte que recibe es constante por parte de su desarrollador, que resuelve diariamente los problemas que encuentran sus usuarios y tiene la capacidad de dividir la carga de trabajo de un análisis de seguridad utilizando escáneres en paralelo (modo *GRID*).

3. Ciclo de Desarrollo

En este apartado nos centraremos en explicar el proceso de desarrollo desde que un cliente contacta con la empresa para realizar un proyecto hasta que ese proyecto es llevado a cabo y se entrega al cliente, así como tratar de incluir en dicho proceso medidas para asegurar cierto nivel de calidad en cuanto a seguridad se refiere [17].

Cuando un cliente contacta con la empresa (Figura 6) para trasladarle una necesidad, lo hace a través del Gerente de Cuenta que tiene asignado. En ese momento, el Gerente de Cuenta solicita ayuda al departamento de Desarrollo de Negocio, que es el encargado de dar soporte técnico a la red comercial, asignando a un Técnico Preventa. El Técnico Preventa, junto con el Gerente de Cuenta, coordina con el cliente las reuniones o contactos necesarios para poder definir lo mejor posible los requisitos funcionales que describen la necesidad del cliente, así como los temas técnicos que resulten importantes para poder llevar a cabo el proyecto: integraciones con sistemas del cliente, tecnologías hardware o software a utilizar, etc. Tras este trabajo es cuando el departamento comercial y técnico realizan una oferta que se traslada al cliente.

3.1 Metodología actual

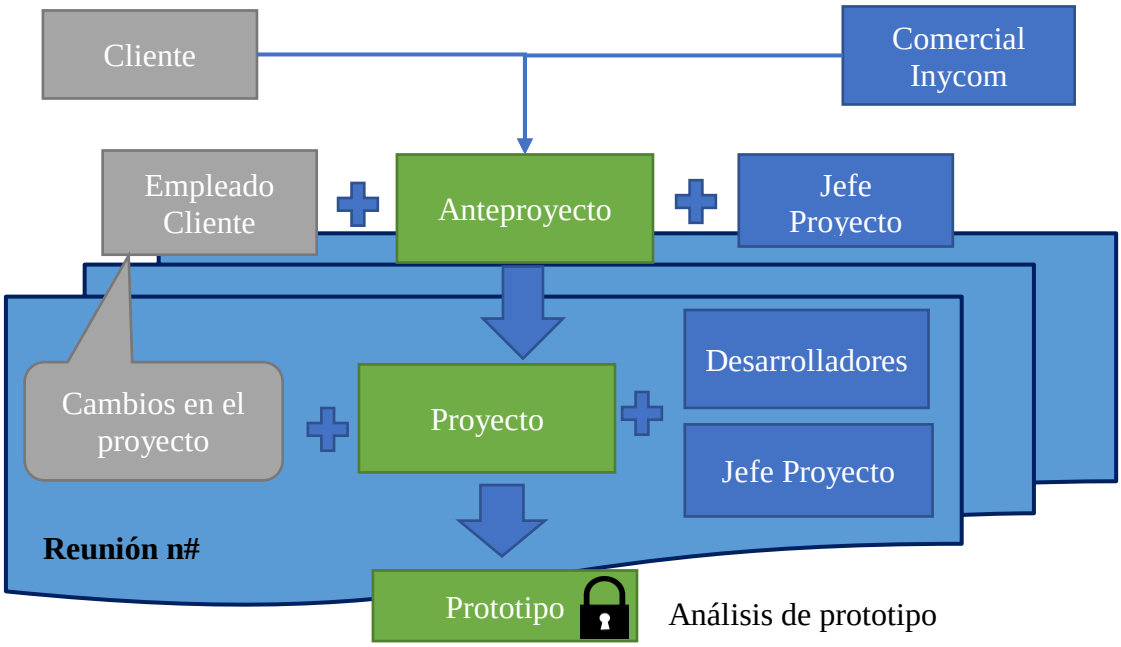


Figura 6 Proceso de desarrollo de un proyecto en Inycom

Una vez el cliente acepta la oferta, es cuando comienza el proyecto, asignándose un Jefe de Proyecto y un equipo de trabajo para la realización del mismo. El Jefe de Proyecto, junto con el equipo, realizan una reunión de arranque interna en la que se trasladan todos los datos funcionales y técnicos del proyecto, sus riesgos y las características que deban ser tenidas en cuenta, y se comienza a planificar el desarrollo y las tareas a realizar. En paralelo, se concierta una reunión de arranque del proyecto, o de KickOff, entre el Jefe de Proyecto y el cliente, a la que si es necesario pueden acudir el Gerente de Cuenta y el Técnico Preventa. En dicha reunión se confirman los datos de

la oferta y se definen los alcances, entregables, fechas de entrega y todo lo relevante del proyecto: accesos, tecnologías, interlocutores, forma de comunicación, puesta en común de los riesgos detectados, etc.Después, el jefe de proyecto realiza una reunión interna con los desarrolladores a su cargo, definiendo la infraestructura que utilizarán, las metodologías internas de desarrollo, un calendario de reuniones internas y con el cliente, etc.

Durante el desarrollo del proyecto, además, se llevan a cabo reuniones de seguimiento, tanto internas como con el cliente, en las que se muestran los avances del mismo y se sacan a la luz los problemas o incidencias que puedan haber ocurrido, para poder mitigar lo antes posible cualquier desviación que pudiera producirse. Dicho seguimiento y comunicación es uno de los aspectos más importantes para que un proyecto sea un éxito para todas las partes, que es el objetivo principal.

3.2 Metodología propuesta

Cuando se busca información sobre cómo implementar metodologías de seguridad en los proyectos de desarrollo web en una empresa, se suele obtener información relacionada con metodologías *DevOpsSec* o el uso de herramientas de automatización como *Jenkins.* En la mayoría de estos casos, los proyectos desarrollados son de larga duración y de carácter interno, de forma que no son proyectos para clientes.

Los proyectos en la empresa tienen, generalmente, una duración comprendida desde los 2 meses hasta los 6 y dependen de un cliente, siendo este el que marca el ritmo del desarrollo.

Además de todo lo anterior, hacer modificaciones en la forma de trabajo cuando sabemos que actualmente está funcionando correctamente es arriesgado. Es por ello por lo que la introducción de estas metodologías de seguridad se hará de forma gradual y localizada dentro de la cadena.

Como primera aproximación, se creará un rol dentro de cada proyecto que hará las funciones de analista de seguridad. Su función será seguir la evolución de los proyectos a su cargo identificando amenazas y trabajando con el jefe de proyecto para su subsanación.

El analista de seguridad acompañará al equipo en cada reunión, tanto con el cliente, como en las internas, obteniendo así una visión sobre el estado y la evolución del proyecto sin que sea necesario ser notificado de cada cambio en el mismo.

En estas reuniones la presencia del analista deberá pasar desapercibida, siendo casi todo el tiempo un oyente y anotando los riesgos de cada nuevo cambio introducido. Así evitamos entorpecer el proceso de desarrollo cuestionando cada cambio que trae consigo un riesgo.

Tras la reunión, el analista plasmará en un modelo de amenazas los resultados, comunicándole, en caso necesario, los riesgos no mitigados al jefe de proyecto. El resto del tiempo el analista centrará sus esfuerzos en realizar análisis periódicos del proyecto, utilizando para ello escáneres de seguridad, revisando componentes que presentan un alto riesgo de amenaza o trabajando en el modelo de riesgos, quedando el proceso final como se ve en la Figura 7.

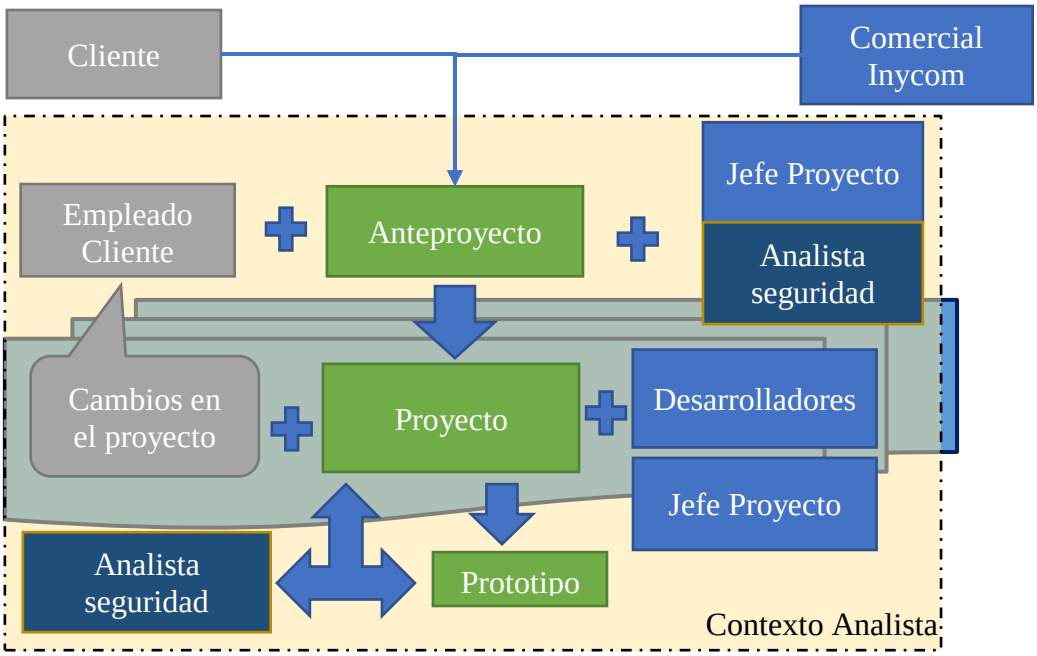


Figura 7 Proceso de desarrollo de un proyecto en Inycom con seguridad

De esta forma hemos logrado añadir seguridad, no sólo desde el principio de la cadena, sino también de forma continua tanto en cada cambio del proyecto, como en la fase final en la que se entrega un prototipo.

En caso de que un proyecto, que haya finalizado y esté funcionando, sufriera algún ataque, el analista deberá proporcionar estos modelos de riesgos e informes sobre vulnerabilidades al equipo de respuesta encargado de dicho ataque y coordinarse con ellos posibilitando a cualquier equipo no relacionado con el desarrollo del proyecto, conocer el funcionamiento, la infraestructura, así como los puntos donde el ataque pueda estar centrándose en pocos minutos.

Ni que decir tiene, que toda esta información proporcionada por el analista de seguridad deberá ser guardada correctamente, porque, igual que puede ayudar a un equipo de respuesta a defenderse, su exfiltración puede desencadenar un ataque más preciso por parte de un agresor.

3.3 Microsoft Threat Modeling Tool

Para la creación de los modelos de amenaza se ha propuesto la utilización de la herramienta Microsoft Threat Modeling Tool [12]. Esta herramienta creada por Microsoft permite identificar y mitigar posibles problemas de seguridad dentro del ciclo de vida de desarrollo, estando pensada para expertos no relacionados con la seguridad, facilitando que cualquier desarrollador pueda crear sus propios modelos de amenazas

así como la generación de informes con una lista de problemas de seguridad inherente al modelo y sugerencias para su posible mitigación. Está basado en el enfoque de SDL de Microsoft y puede variar con respecto a OWASP [10].

Microsoft Threat Modeling Tool, a partir de ahora MTMT, es ampliamente utilizada en proyectos de la propia Microsoft, lo que nos da un cierto nivel de confianza, en cuanto a su utilidad real, el mantenimiento futuro que recibirá, así como la documentación disponible para su uso.

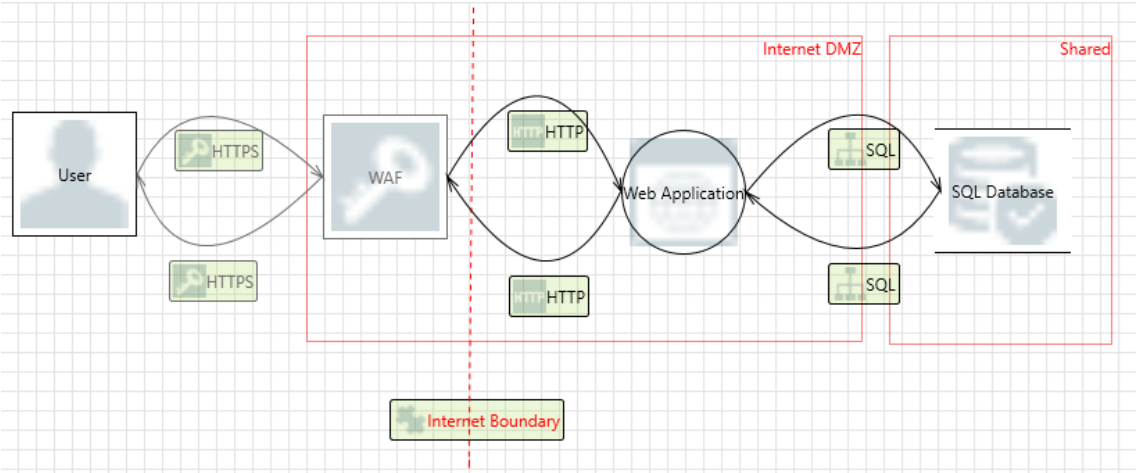


Figura 8 Ejemplo Microsoft Threat Modeling Tool

Como se puede apreciar en la Figura 8, el modelo de amenazas se crea dibujando sobre un lienzo, tanto la infraestructura de nuestra aplicación, como los flujos de datos, facilitando el entendimiento entre desarolladores y analístas de seguridad, reduciendo la brecha de conocimiento entre ambos mundos.

Como resultado MTMT genera unos informes de amenazas que permiten comprender rápidamente los riesgos asociados, así como los pasos que se deben dar para mitigarlos. En la Figura 9 hay un ejemplo de un riesgo generado e incluido en el informe de seguridad realizado por MTMT basándonos en una plantilla preexistente [18].

3. Spoofing the WA External Entity [State: Not Started] [Priority: High]

Category:	Spoofing
Description:	WAF may be spoofed by an attacker and this may lead to unauthorized access to Web Application. Consider using a standard authentication mechanism to identify the external entity.
Justification:	<no mitigation provided>
Countermeasure:	
Risk:	High
Team:	Team1

Figura 9 Riesgo creado por MTMT

Para la generación de los modelos se utilizan plantillas compuestas de diferentes pinceles llamados *stencils* y tipos de amenazas. Los diseños incluyen tanto

aplicaciones y usuarios como el flujo de datos intercambiados a través de la red en la que operan.

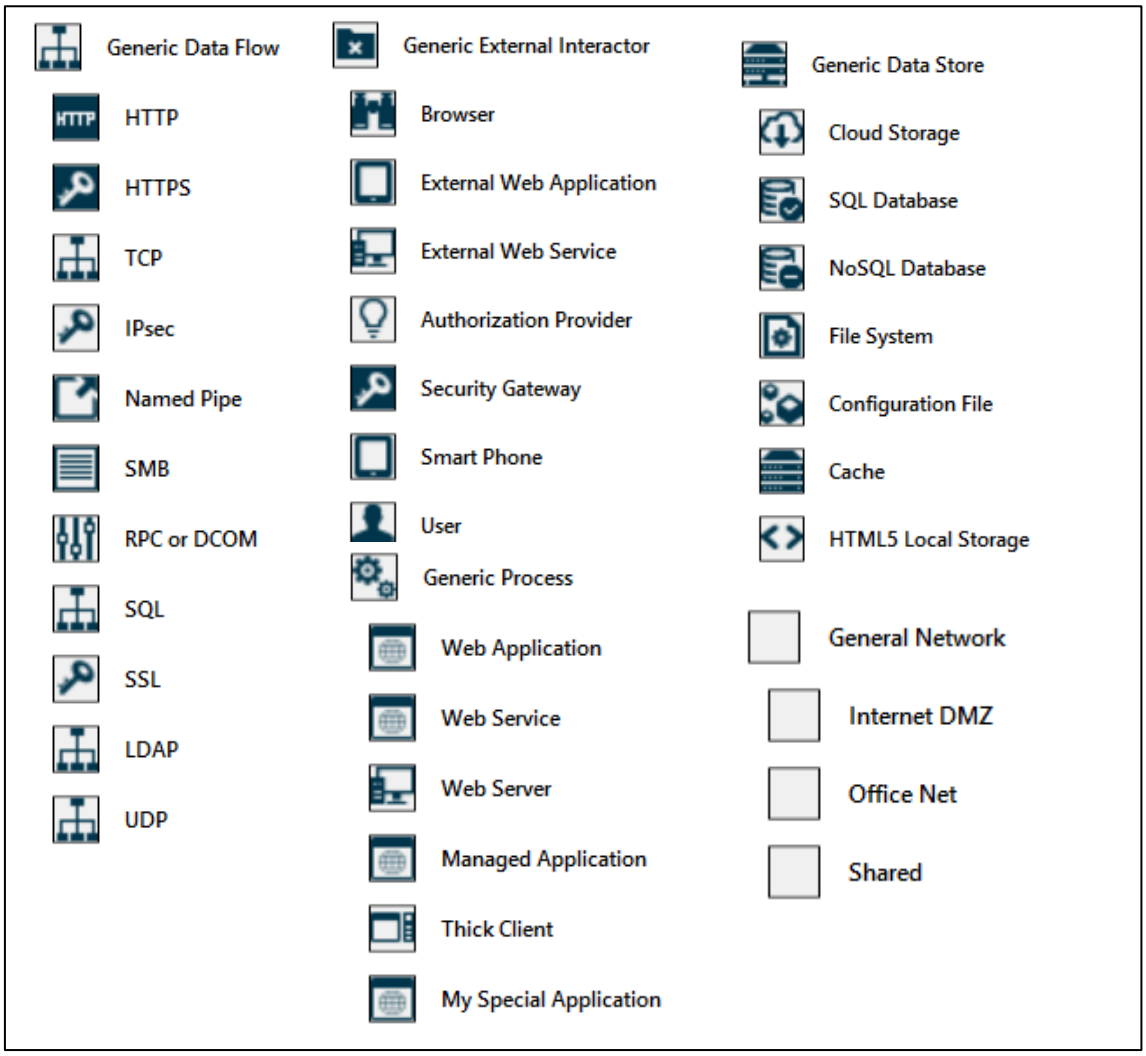


Figura 10 Lista de elementos con los que confeccionar un modelo de amenazas

Una lista de *stencils* de una plantilla se puede consultar en la Figura 10.

El registro de nuevas amenazas se hará asociándolas con los diferentes elementos que hemos creado cuando se dan ciertas relaciones entre ellos. Así podemos tener que, a un proceso que interactúa con una base de datos, se le asocia un riesgo de inyección de código SQL.

Durante el desarrollo de un proyecto, se irán encontrando nuevos riesgos, mejorando no solo el modelo de riesgos de dicho proyecto, sino la plantilla que utilizaremos con el resto contribuyendo cada uno a la mejora de todos.

En el repositorio de código de Microsoft [18], podemos encontrar varios ejemplos sobre plantillas y modelos en los que basarnos para la confección de los nuestros propios para la empresa, pudiendo modificar los ya existentes (Figura 11).

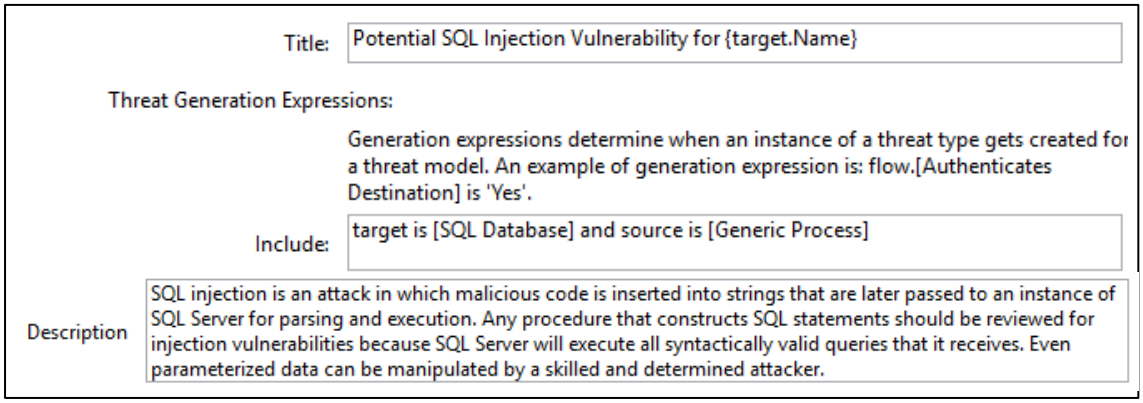


Figura 11 Edición de amenazas (MTMT)

4. Plataforma de seguridad continua

Como resultado del TFG, se ha desarrollado una plataforma de seguridad que nos permite llevar a cabo, tanto el inventariado de las aplicaciones, como la realización de análisis de seguridad, centrándonos sobre todo en este apartado al desarrollar dicha plataforma.

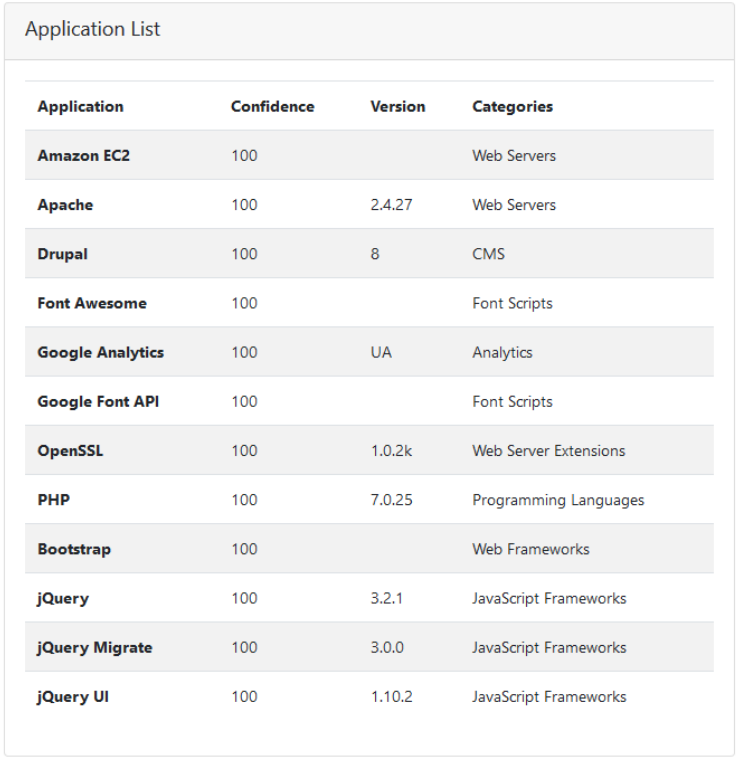
A traves del editor de pipelines que incorpora la plataforma, el analista de seguridad y el jefe de proyecto, pueden iniciar escáneres de seguridad o programarlos a ciertas horas de la semana. Además, ambos pueden subir modelos de amenaza, creados previamente con MTMT, y trabajar conjuntamente en su mejora sin pisarse el trabajo mutuamente.

4.1 Requisitos

Se plantearán a continuación una serie de necesidades que se deben satisfacer.

4.1.1 Control de Dependencias

En un primer momento se plantea la posibilidad de hacer un seguimiento de las dependencias que utiliza la aplicación (Figura 12). Esto parece algo muy sencillo, pero que en la práctica se vuelve difícil de mantener debido a que estas se actualizan continuamente. Mantener un inventariado de las mismas se vuelve indispensable para conocer si alguna vulnerabilidad nos afecta o si dicha actualización cambia drásticamente el funcionamiento de un componente y el sistema se vuelve inestable perdiendo tiempo en buscar el fallo.



Application List

Application	Confidence	Version	Categories
Amazon EC2	100		Web Servers
Apache	100	2.4.27	Web Servers
Drupal	100	8	CMS
Font Awesome	100		Font Scripts
Google Analytics	100	UA	Analytics
Google Font API	100		Font Scripts
OpenSSL	100	1.0.2k	Web Server Extensions
PHP	100	7.0.25	Programming Languages
Bootstrap	100		Web Frameworks
jQuery	100	3.2.1	JavaScript Frameworks
jQuery Migrate	100	3.0.0	JavaScript Frameworks
jQuery UI	100	1.10.2	JavaScript Frameworks

Figura 12 Control de dependencias de un proyecto

Estas dependencias pueden ser, tanto el lenguaje de programación usado, los frameworks que se están utilizando, así como las librerías empleadas en estos lenguajes de programación identificando correctamente las versiones utilizadas.

Habrá, pues, que crear unas normas y metodologías aplicables a las actualizaciones e inserciones de nuevas dependencias, puesto que no solo de ello depende la seguridad del sistema sino también la estabilidad del mismo.

4.1.2 Gestión de Modelos de Riesgo

Desde el primer momento se ha tratado de integrar la seguridad en el desarrollo y nada mejor que tener en cuenta los requisitos de seguridad desde el arranque del proyecto. Estos modelos evolucionan junto con el proyecto, tendiendo ambos a cambiar mucho a lo largo del tiempo. Esto significa, en muchos casos, el tener multitud de archivos correspondiendo a versiones diferentes (y por tanto, con nombres de archivo similares) en una misma carpeta, dificultando encontrar la última versión.

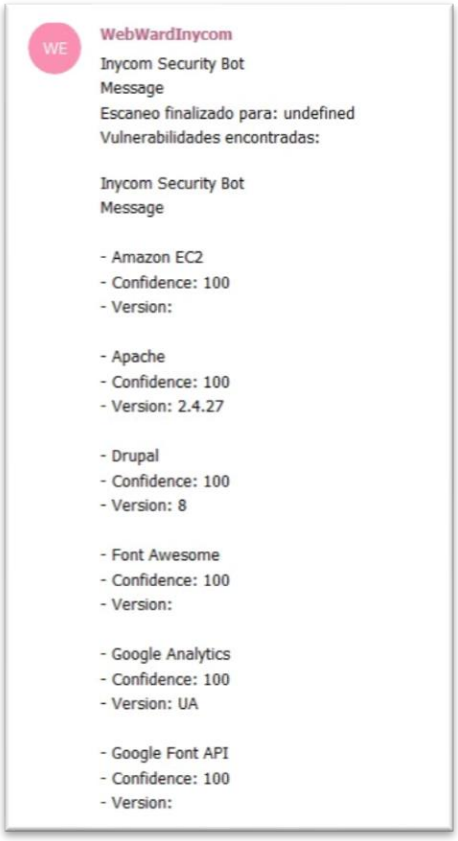
Además, el modelo de amenazas debe ser completado tanto por los desarrolladores y jefes de proyecto como por el analísta de seguridad, debiendo este último poder acceder a todos los modelos de riesgo de los proyectos bajo su supervisión.

4.1.3 Análisis de vulnerabilidades

Un término que vienen a la mente cuando se piensa en seguridad es el análisis de vulnerabilidades. Hoy en día hay una gran cantidad de herramientas, tanto de pago como gratuitas, disponibles para dicho fin. El uso eficaz de cada una de estas herramientas toma tiempo y los problemas de seguridad descubiertos por ellas están limitados a lo que les ha sido programado por sus desarrolladores.

No se pretende en ningún momento realizar un escáner de vulnerabilidades. Se pretende facilitar su uso al personal que no posee el perfil de seguridad informática que de otra forma sería necesario para el manejo de estas herramientas, pues existen ya multitud de analizadores, escáneres y otras herramientas para dicha labor y que además cuentan con el apoyo de comunidades de desarrolladores detrás, mejorándolas cada día.

4.1.4 Informes personalizados



En la primera reunión que se realizó, se planteó la posibilidad de enviar los resultados de análisis de seguridad, utilizando otros métodos distintos al correo electrónico para evitar la acumulación de correos y asegurarnos de que el destinatario no pase por alto el informe.

La comunicación entre los diferentes integrantes de un proyecto pasa por el uso de Skype que recientemente ha integrado en la aplicación los bots de mensajería, una opción muy interesante para la integración de notificaciones directamente en las conversaciones de los equipos, como se aprecia en la Figura 13.

Al final, se ha utilizado Telegram, en lugar de Skype, debido a su facilidad a la hora de programar y hacer pruebas [19], pero los mismos conceptos pueden ser aplicados a un bot de Skype u otros servicios de mensajeria en un futuro.

Figura 13 Notificación en Telegram

4.1.5 Requisitos inherentes a la seguridad

Todo el proyecto está en el marco de la seguridad y como tal, la plataforma no puede ser menos y estar pensada desde el primer momento con la seguridad en mente. Entre los requisitos necesarios se destacan:

- **Cabeceras HTTP de seguridad:** se hará uso de cabeceras que implementan la gran mayoría de navegadores actuales (se descarta por completo la retrocompatibilidad con versiones antiguas de Internet Explorer) tales como CORS, CSP, Frame-Options, Content-Type-Options nosniff, XSS-Protection o la reciente Expect-CT. Todas ellas dan un añadido de seguridad sin coste alguno en cuanto a tiempo de implementación. Además, estas permiten indicar rutas para notificarnos cuando se violen las reglas definidas por ellas.
- **Sentencias SQL preparadas:** Todas las llamadas a la base de datos se harán por medio de sentencias preparadas que evitan el SQL Injection. También se hará así para búsquedas que utilizen parámetros obtenidos de otra consulta para evitar que un SQL Injection almacenado en la BB.DD pueda comprometernos.
- **Sanear entradas:** Todas las entradas serán saneadas para evitar almacenar información vulnerada utilizando XSS.
- **Sanear salidas:** La interfaz de usuario se encargará de sanear toda entrada que le llegue desde el servidor (salida del servidor), de forma que se evite el XSS.

- **Permisos de usuario:** La jerarquía de usuarios debe ser robusta de forma que un usuario no acceda a información sin autorización.
- La aplicación debe ser estable, tolerante a fallos y escalable.

4.2 Análisis y diseño

En este apartado esbozaremos la infraestructura informática de la que se hace uso.

4.2.1 Entorno de ejecución

Uno de los requisitos que se han nombrado es la facilidad de uso. Siguiendo esta línea se puede facilitar también la instalación del sistema. Hoy en día, cualquier página web hace uso de una base de datos. A la hora de instalarla es necesario instalar por un lado la aplicación web, luego la base de datos y configurar la primera para usar la segunda.

Aunque este proceso no es más complejo que introducir un par de comandos, su complejidad puede marcar la diferencia entre usarla y no. Además de la base de datos, se requerirá lanzar un escáner de vulnerabilidades web. Por ello daremos la posibilidad a los usuarios de ejecutarlo en entornos virtualizados por medio de contenedores LXC de la empresa *Docker* [20], con la opción de utilizar un método de orquestación como *Kubernetes* [21], el cual permite definir una serie de características como el número de contenedores en ejecución, los sistemas de almacenamiento de información o la forma de acceder a los contenedores por medio de conexiones TCP.

Simplificando lo anterior, la aplicación puede correr tanto en un servidor o máquina virtual, instalado de forma manual, por medio de contenedores, que ya contienen todo lo necesario, o simplificar el proceso aún más, utilizando *Kubernetes*, aprovisionándose así mismo de la base de datos y de los escáneres de seguridad, teniendo el usuario únicamente que ejecutar un simple script.

4.2.2 Proceso de arranque

El proceso de arranque de la aplicación será el siguiente (resumen en Diagrama 1):

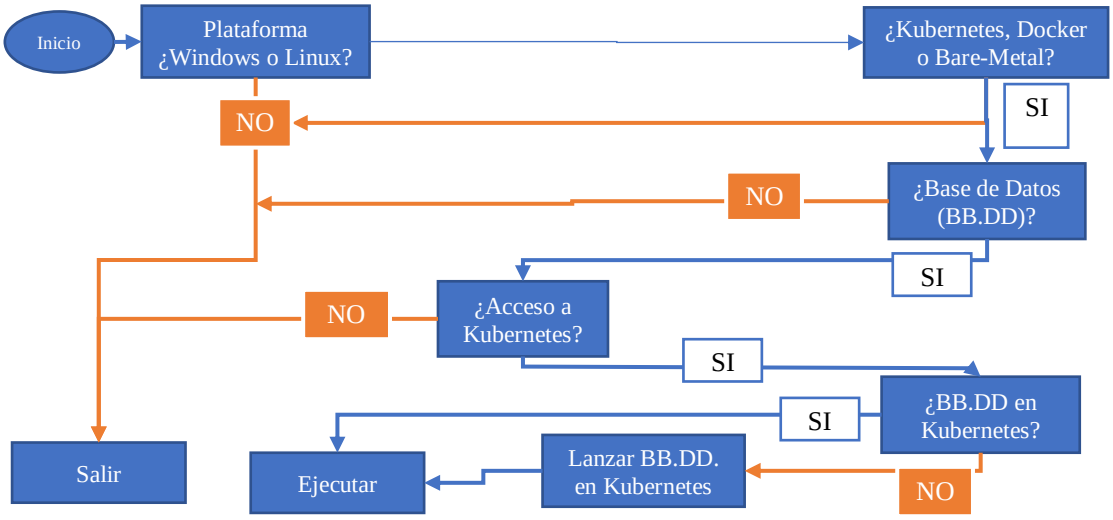


Diagrama 1 Proceso de inicio de la plataforma

1. Se comprueba el sistema en el que estamos corriendo:
    a. Plataforma utilizada: Windows o Linux
    b. Comprobar entorno de ejecución: Kubernetes, Docker o local (Bare-Metal)
        i. Si estamos en local, comprobamos si tenemos Kubernetes en el sistema o Docker.
2. Comprobamos acceso a la BB.DD
    a. Si no hay una base de datos disponible, pero tenemos acceso a Docker o Kubernetes:
        i. Buscamos la BB.DD.
        ii. Si no existe la BB.DD la creamos (descargar y ejecutar contenedor)
    b. Si aun así no hay base de datos finalizamos la ejecución.
    c. Comprobamos el modo de ejecución: Modo API REST, Pipeline o mixto
3. Iniciamos el servidor

4.2.3 Base de Datos

Respecto a la base de datos se ha decidido utilizar PostgreSQL. Para empezar, se prefería utilizar una base de datos relacional de tipo SQL puesto que, para datos estructurados, resulta más rápido realizar búsquedas que en las no-relacionales.

De entre las alternativas de bases de datos SQL se acabó imponiendo PostgreSQL frente a MariaDB, MySQL, SQLServer o Oracle. Los motivos principales fueron: su buena documentación, la posibilidad de indexar datos en formato JSON directamente en la base de datos y no requerir del pago de licencias para su uso al ser de código libre. Además de lo anteriormente mencionado, PostgreSQL es la base de datos de la que hace uso el escáner de vulnerabilidades Arachni si funciona en modo GRID, que, aunque no se utilice, siempre cabe la posibilidad de permitirlo en un futuro sin instalar una segunda base de datos.

La estuctura de los datos almacenados se puede consultar en el *ANEXO F: Tablas en la base de datos*.

4.2.4 Servidor

Como infraestructura de servidor (*Backend*) se ha elegido trabajar con *NodeJS* [22], un entorno de programación muy popular, con extensa comunidad, cientos de miles de librerías disponibles y basado en el lenguaje de programación *Javascript*. Es el entorno preferido por una gran cantidad de empresas para crear sus servidores *REST* (Transferencia de Estado Representacional) debido a la programación asíncrona que no hace uso de hilos en contraposición a otros lenguajes como *Java*.

Estos servidores REST reciben comandos *HTTP* para buscar, eliminar, crear o modificar contenido en sus sistemas permitiendo que cualquier página web interactúe con nuestra plataforma. De esta forma, podremos crear un servidor REST y una Interfaz de Usuario web utilizando únicamente el lenguaje de programación *Javascript*.

Así mismo, integraremos en nuestro servidor la capacidad de acceso a entornos de virtualización, no solo para provisionarse de recursos como la base de datos, sino para en un futuro poder lanzar réplicas de páginas web en producción para facilitar la realización de pruebas.

4.2.5 Interfaz de usuario

Respecto a la interfaz de usuario, a la que a partir de ahora nos referiremos como *frontend*, se optó por la utilización de una interfaz web. Elegimos *Angular* [23] como entorno de trabajo (*framework*) para su diseño debido principalmente a la gran comunidad, la documentación disponible y la simplificación que aporta trabajar de forma independiente en cada componente de la página web lo que iba a reducir el tiempo necesario para su desarrollo.

Angular es un *framework* para crear aplicaciones web de forma sencilla dividiendo toda la aplicación en componentes más pequeños y reutilizables. Cada componente se diseña y programa de forma independiente al resto permitiendo la comunicación entre ellos por medio de servicios.

Toda la interfaz se ha creado desde cero utilizando como plantilla visual *Bootstrap,* como sistema MVC *Angular* y para los pipelines y los nodos se ha utilizado *SVG* en conjunto con *Angular*.

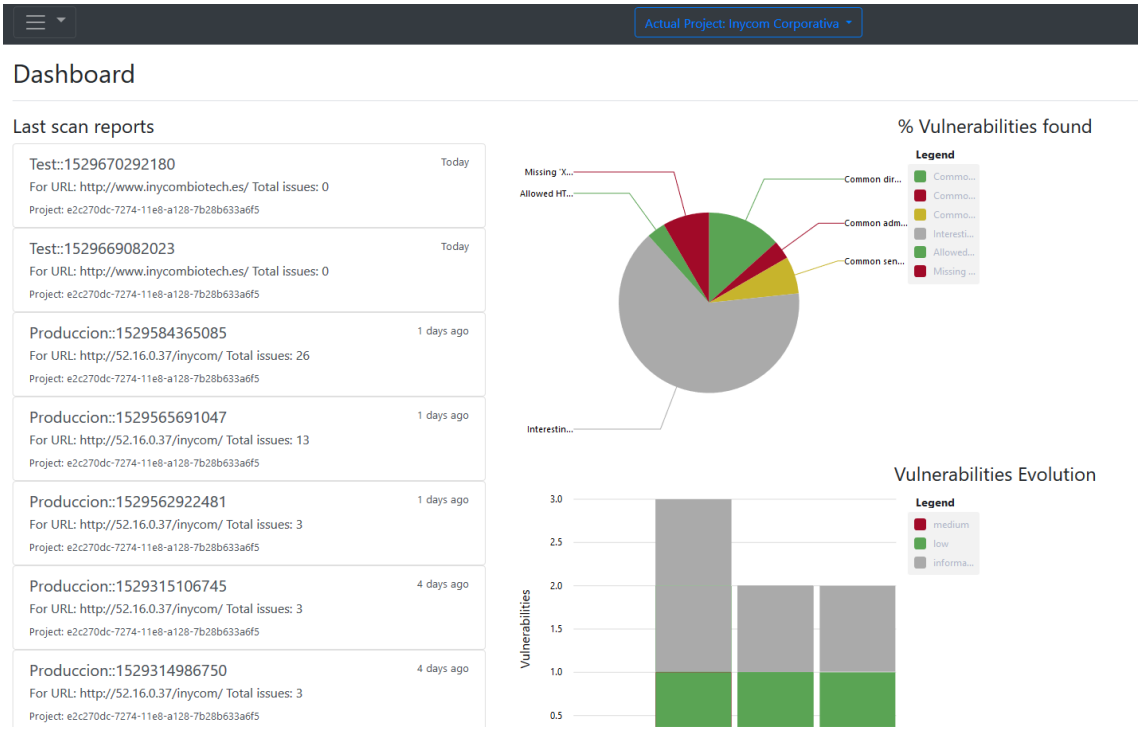


Figura 14 Aspecto visual final

El aspecto final de la interfaz se puede ver en la Figura 14, el resto de componentes visuales se pueden consultar en el *ANEXO E: Aspecto visual de la plataforma*.

4.3 Implementación

Para poder hacer más sencilla la programación de un análisis de vulnerabilidades, se ha creado una interfaz gráfica para crear "pipelines". Un pipeline define los pasos que se van a seguir cuando nuestra aplicación reciba un evento, como por ejemplo una actualización en el código fuente de una página web o una activación de forma programada a cierta hora. Estos pasos incluyen la realización de análisis de seguridad con perfiles personalizados, notificar a un usuario o replicar una web, utilizando su código fuente, para realizar análisis de seguridad de forma más rápida sin tener que hacer peticiones a una web remota.

En este apartado se hará una explicación superficial de todo el proceso, teniendo que recurrir al ANEXO B: Security Pipeline para su mejor comprensión.

4.3.1 CI/CD Pipeline

En los ciclos CI/CD todo el proceso se conoce como Pipeline.

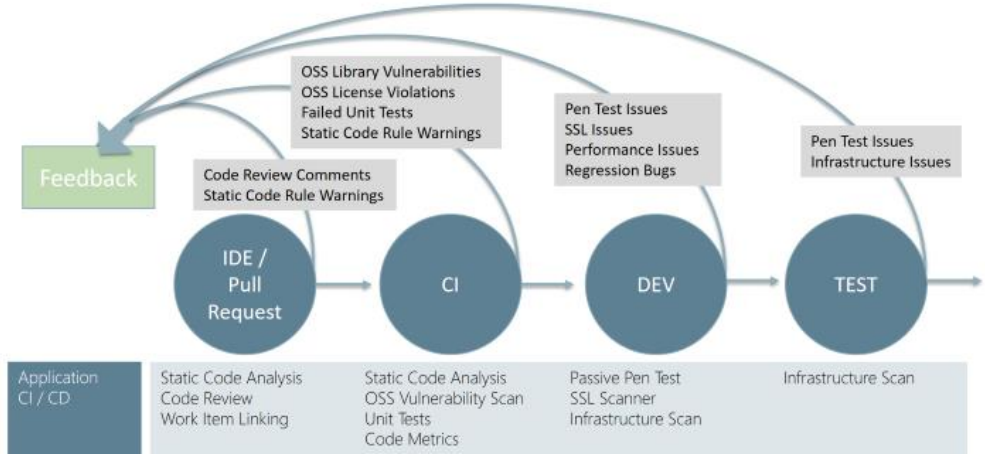


Figura 15 CI/CD Pipeline propuesto por Microsoft

En aplicaciones como Jenkins o GitLab un Pipeline empieza, por ejemplo, con un evento de tipo "git push" [24]. Continúa con la instalación en un entorno virtual de las dependencias que necesita el código que se ha actualizado. Acabado esto, se hace una batería de pruebas conocido como testeo unitario (unit testing) y si alguno de ellos falla, todo el Pipeline finaliza en error y se notifica a los desarrolladores de que algo ha fallado. Una simplificación orientada a seguridad y propuesta por Microsoft [13] se puede ver en la Figura 15.

4.3.2 Pipeline Visual Editor

Nosotros nos llevaremos este concepto de Pipeline al terreno de la seguridad web diseñando una herramienta de creación visual de Pipelines. Al ser visual, el desarrollador no tendrá que programar nada (menor curva de aprendizaje) y podrá identificar fácilmente los pasos que se están ejecutando, actualmente como por ejemplo un análisis de la web de Inycom o si por el contrario ha acabado y lo está notificando.

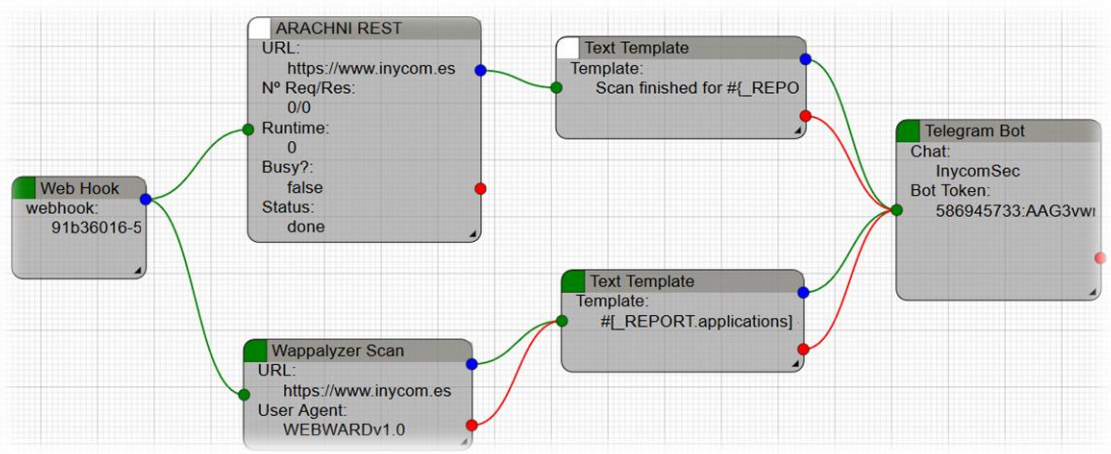


Figura 16 Editor de pipelines de la plataforma

En la Figura 16 hay una muestra de un Pipeline de Seguridad con sus nodos, donde podemos ver que la apariencia es similar a la utilizada por MTMT para que el usuario se sienta familiarizado desde el primer momento con la herramienta.

En este caso la ejecución comienza cuando un agente externo hace una llamada a un *webhook* (petición a un servidor web), este a su vez finaliza y activa los nodos "ARACHNI REST" y "Wappalyzer Scan", utilizados para realizar análisis de seguridad. La ejecución continúa de izquierda a derecha hasta que, finalmente, el último nodo notifica al usuario utilizando la aplicación de mensajería *Telegram*.

El código de colores facilita aún más si cabe seguir de forma ordenada el proceso de ejecución: rojo para errores, verde entrada/salida y azul para salida. Haciendo una analogía podríamos decir que cada nodo es un proceso *UNIX* que tiene una entrada estándar y dos salidas: una de error y otra la salida estándar.

Estos pipelines, aunque son diseñados utilizando la interfaz gráfica, deberán ejecutarse en el servidor. Sin entrar en muchos detalles, existen unos controladores de pipeline que se encargan de, periódicamente, consultar los que están disponibles y ejecutarlos. Para ello cada nodo del pipeline tiene asociado una etiqueta (TAG) que permite identificar qué módulo lo generó.

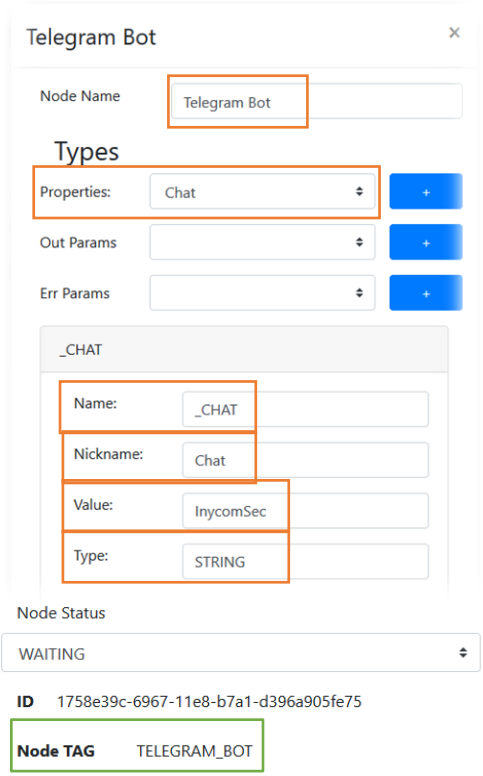


Figura 17 Editor de Nodos de la plataforma

Los módulos simplemente definen la plantilla que permite visualizar el nodo, así como de dónde se obtiene la funcionalidad (script), las propiedades por defecto y los requisitos en cuanto a librerías se refiere.

La configuración de un nodo se realiza modificando las propiedades del nodo. Estas propiedades son como variables de un lenguaje de programación permitiendo a los scripts que contienen la lógica almacenar información.

En la Figura 17, tenemos la interfaz de edición de un nodo llamado “Telegram Bot”, el cual posee una propiedad llamada “_CHAT” cuyo alias es “Chat”, la información que almacena es una cadena de texto (STRING) con valor “InycomSec”.

Cuando un nodo se ejecuta lo hace dentro de una máquina virtual no siendo posible acceder a otros recursos más allá de los que le son proporcionados por el controlador. Al finalizar su ejecución, un nodo pasa una serie de parámetros a los siguientes nodos conectados a él dependiendo de si finaliza con éxito o por el contrario ha finalizado en error. Estos parámetros siguen la misma estructura que las propiedades teniendo un nombre, valor y tipo.

Como hemos visto, un nodo se crea utilizando una plantilla aportada por un módulo, dejándole al usuario la libertad de crear los suyos propios. Sin embargo, queda la cuestión de los tipos de parámetros personalizados para editar parámetros especiales o definidos por los usuarios.

Los dos siguientes parámetros que se pueden ver en la Figura 18 son tipos personalizados, uno es de tipo *WEBHOOK* y el otro de tipo *HOURS* siendo la interfaz de edición de ambos completamente diferente. Esta interfaz de edición puede ser creada por medio de componentes de *Angular,* debiendo actualizar el código de la página web para que los cambios sean efectivos.

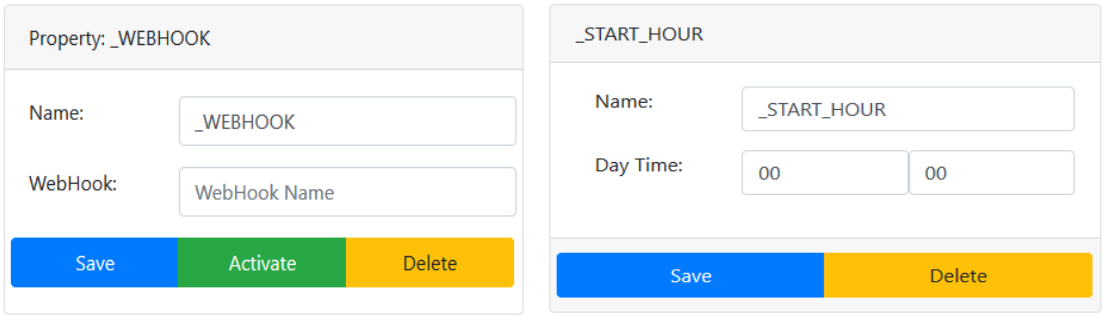


Figura 18 Editor de propiedades y parámetros

Todo lo explicado anteriormente permite que un usuario pueda configurar de forma rápida y sencilla desde una simple notificación hasta programar escaneos de seguridad a ciertas horas y en ciertos días sin tener que escribir una sola línea de código.

4.3.3 Escáneres de Seguridad

Para la realización de análisis de seguridad automatizados utilizaremos el escáner *Arachni,* así como uno más simple llamado *Wappalyzer*.

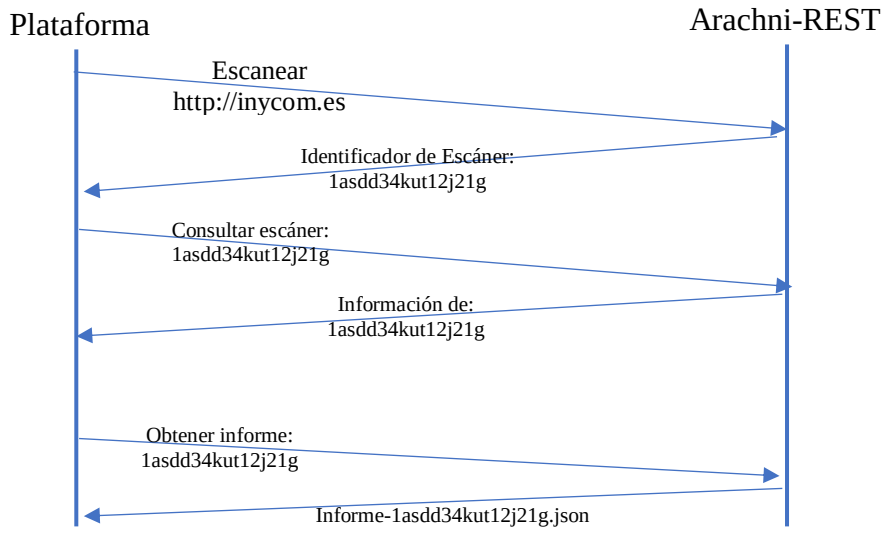


Diagrama 2: Proceso de análisis

Para controlar *Arachni*, se pueden mandar comandos hacia su interfaz *REST* [25], así como hacia una interfaz *RPC* (Remote Procedural Call o llamadas a procedimientos remotos). Para la primera versión del proyecto se ha marcado como objetivo utilizar únicamente la interfaz *REST* dejando *RPC* para usos futuros. Una simplificación del proceso completo en Diagrama 2.

La principal limitación que impone el uso de la interfaz *REST* es la imposibilidad de utilizar el modo GRID no pudiendo distribuir una misma instancia entre múltiples escáneres, aunque esto es más que suficiente para las necesidades actuales.

5 Pruebas

En un principio no conocemos las vulnerabilidades que tiene una página web. Lo que sí podemos llegar a saber es si las vulnerabilidades mostradas por los escáneres están realmente presentes o son falsos positivos. Para ello solo hace falta fijarse en si las rutas afectadas por una supuesta vulnerabilidad apuntan realmente a una dirección válida de la web. Es obvio que una vulnerabilidad en la ruta "http://www.inycom.es/localstart.asp" no existe, puesto que no se hace uso de ASP.

Para la realización de las pruebas se ha desplegado la plataforma haciendo uso de un servidor privado utilizando una instancia EC2 de Amazon Web Services de tipo *t2.medium,* como se aprecia en la Figura 19*,* al comprobar que un servidor más pequeño como el *t2.small* no tenía la potencia suficiente.

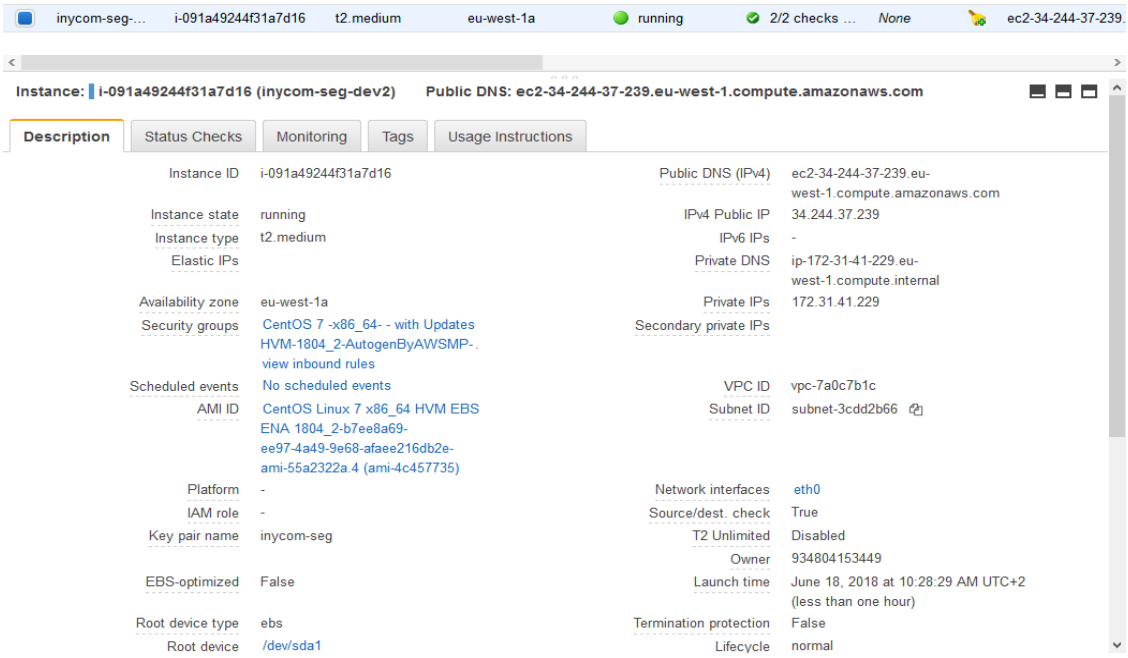


Figura 19 Interfaz de control de AWS

Para la comprobación del correcto funcionamiento de la herramienta, se ha analizado una web con vulnerabilidades conocidas como es *WebGoat* [26]. Para ello, se configuró la plataforma para iniciar sesión de forma automática con el plugin "autologin" de Arachni.

```
"plugins": {
  "autologin": {
    "url": "http://192.168.99.100:31614/WebGoat/login.mvc"
    "parameters": "username=arachni&password=arachni",
    "check": "What is WebGoat?"
  }
},
```

Si no aparece Login form could not be found en los logs de arachni, entonces está correctamente configurado.

Algunos ejemplos de vulnerabilidades descubiertas:

- Unencrypted password form (http://192.168.99.100:31614/WebGoat/login.mvc)
- Private IP address disclosure (http://192.168.99.100:31614/WebGoat/start.mvc)
- Password field with auto-complete (...100:31614/WebGoat/login.mvc)

Una comprobación rápida nos indica que la herramienta se comporta debidamente respecto a las vulnerabilidades existentes en *WebGoat* y a lo descrito en el ranking *SecTool* de escáneres de aplicaciones web [27].

Para las primeras pruebas en producción, se utiliza únicamente el escáner de dependencias *Wappalyzer* por ser más ligero y rápido, configurando el pipeline como en la Figura 20.

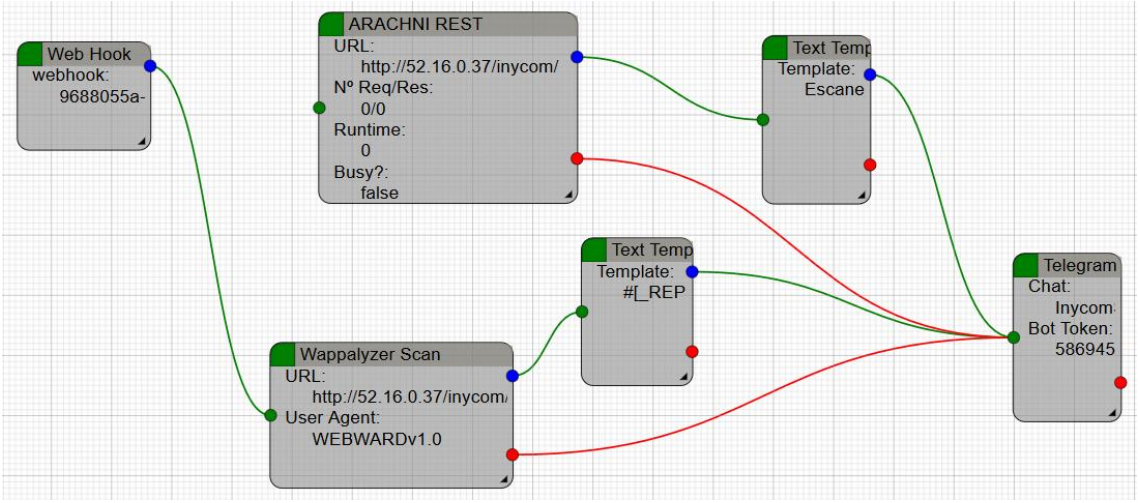
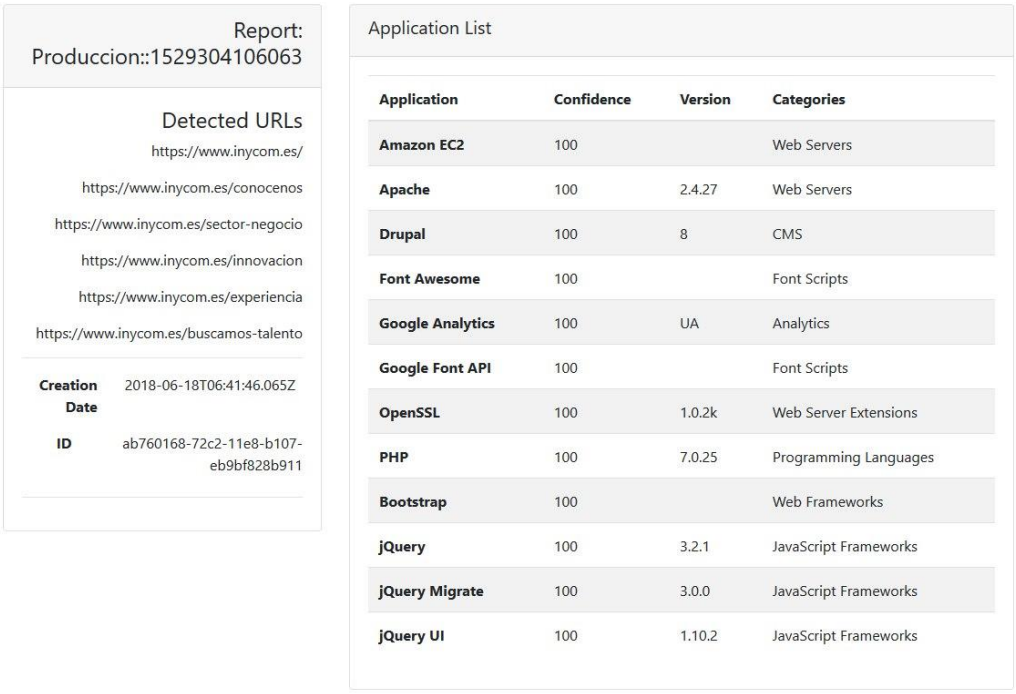


Figura 20 Pipeline configurado con Wappalyzer

Esta es una configuración básica en la que al activarse el *webhook* lanza únicamente un análisis a través del escáner *Wappalyzer* al acabar le pasa el informe a



Report:
Produccion::1529304106063

Detected URLs

https://www.inycom.es/

https://www.inycom.es/conocenos

https://www.inycom.es/sector-negocio

https://www.inycom.es/innovacion

https://www.inycom.es/experiencia

https://www.inycom.es/buscamos-talento

Creation Date	2018-06-18T06:41:46.065Z
ID	ab760168-72c2-11e8-b107-eb9bf828b911

Application List

Application	Confidence	Version	Categories
Amazon EC2	100		Web Servers
Apache	100	2.4.27	Web Servers
Drupal	100	8	CMS
Font Awesome	100		Font Scripts
Google Analytics	100	UA	Analytics
Google Font API	100		Font Scripts
OpenSSL	100	1.0.2k	Web Server Extensions
PHP	100	7.0.25	Programming Languages
Bootstrap	100		Web Frameworks
jQuery	100	3.2.1	JavaScript Frameworks
jQuery Migrate	100	3.0.0	JavaScript Frameworks
jQuery UI	100	1.10.2	JavaScript Frameworks

Figura 21 Resultado análisis wappalyzer

una plantilla (Figura 22), que a su vez le pasa el informe al bot de Telegram para notificarnos, siendo este informe almacenado en el servidor para su posterior visualización y la obtención de estadísticas como la evolución en la cantidad y gravedad

Plantilla utilizada:

```
#[_REPORT.applications]
- #{$.name}
- Confidence: #{$.confidence}
- Version: #{$.version}
#[_REPORT.applications]
```

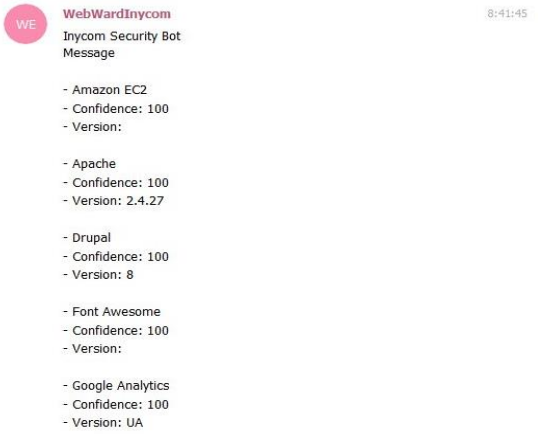


Figura 22 Plantilla y mensaje generado

de las vulnerabilidades.

Además, cada proyecto tiene asignados una serie de dependencias y sistemas que necesita para su funcionamiento. Estos pueden ser modificados de forma dinámica por los escáneres de seguridad, posibilitando en un futuro la edición de los modelos de amenazas por parte de la plataforma, evitando tener que hacer este paso de forma manual a través de MTMT.

A la hora de trabajar con los modelos de amenazas, la plataforma permite subir nuestros propios modelos y plantillas, así como informes generados por los propios modelos.

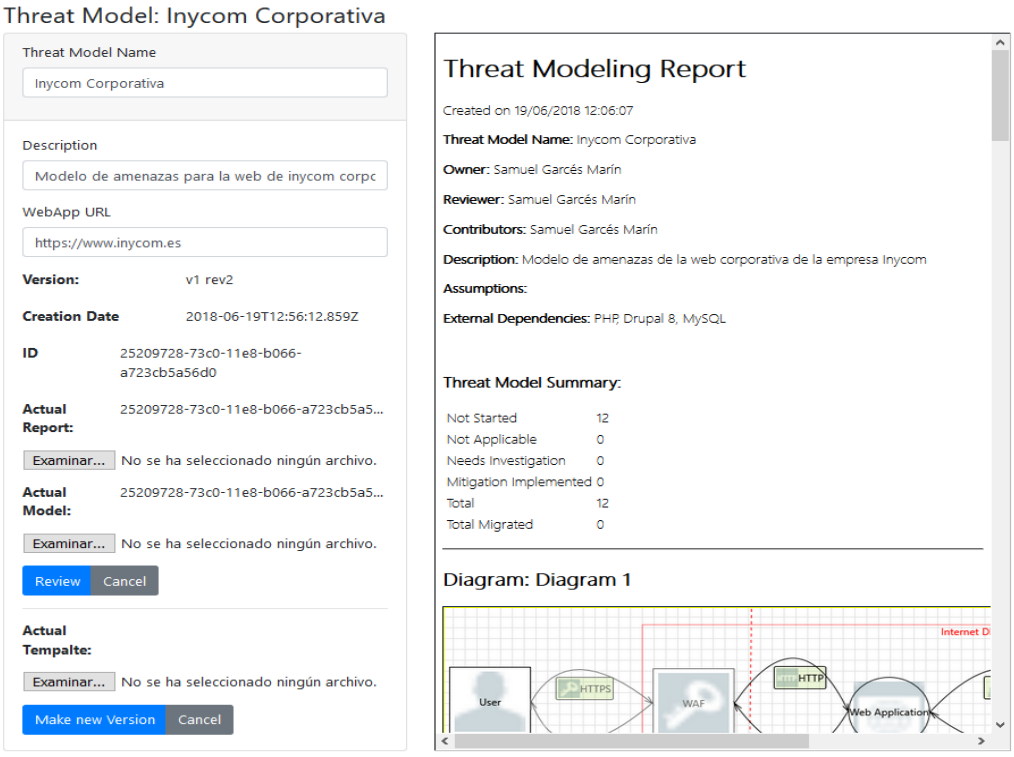


Figura 23 Gestión de versiones de modelos de amenaza

Para la gestión de las distintas versiones se utiliza una sencilla interfaz web, así como revisarlas y corregir errores, como se puede apreciar en la Figura 23. El versionado se realiza al actualizar la plantilla en la que está basado el modelo, mientras que al actualizar el modelo se genera una revisión de este.

Cuando se utiliza el escáner de seguridad *Arachni*, los informes generados son más profundos y de mayor complejidad que los de *Wappalyzer*, pudiendo tardar varias horas o incluso días en llevarse completamente a cabo.

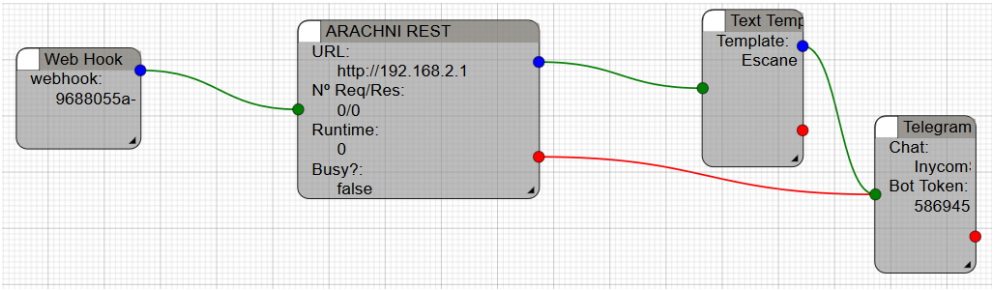


Figura 24 Pipeline solo para Arachni

Para realizar el análisis utilizaremos un perfil menos agresivo como un perfil orientado a producción, el cual no nos dará información sobre *XSS* o *SQLInjection*.

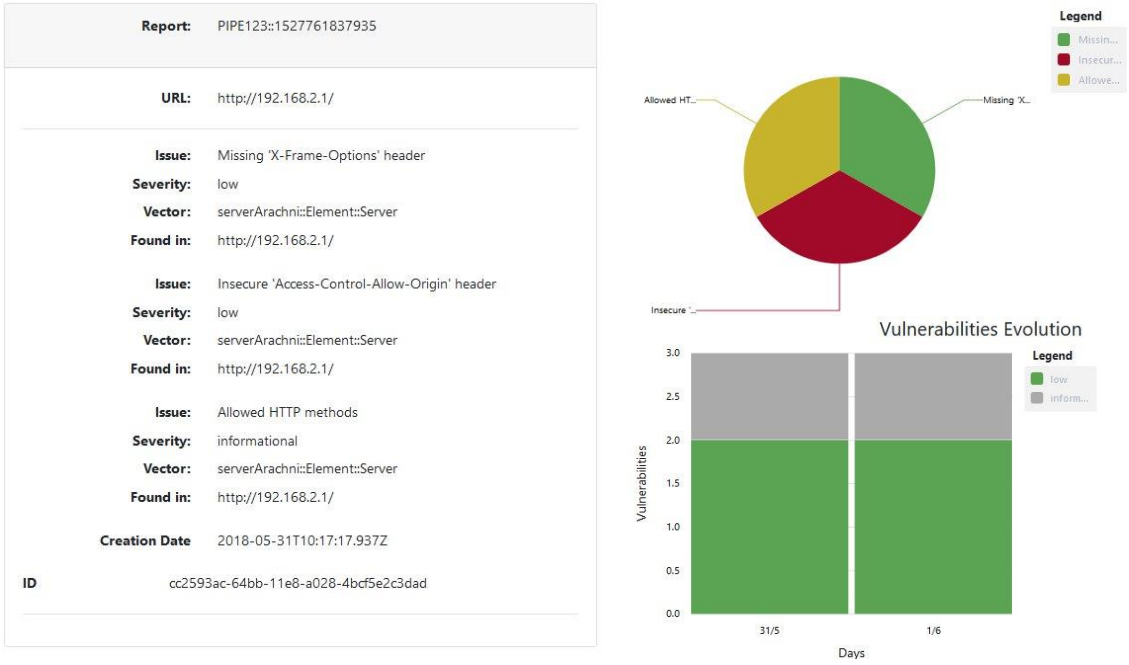


Figura 25 Informe generado por Arachni

Cada informe tiene asociados una lista de problemas (*issues*), clasificados según su gravedad (*severity*), almacenando también dónde se encontraron dichas vulnerabilidades y en qué prueba se encontró la vulnerabilidad (vector).

En la página principal de la plataforma (*dashboard*) podemos encontrar información sobre las vulnerabilidades encontradas a lo largo del tiempo y su evolución del proyecto actual, así como un gráfico sobre el porcentaje de vulnerabilidades de cada tipo encontradas (Figura 25).

Los escáneres de vulnerabilidades no siguen un esquema común sobre las propiedades que devuelven en el informe que generan, siendo necesario crear una plantilla para cada uno de ellos como se ve en las Figura 22 y Figura 26.

Plantilla utilizada:

```
Escaneo finalizado para: #{_REPORT.url}
Vulnerabilidades encontradas:
#[_REPORT.issues]
- #{$.name}
  Gravedad: #{$.severity}
  Encontrado por #{$.vector}
  Localizado en #{$.url}
#[_REPORT.issues]
```

Inycom Security Bot
Message
Escaneo finalizado para: http://52.16.0.37/inycom/
Vulnerabilidades encontradas:

- Missing 'X-Frame-Options' header
  Gravedad: low
  Encontrado por serverArachni::Element::Server
  Localizado en http://52.16.0.37/inycom/

- Interesting response
  Gravedad: informational
  Encontrado por serverArachni::Element::Server
  Localizado en http://52.16.0.37/inycom/

- Interesting response
  Gravedad: informational
  Encontrado por serverArachni::Element::Server
  Localizado en http://52.16.0.37/inycom/Arachni-78978c1b937c4355d23172a4667e5ea6

Figura 26 Notificación generada por la plantilla en Telegram

Se han realizado varios análisis de seguridad a la web corporativa tanto en preproducción como en producción, pero sin entrar en mayor profundidad para evitar saturar la web, habiéndose obtenido vulnerabilidades sencillas y muchos falsos positivos que deberán reducirse en futuras versiones.

```
Escaneo finalizado para: http://www.inycom.es/
Vulnerabilidades encontradas:

- Missing 'X-Frame-Options' header
  Gravedad: low
  Encontrado por serverArachni::Element::Server
  Localizado en http://www.inycom.es/

- Interesting response
  Gravedad: informational
  Encontrado por serverArachni::Element::Server
  Localizado en http://www.inycom.es/localstart.asp

- Interesting response
  Gravedad: informational
  Encontrado por serverArachni::Element::Server
  Localizado en http://www.inycom.es/%3Cmy_tag_94516757f4b3058678d1f7f2232158a4/%3E

- Interesting response
  Gravedad: informational
  Encontrado por serverArachni::Element::Server
  Localizado en
http://www.inycom.es/%3E%22'%3E%3Cmy_tag_94516757f4b3058678d1f7f2232158a4/%3E

- Interesting response
  Gravedad: informational
  Encontrado por serverArachni::Element::Server
  Localizado en http://www.inycom.es/Arachni-94516757f4b3058678d1f7f2232158a4
```

Figura 27 Lista de vulnerabilidades encontradas en un análisis pasivo

Entre los falsos positivos anteriormente mencionados están:

- *Interesting response* en: http://www.inycom.es/localstart.asp puesto que la tecnología es PHP y no ASP.
- *Interesting response* en: http://www.inycom.es/%3Cmy_tag_94516757f4b3058678d1f7f2232158a4/%3 , lleva a una página de error 404 no encontrado.
- *Interesting response* en: http://www.inycom.es/Arachni-94516757f4b3058678d1f7f2232158a4: debido a que no va a haber ninguna ruta llamada Arachni.

Los resultados nos muestran que no podemos esperar un 100% de fiabilidad, pero nos permite tener un cierto margen de seguridad sobre todo en páginas web hechas sin utilizar CMS o sistemas especializados, en los que puede puede existir la duda sobre la seguridad del sistema. No solo eso, también podemos encontrar malas configuraciones realizadas como la falta de la cabecera *X-Frame-Options* que se mostraba en el extracto de la Figura 27, o, servirnos de alerta al detectar un nuevo falso positivo, que antes no estaba, que puede ser debido a una mala configuración de la web.

6 Conclusiones y futuros desarrollos

6.1 Conclusión

Debido al aumento de la complejidad en las aplicaciones web, hoy en día la seguridad se hace un requisito indispensable, pero difícil de mantener a lo largo del ciclo de vida de un proyecto y más cuando los plazos de entrega son muy cortos. Para simplificar y facilitar el mantenimiento de la seguridad en los diferentes proyectos web se ha realizado este proyecto. Aunque es cierto que sigue siendo necesario un trabajo previo para realizar un análisis de seguridad, se ha probado que un usuario sin experiencia en el uso de la aplicación y con detalles básicos de su funcionamiento, es capaz en pocos minutos de lanzar correctamente un análisis de seguridad.

Este proyecto ha reducido la incertidumbre sobre si un proyecto web tiene vulnerabilidades de seguridad, ya que además de detectar las vulnerabilidades, permite llevar un inventariado de las dependencias de una aplicación. Pudiendo programar análisis de seguridad periódicos para cada proyecto, configurar los análisis con perfiles reutilizables y seguir trabajando mientras la plataforma reúne la información para que el analista de seguridad se centre en tareas complejas y que no pueden ser automatizadas.

Actualmente, la plataforma está operativa y monitorizando varios proyectos de la empresa, haciéndose un seguimiento programado semanal, tanto de la web corporativa de la empresa, como de las webs de clientes, en los que se ha visto que incluir los escáneres de seguridad apenas cuesta un esfuerzo extra de unos minutos.

Otra ventaja que cabe destacar es la presentación de la información de cada vulnerabilidad gracias a la utilización de un bot de mensajería para *Telegram*, mostrándose toda la información importante sobre la vulnerabilidad de forma clara e intuitiva y almacenándose en el servidor para futuras revisiones como en la Figura 28.

Además, abrimos la posibilidad a los usuarios de crear sus propios módulos y nodos en función de las necesidades individuales de cada proyecto pudiéndolos comunicar con nodos ya existentes sin requerir un esfuerzo extra, replicando configuraciones ya existentes, con mínimas modificaciones, en otros proyectos.

En total, la plataforma ha hecho uso de unas 50.000 líneas para el servidor y 19.000 para la página web (si contamos ficheros generados por *Angular* hace un total de 200.000).

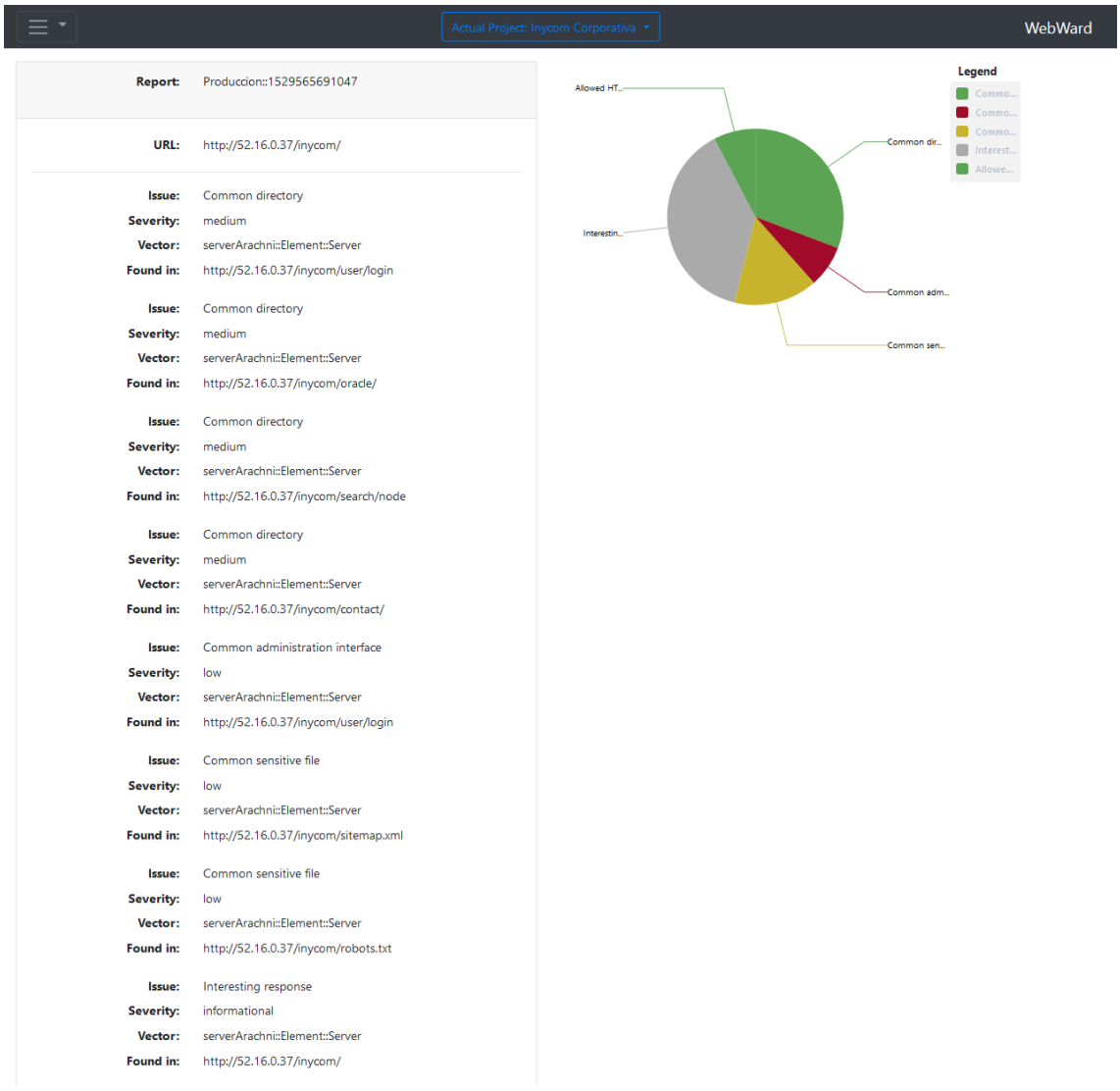


Figura 28 Informe completo de web en producción

6.2 Futuros desarrollos

El escenario que se ha utilizado sólo recoge el uso del escáner de seguridad *Arachni* y el escáner de dependencias *Wappalyzer*. El principal trabajo futuro será la extensión de la plataforma con nuevos escáneres de seguridad.

Además de ello, en un futuro se plantea:

- Simplificar la plataforma separando de ella la lógica y los escáneres utilizando para ello *plugins*, de forma que *Arachni* no esté incluido desde un principio y sea necesario añadirlo, dejando más limpia la aplicación.
- Aumentar el ecosistema de módulos y *plugins* para nuevos escáneres de seguridad y nuevas funcionalidades.
- Integrar completamente *Microsoft Threat Modeling Tool* en la plataforma, abriendo la posibilidad a que algún nodo del pipeline pueda modificar los modelos de amenaza, creando revisiones automáticamente cuando se detecten ciertas vulnerabilidades.

- Permitir la integración de la plataforma con *Jenkins* en la que realizar los análisis de seguridad, para, posteriormente, obtener los resultados y guardarlos en la propia plataforma, como se venía haciendo hasta ahora.
- Reducir el número de falsos positivos cruzando informes de varias herramientas.
- Mejorar la interfaz web de forma que sea más intuitiva y sencilla de utilizar para cualquier usuario, añadiendo además una ayuda y guía de uso.

Bibliografía

[1] S. McConnell, "Typical Errors," in *Code Complete: A Practical Handbook of Software Construction, Second Edition*, Microsoft Press, 2004, p. 517.

[2] OWASP Top 10, "OWASP Top 10," 2017. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. [Accessed 15 Junio 2018].

[3] G. Wagner, "OWASP TOP 10 Data Call Submission," 2016. [Online]. Available: https://www.vantagepoint.sg/blog/69-owasp-top-10-data-call-submission. [Accessed 20 Junio 2018].

[4] OWASP, "OWASP AppSec Pipeline," 2018. [Online]. Available: https://www.owasp.org/index.php/OWASP_AppSec_Pipeline. [Accessed 15 Junio 2018].

[5] . G. Kim, . J. Humble, J. Willis and . P. Debois, The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations, vol. 2, IT Revolution, 2016, p. 480.

[6] D. Hohnstein, "Penetration testing AWS storage," 20 Abril 2017. [Online]. Available: https://rhinosecuritylabs.com/penetration-testing/penetration-testing-aws-storage/. [Accessed 15 Junio 2018].

[7] OWASP, "Vulnerability Scanning Tools," 2018. [Online]. Available: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools. [Accessed 15 Junio 2018].

[8] S. Garcés, "Web Ward," 20 Junio 2018. [Online]. Available: https://github.com/SecSamDev/WebWardRest. [Accessed 20 Junio 2018].

[9] J. Stanganelli, "Which Threat Risk Model Is Right for Your Organization?," 19 Septiembre 2016. [Online]. Available: https://www.esecurityplanet.com/network-security/which-threat-risk-model-is-right-for-your-organization.html. [Accessed 16 Junio 2018].

[10] Owasp, "Application threat modeling," 2018 Junio 11. [Online]. Available: https://www.owasp.org/index.php/Application_Threat_Modeling. [Accessed 2018 Junio 20].

[11] Microsoft, "Security Development Lifecycle," 2018. [Online]. Available: https://www.microsoft.com/en-us/sdl. [Accessed 15 Junio 2018].

[12] R. Santos, "Microsoft Threat Modeling Tool," 2017 Agosto 17. [Online]. Available: https://docs.microsoft.com/es-es/azure/security/azure-security-threat-modeling-tool.

[13] M. Douglas, "Security CI/CD Validation," 4 Abril 2018. [Online]. Available: https://docs.microsoft.com/en-us/vsts/articles/security-validation-cicd-pipeline?view=vsts. [Accessed 15 Junio 2018].

[14] Jenkins, "Jenkins User Documentation," 2018. [Online]. Available: https://jenkins.io/doc/. [Accessed 15 Junio 2018].

[15] OWASP, "OWASP Zed Attack Proxy," 2018. [Online]. Available: OWASP Zed Attack Proxy Project. [Accessed 15 Junio 2018].

[16] T. Laskos, "Arachni Documentation," 13 Septiembre 2014. [Online]. Available: https://github.com/Arachni/arachni/wiki. [Accessed 15 Junio 2018].

[17] OWASP, "Application Security Verification Standard Project," 2018. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project. [Accessed 15 Junio 2018].

[18] M. Rohr, "Open Threat Modeling Templates," 30 Junio 2016. [Online]. Available: https://github.com/matthiasrohr/OTMT. [Accessed 15 Junio 2018].

[19] Telegram, "Telegram Bot API," 2018. [Online]. Available: https://core.telegram.org/bots/api. [Accessed 15 Junio 2018].

[20] Docker, "Container platform provider," 2018. [Online]. Available: https://www.docker.com/. [Accessed 15 Junio 2018].

[21] Kubernetes, "Production-Grade Container Orchestration," 2018. [Online]. Available: https://kubernetes.io/. [Accessed 15 Junio 2018].

[22] Nodejs, "Nodejs Javascript Runtime," 2018. [Online]. Available: https://nodejs.org/en/. [Accessed 15 Junio 2018].

[23] Angular, "Mobile and desktop framework," 2018. [Online]. Available: https://angular.io/. [Accessed Junio 2018].

[24] T. Markova, "Deploy code from Git automatically," 25 Abril 2017. [Online]. Available: https://dzone.com/articles/deploy-code-to-containers-from-git-automatically-1. [Accessed 15 Junio 2018].

[25] T. Laskos, "Arachni REST API," 16 Septiembre 2016. [Online]. Available: https://github.com/Arachni/arachni/wiki/REST-API. [Accessed 15 Junio 2018].

[26] OWASP, "OWASP WebGoat," 2018. [Online]. Available: https://github.com/WebGoat/WebGoat. [Accessed 15 Junio 2018].

[27] SecTool, "Price and Feature Comparison of Web Application Scanners," 18 Septiembre 2016. [Online]. Available: http://sectoolmarket.com/price-and-

feature-comparison-of-web-application-scanners-unified-list.html. [Accessed 15 Junio 2018].

[28] CentOS, "Documentación CentOS," 2018. [Online]. Available: https://wiki.centos.org/. [Accessed 15 Junio 2018].

[29] V. Munjal, "Configuring Reverse Proxy for Node.js application using Apache," 3 Agosto 2017. [Online]. Available: https://linuxtogether.org/configuring-reverse-proxy-for-node-using-apache-mod-proxy/. [Accessed 15 Junio 2018].

[30] PM2, "Advanced, production process manager for Node.js," 2018. [Online]. Available: http://pm2.keymetrics.io/. [Accessed 15 Junio 2018].

ANEXO A: Glosario de Términos

- Angular: framework de diseño que permite la creación de páginas web.
- Apache: servidor web.
- Arachni: analizador de vulnerabilidades en aplicaciones web.
- AWS: Servicio de la empresa Amazon para la creación y gestión de servidores virtuales privados.
- BB.DD: base de datos que almacena información.
- Bot de mensajería: Programa informático con el que interactuar con usuarios físicos a través de una aplicación de mensajería.
- CentOS: sistema operativo basado en Linux.
- CD: el despliegue continuo consiste en, una vez acabada la parte de integración continua, lograr que el programa desarrollado se ponga a funcionar automáticamente evitando tareas manuales por parte del operador. Acrónimo de *Continuous Deployment*.
- CI: la integración continua consiste en unir todas las copias de un programa diariamente para comprobar que cada componente sigue funcionando correctamente. Acrónimo de *Continuous Integration*.
- CI/CD: acrónimo de Integración Continua y Despliegue Continuo.
- CMS: acrónimo de Sistema de Gestion de Contenidos (Content Managment System). Permite manejar el contenido y el diseño de una página web, guardando toda la configuración en una base de datos.
- Contenedor virtual: permite ejecutar una aplicación en un entorno virtual.
- Desarrollador: cualquier persona implicada en el proceso de creación de un software.
- DevOps: Metodología ágil centrada en la colaboración entre distintos departamentos de una empresa para desarrollar aplicaciones informáticas más robustas, con menos fallos y de forma más rápida.
- DevOpsSec: similar a DevOps, pero añadiendo Seguridad en el proceso.
- Docker: sistema de creación y administración de contenedores virtuales de aplicaciones.
- Escalado de aplicación: el escalado horizontal permite tener varias instancias de una misma aplicación de forma distribuida, el escalado vertical se centra en mejorar los recursos de una sola máquina.
- Exploit: mecanismo con el que aprovecharse de una vulnerabilidad en el software.
- GET: petición HTTP relacionada con la búsqueda de recursos.
- Git: software de control y mantenimiento de versiones de código fuente.
- Hash: conjunto de bytes de tamaño fijo al que se puede mapear un conjunto arbitrario de bytes de cualquier tamaño.
- HTTP: Protocolo de transferencia de hipertexto.

- HTTPS: Similar a HTTP, pero cifrado.
- IPTABLES: comando en Linux para configurar el firewall del sistema.
- Javascript: Lenguaje de programación interpretado.
- Json: formato de almacenamiento de información. Acrónimo de *JavaScript Object Notation*.
- Kernel: programa encargado de manejar peticiones I/O y traducirlas a instrucciones de procesador.
- Kubernetes: sistema de administración de contenedores virtuales en forma de cluster para su escalado horizontal.
- Links: navegador web basado en consola de comandos para Linux.
- Linux: familia de sistemas operativos.
- Minikube: Máquina virtual que ejecuta en su interior un cluster de Kubernetes.
- MTMT: herramienta para la creación de modelos de amenazas. Acrónimo de *Microsoft Threat Modeling Tool*.
- MVC: acrónimo de Modelo Vista Controlador. Sistema que divide la lógica de una aplicación en subsistemas independientes que controlan el contenido y su edición (modelo), la interfaz visual (vista) y la adaptación del contenido a la vista (controlador).
- NodeJS: sistema que permite ejecutar código escrito en Javascript fuera del navegador y más concretamente en un servidor.
- PHP: lenguaje de programación interpretado.
- Pipeline: proceso que sigue una secuencia de elementos en los que la entrada de información de uno de ellos es la salida de uno o varios ejecutados con anterioridad a este.
- POST: Petición HTTP relacionada con la creación de contenido.
- PostgreSQL: Base de datos SQL.
- Plugin: componente de software que no forma parte, inicialmente, de una aplicación, pero que puede instalarse en ella para aumentar la funcionalidad de esta.
- REST: interfaz de comunicación entre sistemas basado en HTTP. Acrónimo de *Representational State Transfer.*
- Ruby: lenguaje de programación.
- Sandbox: ejecución de una aplicación de forma aislada.
- Script: programa normalmente simple que suele ser interpretado.
- Skype: aplicación de mensajería instantánea.
- SO: Sistema Operativo.
- SQL: lenguaje para gestionar bases de datos. Acrónimo de *Structured Query Language*.
- SQLInjection: vulnerabilidad similar a XSS, pero de forma que el código insertado se realiza utilizando SQL y ejecutándose en la propia base de datos.

- SVG: lenguaje de etiquetas basado en XML para la creación de contenido gráfico.
- Telegram: servicio de mensajería instantánea que además permite mandar mensajes utilizando software por medio de los llamados bots de mensajería.
- Threat model: modelo de amenazas.
- Template: en español plantilla.
- Test Unitario: también conocido como *Unit Test*, permite probar componentes concretos de una aplicación de forma aislada sin ejecutar toda ella para comprobar que un cambio no modifica el funcionamiento esperado del componente.
- URL: permite acceder a contenido en una página web. Acrónimo de *Uniform Resource Locator*.
- VM: Máquina virtual. Acrónimo de Virtual Machine.
- Vulnerabilidad: punto débil de un sistema informático que permite a un atacante comprometer la integridad, disponibilidad o confidencialidad de este.
- Wappalyzer: sencillo analizador de vulnerabilidades enfocado en descubrir las tecnologías utilizadas en una página web.
- Web Hook: permiten intercambiar información entre aplicaciones de forma asíncrona. Basan su funcionamiento en REST haciendo que la respuesta no se envíe inmediatamente, sino que se envíe en un futuro a una URL concreta.
- XSS: vulnerabilidad común en páginas web consistente en inyectar código HTML malicioso. Acrónimo de *Cross* (X) *Site Scripting*.

Términos Propios

En el contexto del uso de la plataforma desarrollada:

- **Módulo**: definen el comportamiento de los nodos.
- **Nodo**: cada uno de los elementos individuales que componen un pipeline.
- **Pipeline**: sistema con nodos interconectados para programar análisis de seguridad.
- **Plataforma**: consta tanto del servidor como de los escáneres de seguridad, bases de datos y, en definitiva, todos los servicios necesarios.
- **Proyecto**: proyecto web en desarrollo en la empresa, tendrá asociados unos modelos de amenazas y unos pipelines. Estos pipelines estarán orientados a la realización de tareas concretas y diferenciables, como el análisis por separado de los entornos de preproducción y producción.
- **Servidor**: código en ejecución que contiene el servidor REST, el controlador de pipelines o ambos.

ANEXO B: Security Pipeline

Vamos a proceder a definir todos los detalles sobre el funcionamiento de un pipeline y los nodos que poseen.

Creación de Pipelines

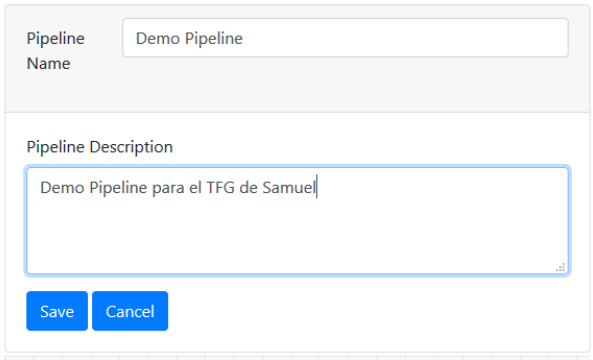


Figura 29 Creación de un nuevo Pipeline

El ciclo de vida de un pipeline comienza con su creación utilizando el *frontend* (Figura 29). Este hace una petición POST al servidor REST con la información para su creación. A partir de este momento podemos empezar a añadirle nodos al pipeline. Este pipeline empezará en estado inactivo para evitar que un sea inicializado en el *backend* y estará asociado a un proyecto Web.

Podemos elegir añadir un nodo con parámetros por defecto o añadir un nodo que hemos almacenado nosotros y que ya está configurado (Figura 30).

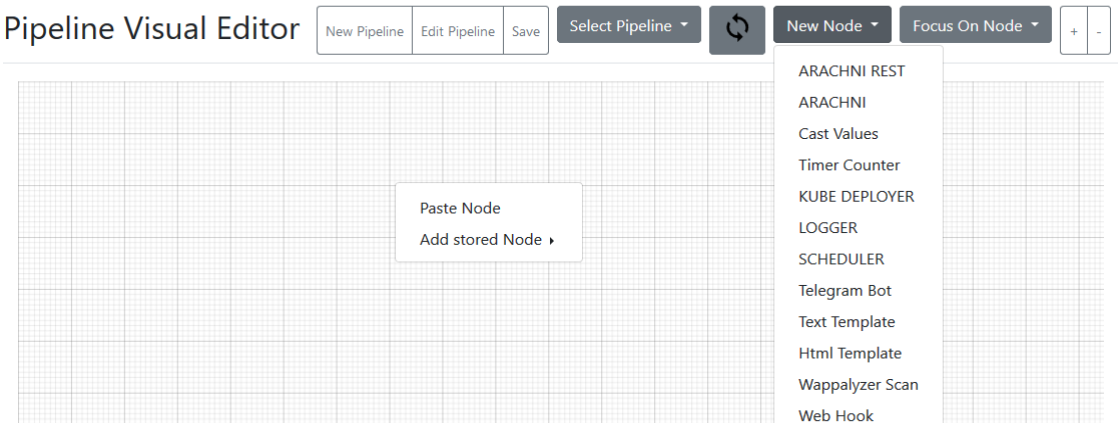


Figura 30 Creación de nodos en un pipeline

Vamos a explicar el funcionamiento de un nodo sencillo y versátil como lo es el nodo **Web Hook** (Figura 31), el cual permite que agentes externos interactúen con el pipeline.

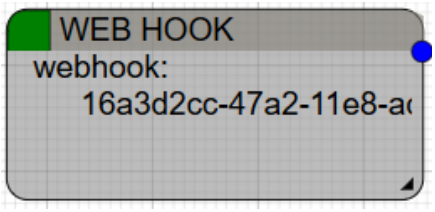


Figura 31 Ejemplo de Nodo

Este tiene asociados una serie de propiedades, una lista de parámetros de entrada, otra lista de parámetros de salida y aunque no se permite su visualización al

estar almacenados en la BB.DD, también dispone de una lista de los últimos parámetros que ha recibido del nodo anterior y los últimos parámetros que le ha pasado al siguiente nodo, un ejemplo de ello se puede ver en la Figura 32.

Un parámetro/propiedad tiene asociado un nombre, un alias (*nickname*), el tipo de parámetro que almacena y si la propiedad es opcional. El nombre de una propiedad normalmente empieza con una "_" para indicar que este parámetro no debe ser eliminado al limpiar un pipeline.

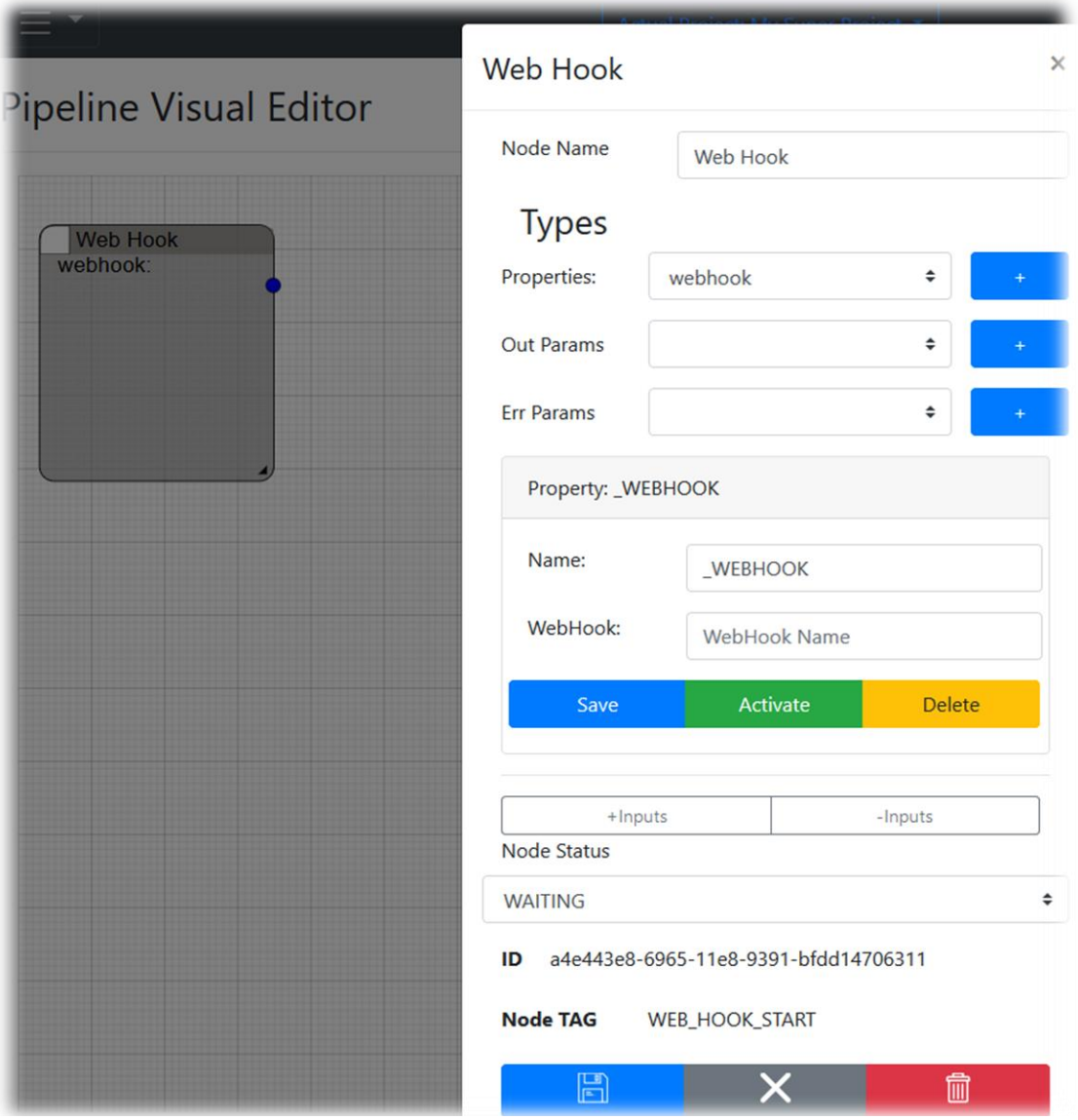


Figura 32 Editor de Nodos

Un nodo viene con varias propiedades que necesita para funcionar, pero es posible añadir otras propiedades que modifiquen el comportamiento, no solo del nodo, sino también de todo el Pipeline.

Los parámetros de salida y de error únicamente son para permitir que el sistema que controla los nodos pueda redirigir parámetros de la entrada hacia la salida (parámetros que vienen de otros nodos) o limitar los parámetros que va a recibir de otros nodos.

Los tipos de parámetros que se aceptan son muy variados y permiten su extensión con nuevos tipos (aunque el sistema no asegurará que los datos almacenan lo que dicen almacenar). Por ejemplo, el tipo "PORT" almacenara números entre 0 y 65555 que corresponde al rango de puertos de los protocolos TCP y UDP y un valor no comprendido en este rango o de tipo texto hará que este parámetro no sea pasado al nodo.

Una vez creado nuestro pipeline, podemos cambiar su estado de "inactivo" a "esperando". Todos los controladores de pipeline que están ejecutándose en diferentes servidores comenzarán a comprobar si hay algún pipeline activo y que no sea propiedad de otro controlador (declaran la titularidad de dicho pipeline actualizando el pipeline en

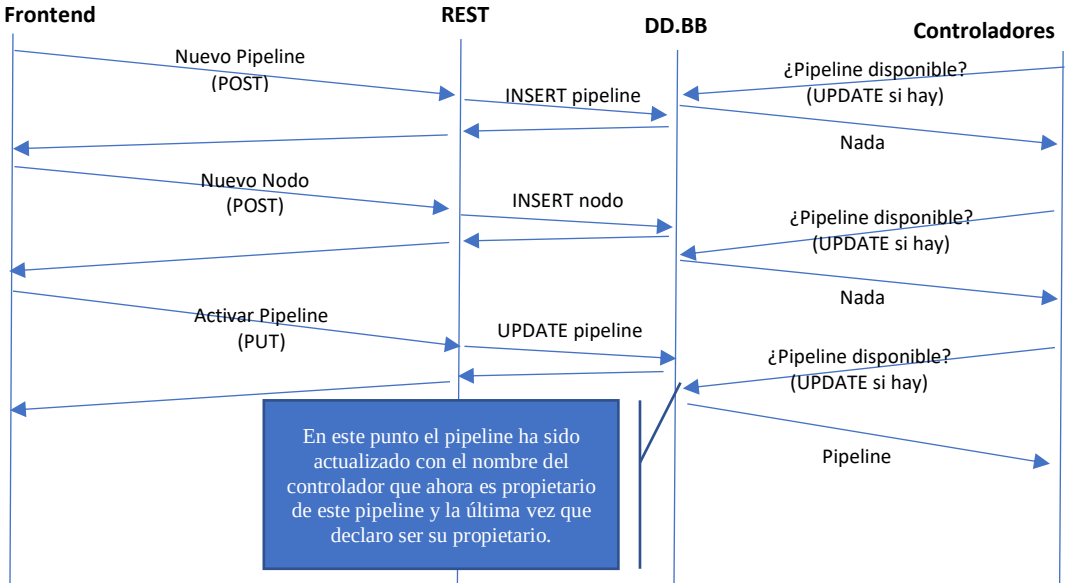


Diagrama 3 Proceso de creación de un pipeline

la BB.DD con el campo "owner"=nombre del controlador y "last_aquired" = fecha actual) haciendo peticiones a la base de datos para actualizar un pipeline que no haya sido actualizado en más de 30 segundos (tiempo configurable). Un resumen más sencillo de todo el proceso se encuentra en Diagrama 3.

Si por casualidad un controlador obtiene un pipeline de otro controlador que aún es el propietario, pero que por cualquier razón no ha sido capaz de informar de que aún lo posee, entonces, gracias a que en cada actualización de un pipeline debe indicarse el nombre del propietario que hace la actualización, si este supuesto propietario no es el propietario actual, entonces no se actualiza y este deja de trabajar en dicho pipeline.

Funcionamiento de los Pipeline Nodes

En este apartado nos centraremos en el funcionamiento de los nodos desde que son creados por el *frontend* hasta que acaban su ejecución en el *backend*.

Anteriormente hemos comentado que un nodo se crea realizando una al servidor REST (HTTP POST) y que un controlador de pipelines va haciendo peticiones a lo largo del tiempo para comprobar si algún pipeline está activo y no tiene propietario.

Pues bien, una vez que obtiene un pipeline, este controlador busca todos los nodos asociados a dicho pipeline y asigna a cada uno de ellos un script almacenado cuyo nombre de módulo es el mismo que el de la etiqueta del nodo (TAG). Estos scripts son compilados por el motor V8 (*javascipt*) de forma que no haya que reinterpretar un script para cada nodo, logrando también un ahorro de memoria RAM, al haber tantas instancias en memoria de los scripts como módulos y no como nodos.

Cuando un nodo se activa, el controlador le asigna unos parámetros de entrada y le pasa una serie de objetos (*sandbox*), librerías que puede utilizar en función de lo que requiere el módulo, así como una interfaz para comunicarle al controlador si ha acabado su ejecución satisfactoriamente, si aún no ha acabado, si hay algún error, leer parámetros o asignarlos.

La ejecución de la lógica de los nodos se realiza utilizando una máquina virtual dentro del motor V8 de *javascript*, de forma que se puede evitar ejecuciones indefinidas poniendo como límite de tiempo ejecuciones de máximo un segundo, de tal manera que cualquier función ejecutada dentro de dicho *sandbox* quedará interrumpida, imposibilitando que el script acceda a recursos que no le corresponden.

Hay que indicar, no obstante que en la programación asíncrona se pueden pasar funciones como parámetros a otras funciones, siendo estas últimas ejecutadas fuera del *sandbox* y no siendo posible su interrupción. Para solventarlo, la propia interfaz con el controlador incorpora métodos de comprobación que permiten saber si la ejecución ha finalizado, no permitiendo la modificación posterior del nodo.

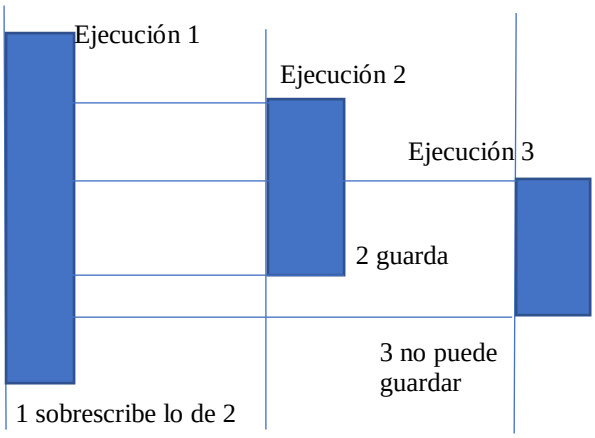


Diagrama 4: Ejecuciones de un mismo nodo solapadas

Es posible, a pesar de ello, modificar la información de un nodo a pesar de haber acabado la ejecución. Para ello hay que pasar un indicador del tiempo en que se empezó la ejecución de la función que realiza la modificación. De esta forma solo la información que almacena la primera ejecución es la que se mantiene en caso de existir sobre escritura.

En el Diagrama 4 se puede ver que a pesar de que la ejecución 3 es la última en ejecutarse, no puede almacenar nada porque la segunda ejecución ya ha guardado datos y es una ejecución más antigua. Pero la ejecución 1, al acabar, como es más antigua que la 2, puede guardar datos y hasta sobrescribir lo almacenado por la ejecución 2.

Esto solo debe utilizarse en los casos en los que la ejecución de la lógica de los nodos se hace cada poco tiempo dando como resultado el solape de ejecuciones, así como en casos en los que la invocación de comandos en la interfaz se realiza después de la finalización de la ejecución, como por ejemplo al hacer peticiones a un servidor que tarda un tiempo en contestar. En este último caso hay que diseñar el script teniendo

en cuenta que la invocación del comando para finalizar la ejecución no se realice directamente tras el retraso, sino que se almacene una variable temporal y se espere a la siguiente ejecución (evitar múltiples llamadas para finalizar el nodo).

En este punto el lector puede preguntarse la razón de no utilizar hilos para ejecutar la lógica de los nodos, siendo que los scripts se podrían simplificar no requiriendo tener en cuenta en su diseño la separación de la lógica en iteraciones. La razón es bastante sencilla: aparte de que *nodejs* no soporta el uso de hilos (versiones experimentales incorporan *wokers* similares al funcionamiento de hilos), la ejecución por hilos habría añadido la problemática, no solo de sincronizar múltiples hilos entre sí vigilando que ningún hilo muera o tenga una ejecución indefinida y afecte al resto, sino también del consumo que poseen los hilos. No hay que perder de vista que la plataforma debe ser escalable horizontalmente para poder satisfacer necesidades futuras y que un controlador lance cientos de hilos al poseer varios pipelines puede ser un problema grave de consumo excesivo de recursos.

En nuestro caso, al imponer en la ejecución de un nodo un máximo de 1 segundo, hacemos que en cada iteración (ejecuciones sucesivas del mismo script) se ejecute solo una parte de la lógica total simplificando en gran medida el diseño del script (ejecuciones típicas del orden de 12 ms incluyendo inicialización).

Para permitir que el sistema se pueda adaptar a diferentes metodologías de trabajo hemos definido una serie de variables especiales que modifican el comportamiento general del nodo o del pipeline:

- **_SHOW**: Esta propiedad controla que las propiedades que se le mostrarán al usuario en el nodo (Figura 33).
- **_FORCES_END**: un nodo activado con esta propiedad en verdadero hace que todo el pipeline acabe (todos los nodos a END).
- **FORCES_RESTART**: Al activarse el nodo fuerza el reinicio del Pipeline y la cancelación del resto de nodos.
- **_UNTIL_END**: Hace que un nodo no puede activarse si está ya activo. Debe esperar a que se apague para volver a activarse.

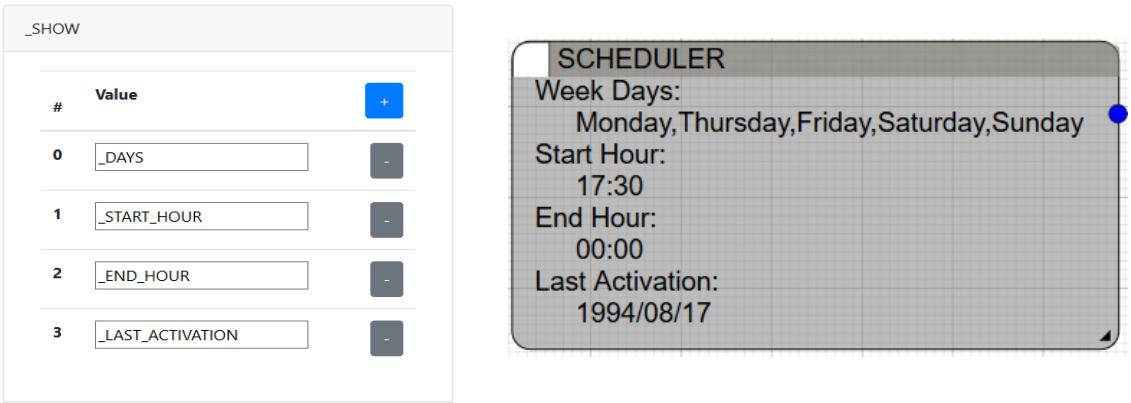


Figura 33 Modificando propiedades visibles

Todos los elementos que intervienen en la ejecución de un pipeline se pueden observar en la Figura 34.

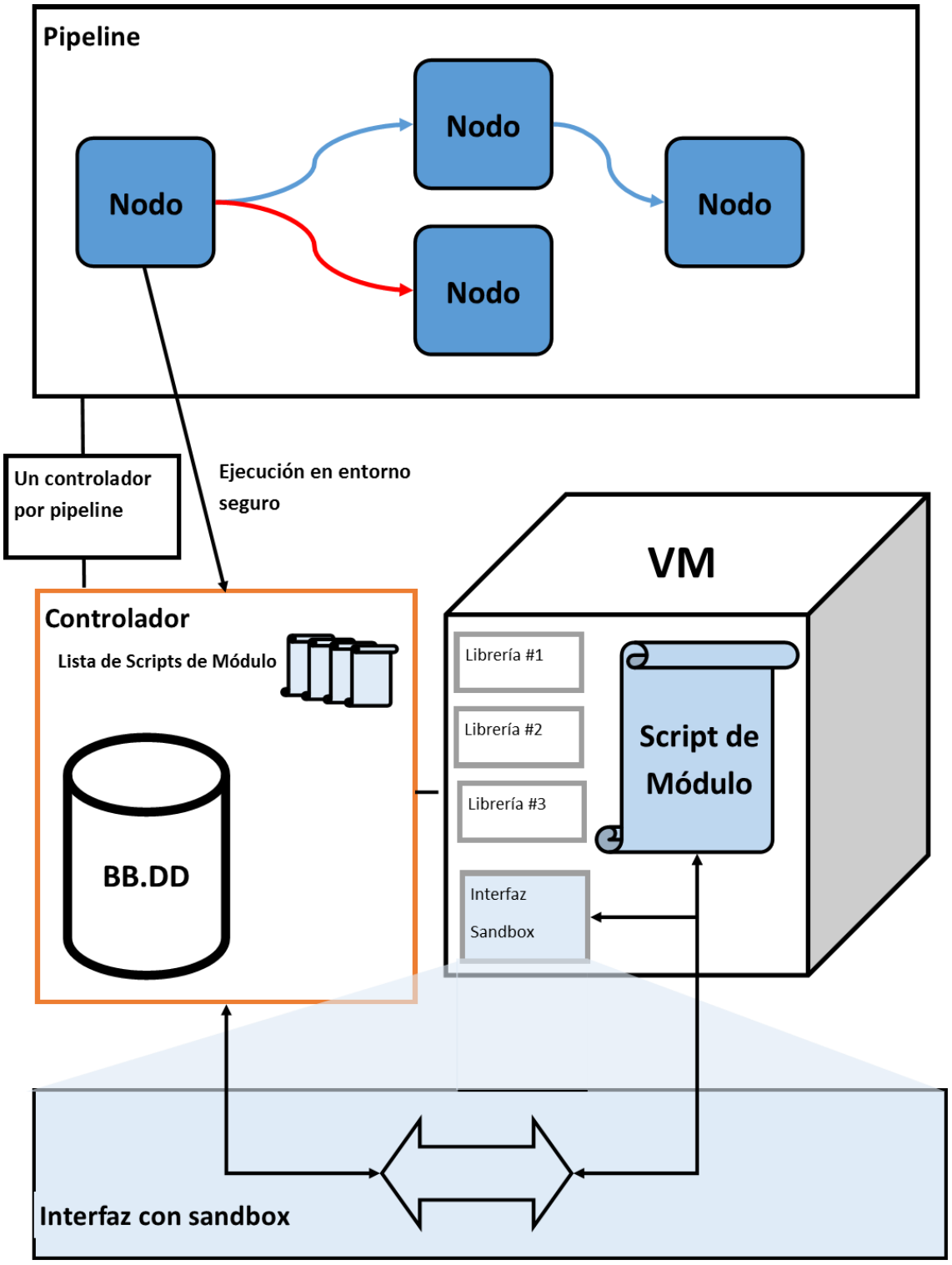


Figura 34 Interacción entre los objetos y componentes que intervienen en todo el proceso de ejecución de un pipeline

Librerías de nodo

Con este concepto de librería de nodo nos referiremos a todas aquellas librerías y funciones de las que hará uso un nodo durante su ejecución.

Estas librerías incluyen:

- **Consola:** actúa como salida estándar de un proceso. Todos los nodos comparten la misma salida que está redirigida a un fichero de texto.
- **Request:** para realizar peticiones HTTP.
- **ArachniREST:** para acceder al analizador de vulnerabilidades *Arachni* utilizando una interfaz basada en peticiones HTTP. Debe ser inyectada ya que la información sobre la ubicación del escáner A*rachni* la posee el propio controlador.
- **TelegramBot**: para realizar notificaciones por medio de un bot de *Telegram*. Para crear un bot se requiere el token de acceso y el nombre del chat al que enviar mensajes.
- **ReportStore:** para almacenar un análisis de seguridad en la BB.DD. independientemente del escáner de seguridad utilizado.
- **ProjectInfo:** para modificar información relacionada con dependencias de un proyecto. Debe ser inyectada con la información sobre el proyecto.
- **Wappalyzer:** que permite buscar las dependencias de una página web.
- **Require:** para acceder a librerías propias de nodejs. Desaconsejado su uso, excepto por causas mayores como la utilización de métodos definidos en otros scripts.

Algunas de estas librerías requieren ser “inyectadas” dentro del nodo, es decir, el controlador debe asignarles unos parámetros previos, como por ejemplo direcciones IP o contraseñas. Dicha información no debe almacenarse en los nodos o sencillamente resulta más sencillo que dicha información la proporcione el propio controlador.

Por ejemplo: un nodo no debería por qué saber el nombre del proyecto al que está asociado el pipeline en el que estamos, pero la librería **projectInfo** necesita dicha información para poder almacenar las nuevas dependencias que se han descubierto en un proyecto.

Módulos

Como se ha comentado en el anterior apartado, un módulo permite crear nodos a partir de una plantilla definiendo su comportamiento. Se ha diseñado toda la plataforma para poder crear módulos personalizados que no tienen por qué ser los que vienen incluidos, de forma que si un proyecto necesita, por ejemplo, enviar información utilizando el protocolo HTTP hacia un servidor de nuestro cliente, pueda crearse un pequeño módulo para ello.

Para facilitar el desarrollo de nuevos módulos se ha creado una herramienta especializada que hace las funciones de una mesa de trabajo (*workbench*)

permitiéndonos disponer de un entorno de pruebas sin tener que lanzar un servidor completo.

Más concretamente este *workbench* genera un controlador modificado donde cargar el nodo cuyo script queremos probar, asignándole unas propiedades por defecto y conectando su salida a un nodo vacío que simplemente imprimirá por pantalla el resultado, tanto si el nodo a probar sufre un error, como si finaliza correctamente.

```
\WW_MODULES\TEMPLATE> node .\test.js
1528293326825 :: FakePipeID :: RUN NODE TEXT_TEMPLATE-1-Node2Test
1528293326851 :: Saving status of node FakeNodeID
1528293326852 :: Saving status of node Node2Test
1528293326852 :: FakePipeID :: CYCLE 1528293326825 DURATION 27ms
1528293327853 :: FakePipeID :: RUN NODE FakeNode-1-FakeNodeID
      Scan finished for URL: http://192.168.2.1/
      Issues:

      Name: Missing 'X-Frame-Options' header
      Severity: low
      Vector: serverArachni::Element::Server
      Found in URL: http://192.168.2.1/

      Name: Insecure 'Access-Control-Allow-Origin' header
      Severity: low
      Vector: serverArachni::Element::Server
      Found in URL: http://192.168.2.1/

      Name: Allowed HTTP methods
      Severity: informational
      Vector: serverArachni::Element::Server
      Found in URL: http://192.168.2.1/
1528293327857 :: FakePipeID :: CYCLE 1528293327853 DURATION 4ms
```

Figura 35 Creación de nuevos módulos con workbench

En el extracto de la Figura 35 podemos ver que al probar el módulo TEMPLATE se imprime por pantalla un texto que ha obtenido rellenando una plantilla utilizando datos de un objeto en formato JSON. Podemos ver que en total el nodo ha utilizado 27ms desde que comenzó a rellenar la plantilla hasta que finalizó. En concreto el extracto corresponde a las pruebas realizadas durante el diseño del módulo de plantillas (TEXT y HTML TEMPLATE).

Los comandos disponibles en la interfaz de conexión de un nodo y que le son necesarios a un módulo se pueden consultar en la Tabla 1:

Tabla 1: Lista de métodos disponibles en la interfaz del módulo con el controlador

addProperties	addProperty	removeProperty
addPropertiesFuture	addPropertyFuture	removePropertyFuture
addOutParameters	addErrParameters	endInError
addOutParametersFuture	addErrParametersFuture	endInErrorFuture
endCycle	forceSaving	endInSuccess
endButStarted	forceSavingFuture	endInSuccessFuture

getInParams	getParams	getParam
getProperty	getStatus	registerService
removeService	getService	

Las funciones relacionadas con servicios permiten almacenar objetos en memoria (los parámetros se almacenan no solo en memoria sino en la BB.DD), sobre todo para objetos cuya inicialización requiere tiempo o dicho objeto mantiene una conexión con algún servidor remoto de forma que se mantenga entre distintos ciclos de ejecución de un mismo nodo.

Las funciones acabadas en "Future" hacen lo mismo que sus homónimas sin "Future" con la diferencia de que, si ha acabado la ejecución del script, la llamada no se realizará, a no ser que, se proporcione una marca de tiempo.

El nombre de las funciones de la interfaz es lo suficientemente descriptivo para entender su finalidad, aunque algunas de ellas requieren una mención especial:

- **EndCycle:** permite finalizar la ejecución de esta iteración.
- **ForceSaving:** permite forzar el guardado del nodo en la BB.DD. Útil cuando se ha obtenido un valor crítico, como por ejemplo un token de acceso a algún servicio y cuya pérdida implique dejar un proceso abierto (proceso zombie) consumiendo recursos en vano.
- **GetParam:** como su nombre indica permite obtener un parámetro, no obstante, cabe hacer la distinción con **GetProperty** puesto que el primero busca no solo en las propiedades del nodo sino también en los parámetros de entrada, de forma que un parámetro de entrada con el mismo nombre que una propiedad tiene preferencia frente a esta última.
- **EndButStarted:** permite finalizar con éxito un nodo, pero manteniéndolo en estado de funcionamiento. Es decir, llama a los nodos conectados a él como si hubiera finalizado, pero no lo hace.

ANEXO C: Instalación

Durante todo el desarrollo de la plataforma se han hecho pruebas tanto en un entorno local en un ordenador portátil como en un entorno virtualizado basado en *Minikube*. Las pruebas finales han sido llevadas a cabo en un servidor privado haciendo uso de los servicios de *Amazon Web Services*.

En concreto el servidor está basado en *CentOS 7* [28] sobre el que se han instalado todos los programas necesarios comenzando por la actualización de todos sus componentes, así como la instalación de la base de datos *PostgreSQL* y un servidor web *Apache* como balanceador de carga [29].

```
sudo yum update && sudo yum upgrade
sudo yum install postgresql10 postgresql10-server postgresql10-contrib
sudo /usr/pgsql-10/bin/postgresql-10-setup initdb
sudo systemctl start postgresql-10
sudo systemctl enable postgresql-10
# Ser root
sudo su
# Cambiar a usuario postgres
su -l postgres
#Conectarse a la base de datos
psql
CREATE EXTENSION IF NOT EXISTS “pgcrypto”;
CREATE EXTENSION IF NOT EXISTS “uuid-ossp”;
CREATE USER webward WITH SUPERUSER PASSWORD ‘webward’;
CREATE DATABASE webward;
GRANT ALL PRIVILEGES ON DATABASE webward TO webward;
```

Comprobamos que la base de datos está funcionando correctamente y continuamos con la instalación de *git*, *apache* y *links*:

```
sudo yum install git links httpd
sudo systemctl enable httpd.service
sudo systemctl start httpd.service
links http://localhost:80
```

Ahora instalaremos un controlador de versiones para *nodejs* llamado *NVM* que nos permitirá tener diferentes versiones instaladas al mismo tiempo:

```
curl https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash
nvm install node
# Siguiente linea para permitir el acceso al comando npm usando sudo
cat alias sudo=’sudo env PATH=$PATH:NVM_BR’ >> ~/.bashrc
# Instalar sistema de gestión y monitorización de aplicaciones
sudo npm install pm2 -g
```

Configuramos las variables de entorno a través del fichero *~/.bashrc* .

```
export PORT=”3000”
export PGHOST=”localhost”
export PGPORT=”5432”
export PGUSER=”webward”
export PGDATABASE=”webward”
export PGPASSWORD=”webward”
export ADMIN_PASS=”MyDefaultAdminPass”
```

Para la clonación del repositorio utilizaremos unos tokens de solo lectura y revocables para evitar introducir nuestro usuario y contraseña o utilizar la versión pública de *github*:

```
sudo git clone https://{user-token}:{pass-token}@gitlab.com/{URL codigo}
sudo git clone https://github.com/SecSamDev/WebWardRest.git
```

A partir de este punto cada vez que hagamos una modificación del código fuente en el repositorio, simplemente deberemos utilizar el comando ¨git pull¨en la carpeta en la que fue clonado el repositorio.
El código fuente ya está en nuestro servidor, pero este hace uso de librerías que deberemos instalar utilizando:

```
sudo npm install
```

Con el programa *pm2* [30], el cual hemos instalado anteriormente, lanzaremos nuestra aplicación nodejs de forma que se ejecute en cada reinicio:

```
pm2 start npm -- start
```

El anterior comando, que deberá ejecutarse en la carpeta de nuestra aplicación lanza el comando npm con el argumento start, la cual es una forma muy común de lanzar aplicaciones basadas en nodejs.
La configuración del servidor web Apache se hace modificando el archivo *httpd.conf*:

```
<VirtualHost *:80>
  ServerName webward.com
  ServerAlias www.webward.com
  DocumentRoot /var/www/webward/
  Options -Indexes
  ProxyRequests on
  ProxyPass /index.html !
  ProxyPass /public/ !
  ProxyPass / http://127.0.0.1:3000/
  ProxyPassReverse / http://127.0.0.1:3000/
</VirtualHost>
```

Hay que habilitar la redirección y también comprobar los permisos de SELinux:

```
sudo /usr/sbin/setsebool -P httpd_can_network_connect 1
# Desactivamos SELinux pues es dificil de configurar correctamente
sudo setenforce 0
```

Comprobamos que el servidor web está funcionando haciendo una petición desde nuestro navegador hacia la URL remota en la que está alojado nuestro servidor, iniciamos sesión con usuario *admin* y la contraseña definida en ADMIN_PASS: *MyDefaultAdminPass* (por defecto *admin*) y navegamos por las diferentes secciones para comprobar que las peticiones llegan a nuestro servidor *Nodejs*, pero que los archivos estáticos de la web son servidos por Apache.

En las nuevas versiones de *centOS*, se permite crear servicios personalizados haciendo posible que se ejecuten scripts en cada reinicio de la máquina. Nos valdremos de esto para crear un servicio que lance la instancia del escáner web *Arachni*.

```
# vi /etc/systemd/system/arachni.service
[Unit]
Description=Autoexecute Arachni locally in 127.0.0.1:7331
After=network.target

[Service]
Type=simple
ExecStart=/var/tmp/arachni_script.sh
TimeoutStartSec=0

[Install]
WantedBy=default.target

# vi /var/tmp/arachni_script.sh
#!/bin/bash
/home/centos/arachni-1.5.1-0.5.12/bin/arachni_rest_server --address=127.0.0.1
```

Solo nos faltará indicar a nuestra aplicación donde está funcionando la instancia de *Arachni* a través de variables de entorno:

```
# vi ~/.bashrc
export ARACHNI_URL="127.0.0.1"
export ARACHNI_PORT="7331"
```

Reiniciamos pm2 y aplicamos cambios:

```
pm2 update
pm2 stop all
pm2 kill
pm2 start npm --start
```

Los logs de pm2 nos informarán si el proceso ha iniciado correctamente:

```
0|npm     | > webwardrest@0.0.0 start /home/centos/WebWardRest
0|npm     | > node ./bin/init
0|npm     | Cluster Name: eomej2ox3pza71y
0|npm     | Platfrom detected: linux
0|npm     | Trying to detect kubernetes...
0|npm     | Not running in Kubernetes
0|npm     | Postgres DB running on <localhost:5432>...
0|npm     | Trying ARACHNI on 127.0.0.1:7331
0|npm     | ARACHNI: 127.0.0.1 7331
0|npm     | Arachni working
0|npm     | Cluster Name: eomettqgxetbhxfgi
0|npm     | 2018-06-14T15:11:22.776Z Worker is running on 31806
0|npm     | ARACHNI: 127.0.0.1 7331
```

ANEXO D: Módulos disponibles

ARACHNI REST

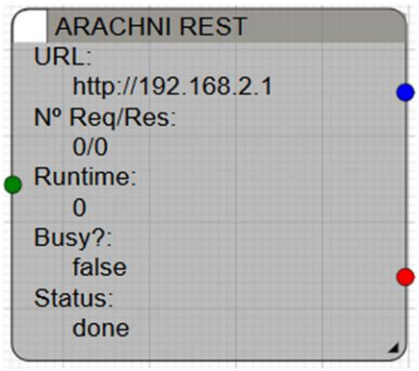


Figura 36 Nodo Arachni REST

Permite realizar análisis de seguridad, el escáner averigua si hay una vulnerabilidad al analizar la respuesta que obtiene de una página web ante cierta acción. Entre las propiedades que necesita están:

- **_STATUS**: informa sobre el estado del análisis.
- **_RUNTIME**: tiempo en ejecución.
- **_CHECKS**: Perfil para la realización del análisis.
- **_SCAN_URL**: Página web a la que hacer el análisis.
- **_SCAN_OPTS**: Opciones propias de *Arachni* para la realización del análisis.

Estos valores son mostrados dentro del nodo de la Figura 36.

CASTER

Cambia los nombres de parámetros (casteo de variables). El parámetro **_VALUES** contiene la información para dicho cambio en formato JSON. El siguiente ejemplo de la Figura 37 cambia el nombre del parámetro _SOMTHING a _MESSAGE.

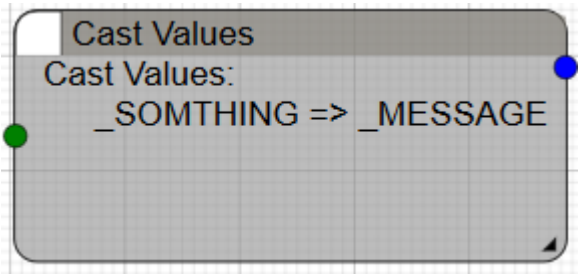


Figura 37 Nodo para el casteo de parámetros

COUNT

Permite retrasar la ejecución de un nodo utilizando para ello el retraso que existe entre ejecuciones sucesivas al tener que realizar varias iteraciones antes de finalizar (Figura 38).

- **_COUNT**: contiene el número de iteración actual.
- **_COUNT_TO**: número hasta el que debe contar.

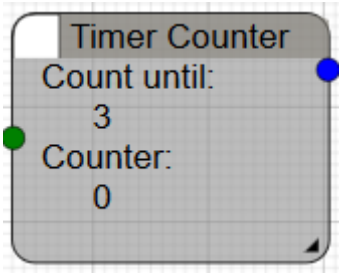


Figura 38 Nodo para retrasar ejecuciones de otros nodos

LOGER

Todos los parámetros que recibe son escritos en un archivo de texto, haciendo que este módulo muy útil para localizar errores (Figura 39).

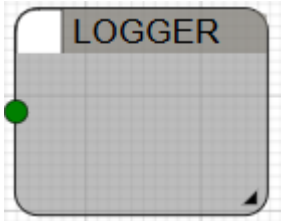


Figura 39 Nodo para imprimir parámetros en un archivo de texto

SCHEDULER

Ejecución de un nodo en unas horas concretas. La propiedad **_LAST_ACTIVATION** almacena la fecha de última ejecución para que no se ejecute de forma continua, la propiedad **_DAYS** nos permite decir qué días de la semana queremos que se active y con

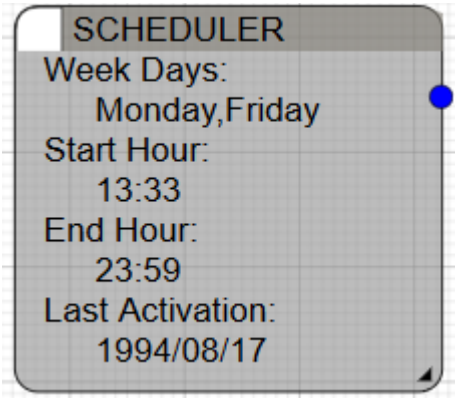


Figura 40 Nodo para la programación de ejecuciones

_START_HOUR y **_END_HOUR** controlaremos el intervalo de horas en las que se activará este nodo (Figura 40).

TELEGRAM

Para mandar información a un usuario o un grupo de usuarios utilizando Telegram. Todo parámetro que le llegue a este nodo será enviado a los usuarios convirtiéndolo previamente si fuera necesario a texto. Necesita el nombre del chat en el que se encuentra el bot y al que se quiere mandar la información, así como el token de acceso a la API de Telegram (Figura 41).

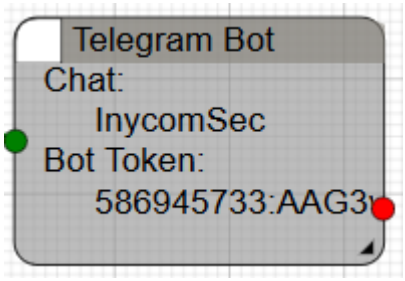


Figura 41 Nodo para notificar al usuario por medio de telegram

TEMPLATE

Posee dos submódulos, uno para plantillas de texto y otro para plantillas HTML, de forma que permite convertir cualquier objeto o lista de objetos en formato JSON en un texto. La única propiedad importante es _TEMPLATE que almacena la plantilla como se ve en el siguiente extracto del nodo de la Figura 42.

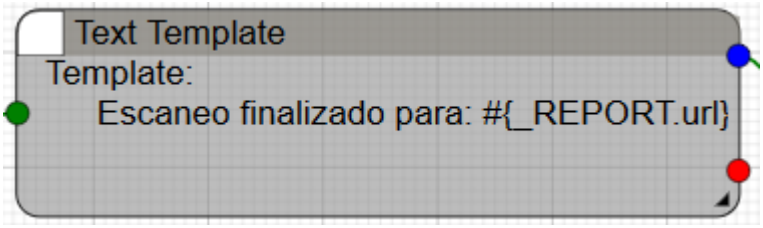


Figura 42 Nodo para rellenar plantillas de texto

```
Escaneo finalizado para: #{_REPORT.url}
Vulnerabilidades encontradas:
#[_REPORT.issues]
- #{$.name}
  Gravedad: #{$.severity}
  Encontrado por #{$.vector}
  Localizado en #{$.url}
#[_REPORT.issues]
```

WAPPALYZER

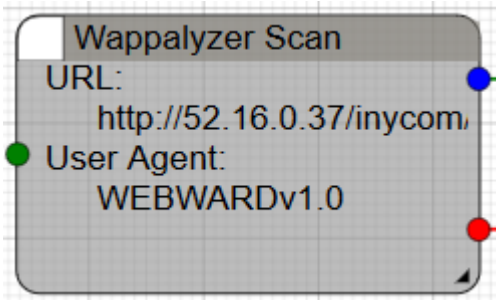


Figura 43 Nodo para realizar análisis utilizando Wappalyzer

Sencillo escáner de seguridad para identificar las tecnologías utilizadas por una aplicación web ya desplegada, sólo necesita la URL que analizar (Figura 43).

WEBHOOK

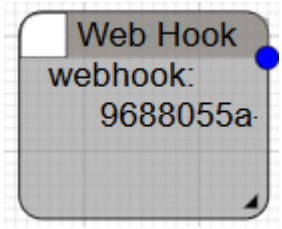


Figura 44 Nodo para la activación remota del pipeline

Permite que agentes externos u otras aplicaciones activen un nodo al ser llamado un webhook (Figura 44) en nuestra plataforma al hacer una llamada del tipo:

HTTP GET https://webward.com/api/webhook/9688055a-7275-11e8-8dc0-e7261ea74d02

ANEXO E: Aspecto visual de la plataforma

La plataforma de seguridad continua consta de dos partes: la parte visual o frontend y la parte que se ejecuta en el servidor o backend. Para acceder a toda la funcionalidad se utiliza la interfaz web cuyo punto de entrada es para iniciar sesión como se ve en la Figura 45.

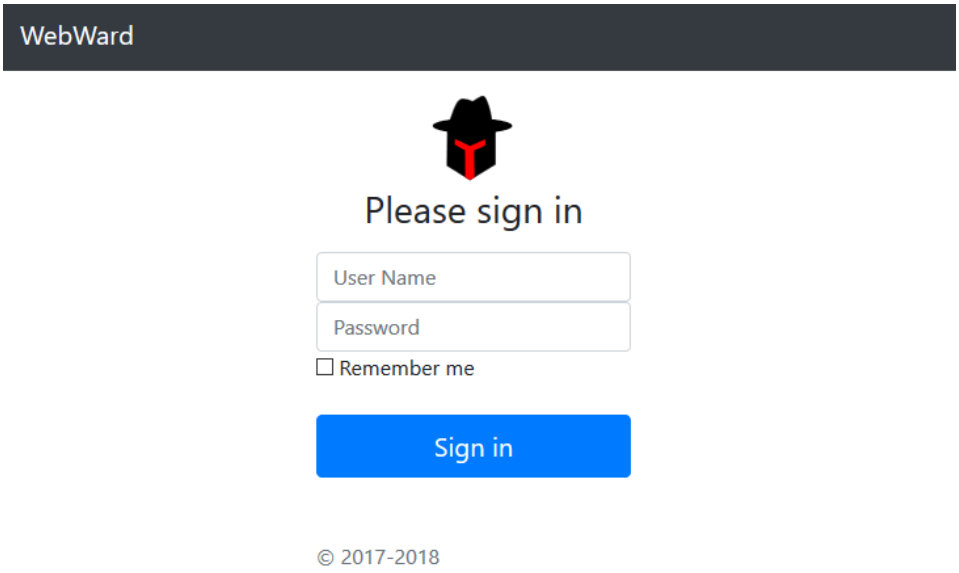


Figura 45 Punto de entrada con inicio de sesión

La página de inicio, Figura 46, contiene la lista de los últimos análisis de seguridad realizados y estadísticas relacionadas con ellos.

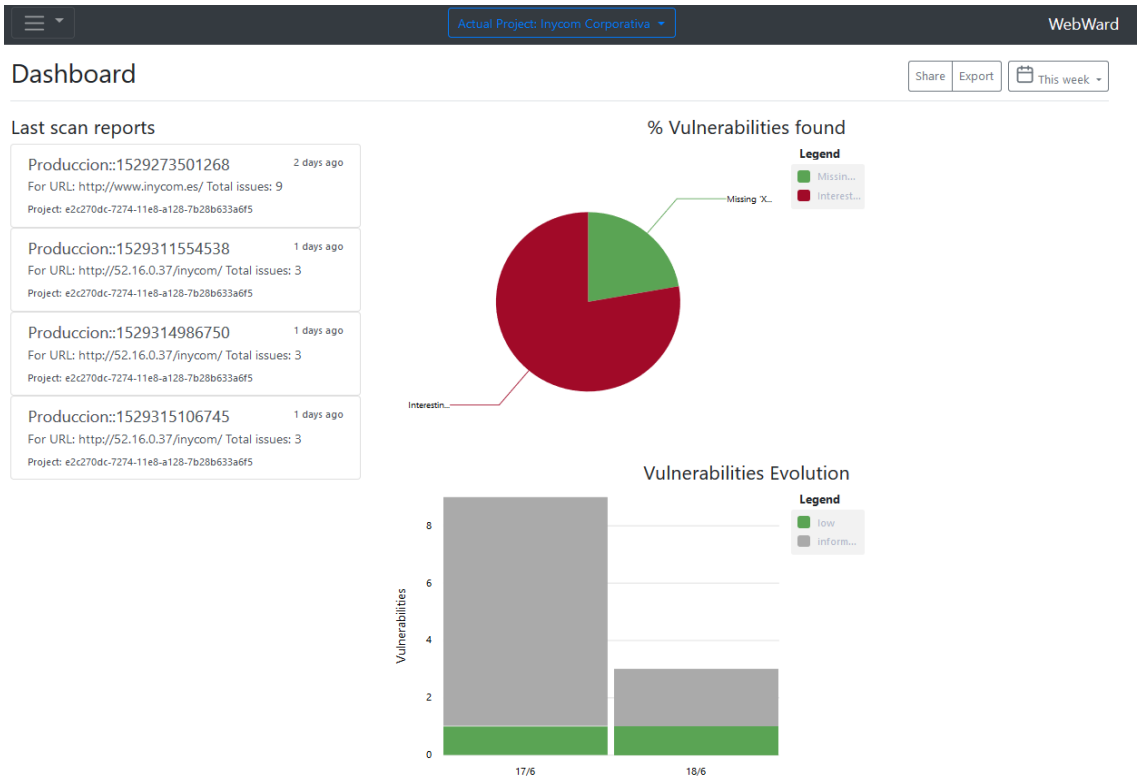


Figura 46 Página principal con un resumen de los últimos análisis

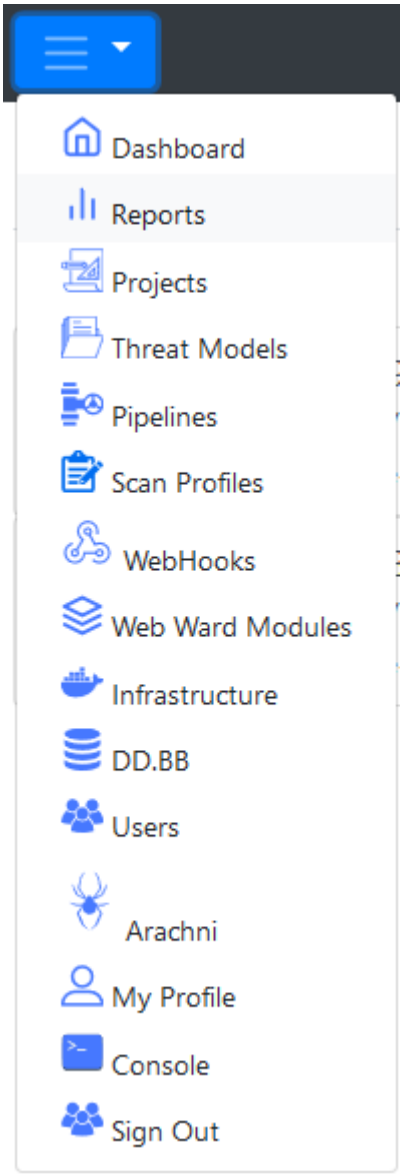


Figura 47 Navegación en la herramienta

Entre las diferentes pestañas por las que podemos navegar en función del tipo de cuenta y permisos que tengamos como usuario se encuentran como se puede ver en la Figura 47:

- **Dashboard**: página principal antes mostrada.
- **Reports**: Informes de seguridad para el Proyecto actual seleccionado.
- **Projects**: Lista de proyectos en los que estamos involucrados.
- **Threat Models**: gestión de modelos de amenazas.
- **Pipelines**: Editor de pipelines para realizar y programar análisis de seguridad.
- **Scan profiles**: Creación de perfiles para los escáneres de seguridad.
- **WebHook**: administración de activadores para los pipelines.
- **Web Ward Modules**: permite cargar dinámicamente módulos desde URL remotas en formato zip.
- **Infrastructure**: Si la plataforma tiene acceso a una API kubernetes podremos desplegar o eliminar objetos de kubernetes (solo para administradores).
- **DD.BB**: ejecución de comandos directamente en la base de datos (solo para administradores).
- **Users**: Gestión de usuarios de los que eres su jefe.
- **Arachni**: permite consultar directamente la API REST de Arachni y limpiar instancias si se han quedado en ejecución.
- **My profile**: muestra información sobre nuestro usuario y permite cambiar nuestra contraseña que por defecto es la misma que nuestro nombre de usuario.
- **Console**: ejecución de comandos en el servidor remoto (sólo para administradores).
- **Sign Out**: permite cerrar sesión.

Entre las páginas que más nos pueden interesar y que cabe destacar se encuentra la lista de informes de seguridad mostrada en la Figura 48, así como la lista de modelos de amenazas de la Figura 49.

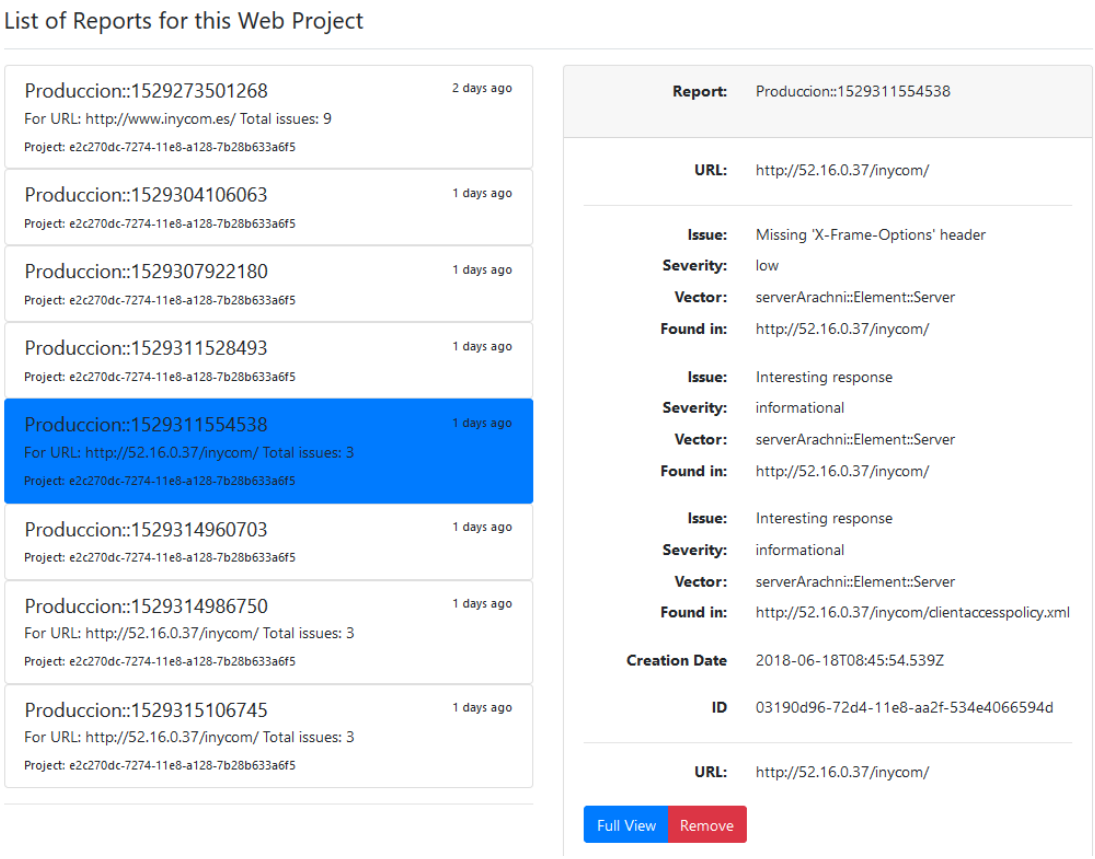


Figura 48 Listado de informes de seguridad

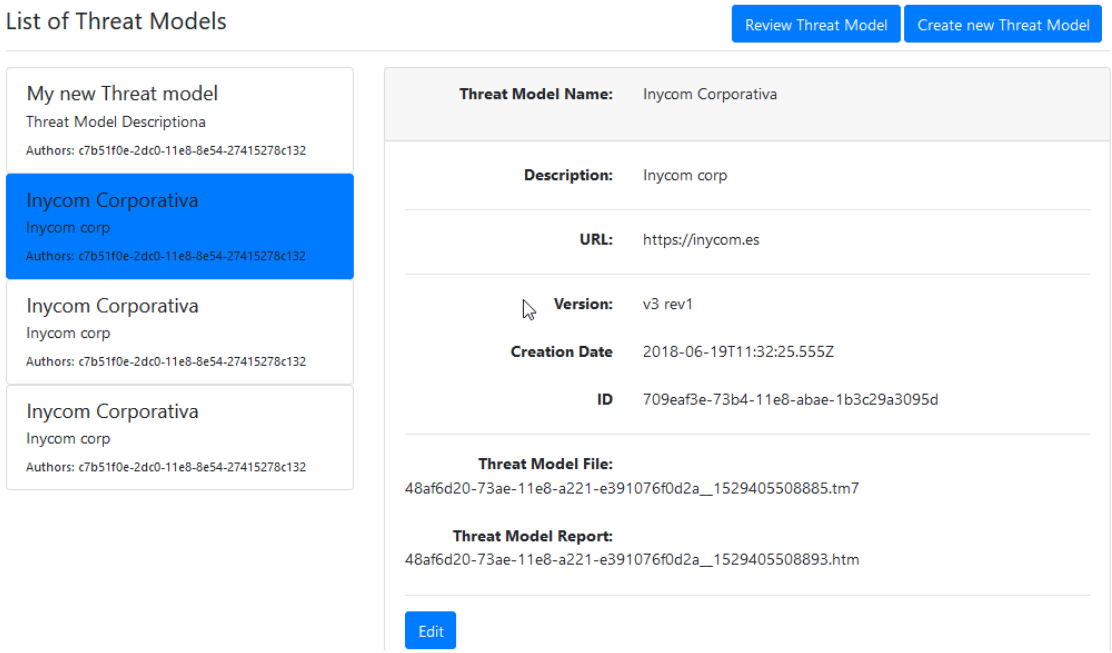


Figura 49 Listado de modelos de amenaza

ANEXO F: Tablas en la base de datos

```
CREATE TABLE IF NOT EXISTS users (
	id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
	name text NOT NULL UNIQUE,
	password text NOT NULL,
	email text NOT NULL UNIQUE,
	role int NOT NULL DEFAULT 0,
	manager UUID,
	create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
	last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
	status int NOT NULL DEFAULT 0
);



CREATE TABLE IF NOT EXISTS web_projects (
	id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
	name text NOT NULL UNIQUE,
	description text NOT NULL DEFAULT 'Insert Description',
	system jsonb,
	platform jsonb,
	project_manager UUID NOT NULL,
	create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
	last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
	status int NOT NULL DEFAULT 0
);



-- WebWard Properties. Only accesible by the Admins
CREATE TABLE IF NOT EXISTS ww_properties (
	name text PRIMARY KEY,
	value text NOT NULL
);



-- WebWard modules. Only accesible by the Admins
CREATE TABLE IF NOT EXISTS ww_modules (
	name text PRIMARY KEY,
	origin text NOT NULL,
	value json NOT NULL
);



CREATE TABLE IF NOT EXISTS scan_profile (
	id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
	name text NOT NULL UNIQUE,
	description text NOT NULL DEFAULT 'Insert Description',
	checks text NOT NULL DEFAULT '',
	owner UUID NOT NULL,
	create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
	last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);
```

```
CREATE TABLE IF NOT EXISTS kube_objects (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        name text NOT NULL UNIQUE,
        description text NOT NULL DEFAULT 'Insert Description',
        content jsonb NOT NULL,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);

CREATE TABLE IF NOT EXISTS infrastructure (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        name text NOT NULL UNIQUE,
        description text NOT NULL DEFAULT 'Insert Description',
        content jsonb NOT NULL,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);


CREATE TABLE IF NOT EXISTS web_hooks (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        name text NOT NULL DEFAULT '',
        node UUID NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS pipelines (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        web_project UUID NOT NULL,
        owner UUID NOT NULL,
        name text NOT NULL UNIQUE,
        description text NOT NULL DEFAULT 'Insert Description',
        status int NOT NULL DEFAULT 5,
        cluster_owner text NOT NULL,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        acquire_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT (current_timestamp -
interval '30 seconds'),
        start_date TIMESTAMP WITH TIME ZONE,
        end_date TIMESTAMP WITH TIME ZONE
);


CREATE TABLE IF NOT EXISTS pipeline_nodes (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        tag text NOT NULL,
        name text NOT NULL,
        type text NOT NULL DEFAULT 'ANY',
        pipe UUID NOT NULL,
        x real NOT NULL DEFAULT 1,
        y real NOT NULL DEFAULT 1,
        height real NOT NULL DEFAULT 300,
        width real NOT NULL DEFAULT 200,
        data jsonb,
        io_params jsonb,
        status int NOT NULL DEFAULT 0,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        start_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        end_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);
```

```
CREATE TABLE IF NOT EXISTS node_stored (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        tag text NOT NULL,
        name text NOT NULL,
        type text NOT NULL DEFAULT 'ANY',
        owner UUID NOT NULL,
        pipe text,
        x real NOT NULL DEFAULT 1,
        y real NOT NULL DEFAULT 1,
        height real NOT NULL DEFAULT 300,
        width real NOT NULL DEFAULT 200,
        data jsonb,
        io_params jsonb,
        status int NOT NULL DEFAULT 0,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        start_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        end_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);

CREATE TABLE IF NOT EXISTS scan_report (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        project UUID NOT NULL,
        name text NOT NULL DEFAULT 'Report',
        reporter text NOT NULL DEFAULT 'WebWard',
        data jsonb,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);

CREATE TABLE IF NOT EXISTS threat_model (
        id UUID PRIMARY KEY DEFAULT uuid_generate_v1mc(),
        project UUID NOT NULL,
        name text NOT NULL DEFAULT 'Threat Model',
        description text NOT NULL DEFAULT 'Description for Threat Model',
        applicationType text NOT NULL DEFAULT 'WebApp',
        owners UUID[],
        authors UUID[],
        stakeholders UUID[],
        url text NOT NULL,
        version integer NOT NULL DEFAULT 1,
        review integer NOT NULL DEFAULT 1,
        threatModelTemplate text,
        threatModelFile text,
        threatModelReport text,
        create_date TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp,
        last_update TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT current_timestamp
);
```