Universidad de Buenos Aires
Facultad de Ciencias Económicas

Universidad de Buenos Aires

Facultad de Ciencias Económicas, Ciencias Exactas y Naturales e Ingeniería

Maestría en Seguridad Informática

Tesis de Maestría

Titulo

Seguridad y Pruebas Funcionales: Un enfoque de automatización amalgamado, Basado en el modelo SECDEVOPS

Autor

Ing. Johan Javier Pizarro Martínez

Tutor

Dr. Juan Pedro Hecht.

Año de Presentación

2019

Cohorte

2015

Declaración Jurada de origen de los contenidos

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien, pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

_____

Firma

_____

Aclaración

_____

DNI

Resumen

La competencia constante que tienen las compañías ha aumentado la creación exponencial de aplicaciones de software tanto web como Mobile. Esto ha ocasionado que dentro de los equipos de desarrollo y calidad tengan en cuenta más lo funcional que la seguridad, es por ello que un número sorprendente de sitios corporativos sean vulnerables a piratas informáticos. Las consecuencias de una infracción de seguridad que afrontan las compañías son críticas. Entre ellas se pueden considerar pérdida de ingresos, responsabilidades jurídicas y ausencia de confianza por parte de los clientes.

Como solución a esta problemática las organizaciones deberían implementar un plan de acción de seguridad en sus productos y llevar a cabo auditorías durante todo el ciclo de vida del desarrollo de software SDLC; con el propósito de identificar de manera temprana el origen de los defectos. Asimismo, la comunicación debería ser una de las principales directrices, dado que en el momento que sean encontrados los principales riesgos de seguridad estos puedan ser comunicados en todo el equipo de desarrollo para que puedan ser mitigados.

El conocimiento en seguridad y el entrenamiento constante del personal de sistemas en las organizaciones es de vital importancia para que el desarrollo de software sea más seguro. Además, si todo el equipo conoce cómo está constituida la infraestructura en que operan las aplicaciones que desarrollan y los protocolos de redes que se utilizan, aumentaría las probabilidades de reducir el número de defectos de seguridad. Los riesgos de seguridad identificados mediante las regresiones de QA (Quality Assurance) al final ayudarían a medir y analizar los resultados. Esto contribuiría a una reducción de las responsabilidades en todo el equipo de desarrollo y que los miembros de QA estén más involucrados con la seguridad en los sistemas.

Palabras Claves: SDLC, Protocolos, OWASP, QA.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

AGRADECIMIENTOS

El desarrollo de esta tesis requirió bastante esfuerzo y muchísima dedicación, es por eso que agradezco inmensamente a mis padres y hermana, Luz Marina, Ricardo Antonio y Yenni Alexandra, por darme el valor y la confianza de seguir adelante en este camino de crecimiento profesional y personal, ustedes son mi motor y empuje para seguir adelante en esta vida, sin ustedes nada de esto hubiera sido posible.

Agradezco profundamente, con mucho aprecio y amor a mi pareja Cintia Grosiean, por acompañarme durante todo este proceso, por darme siempre ese aliento cuando más lo necesitaba, brindarme una excelente asesoría y apoyo incondicional durante el desarrollo de todos mis estudios de Maestría, esto también es posible gracias a ella.

Agradezco a mis buenos amigos en Argentina, Verónica Quispe, Solange Lescano, Almendra Quispe, Noemi Saavedra, Luis Fernando Huertas, Jovanna Perez Silva, Jhonatan Fernandez y Andres Pantoja, por brindarme todos sus buenos consejos, solidaridad, hospitalidad, paciencia, apoyo y estar en esos momentos en que más los necesitaba para salir adelante en el país y poder tener fuerzas para sacar la Maestría adelante

A mi Director de Tesis Dr. Juan Pedro Hetch, por aceptar ser mi tutor, estar pendiente de todas mis consultas, por confiar en mi trabajo, gracias por su orientación, su paciencia y motivación durante todo el desarrollo de esta tesis.

1. Fundamentación, justificación

1.1 Antecedentes

El desarrollo de software anteriormente era realizado con base al modelo waterfall, en su definición al castellano el modelo en cascada. Tiene como enfoque metodológico el ordenar rigurosamente las etapas del proceso de desarrollo de software. Esto significaba en su momento que el inicio de cada etapa debía esperar la finalización de la etapa anterior.

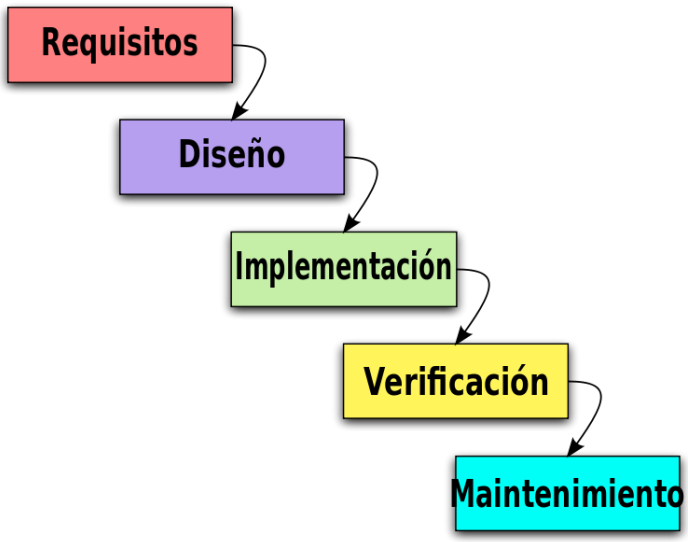


Figura 1 Modelo Waterfall

[1]

Las primeras fases del modelo waterfall contemplaban primero todo un análisis y documentación de los requisitos de los usuarios finales, a continuación se encontraba la fase de diseño que consistía en descomponer y organizar el sistema en elementos que pudieran elaborarse por separado. Una vez conseguido esto el resultado era un documento de diseño de software que contenía la estructura relacional global del sistema y las especificaciones

que debía hacer cada una de sus partes. Terminada esta fase seguía la Implementación que implicaba el desarrollo de algoritmos basados en algún leguaje de programación tomando como referencia los documentos surgidos en las fases anteriores. Como penúltima fase estaba la verificación donde eran probados los elementos del sistema codificados con el propósito de verificar que funcionaran correctamente de acuerdo a los requerimientos de los usuarios finales. Como última fase estaba la de mantenimiento que era considerada una de las más críticas ya que destinaba la mayor cantidad de recursos debido a que muchas veces no cumplía con las expectativas del usuario final.

1.2 Estado actual del tema

El modelo waterfall con el pasar de los años fue perdiendo popularidad por su alto coste en recurso y tiempo, debido a que en la vida real el desarrollo de software no podía seguir rigurosamente las etapas propuestas por este modelo y cada vez eran más los clientes que necesitaban ver los productos operando en sus entornos productivos y adicionando a ello también los cambios de requerimientos solicitados una vez el software había sido terminado.

El advenimiento de las metodologías ágiles fue poco a poco desplazando el modelo tradicional del desarrollo de software waterfall y actualmente cada vez más son las empresas que incorporan en su SDLC[1] estas nuevas tendencias. Esto se debe a que estas permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta de amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno. Además, esta metodología mejora la satisfacción del cliente, dado que este puede involucrarse y comprometerse a lo largo del SDLC.

[1] **SDLC** Es la abreviatura que significa Ciclo de vida de desarrollos de sistemas.

Uno de los principales focos de las metodologías ágiles es tener al cliente más involucrado. Esto significa que los equipos de desarrollo deben hacer entregas (releases[2]) frecuentemente, lo que involucra tener un sistema de despliegues continuo (continuos deployment-CI) o entrega continua (continuos delivery-CD). Por este motivo muchas empresas están recurriendo al uso de la metodología DevOps.

DevOps es una metodología para creación de software. Que ayuda a la integración y comunicación de las áreas de Desarrollo, Quality Assurance (QA) y Operaciones. Esta metodología permite fabricar software más rápidamente con mayor calidad, menor coste y una altísima frecuencia de releases.

1.3 Planteo del problema

Según las estadísticas proporcionales que realiza la empresa Building Security In Maturity Model [2]a las principales y grandes empresas en el mundo, arrojo que en estas compañías tienen una cifra en promedio de 1 persona de seguridad por cada 245 de desarrolladores y probadores **(1:245).**

La empresa IBM Security realiza anualmente un estudio estadístico sobre los costes de los incidentes relacionados con las brechas de seguridad. Para el año 2017 dio a conocer que los ataques mal intencionados o delictivos continúan siendo la causa principal de las brechas de seguridad en los datos. El estudio fue realizado a 63 corporaciones y según las estadísticas el 52% de los incidentes consistieron en un ataque mal intencionado o delictivo, mientras que el 24% se debieron a negligencia de los empleados y otro 24% fueron causados por errores del sistemas, incluidos errores de TI y de los procesos empresariales. [3]

[2] **Release**, Es el proceso de entregas de nuevas actualizaciones o funcionalidades de un producto de software.

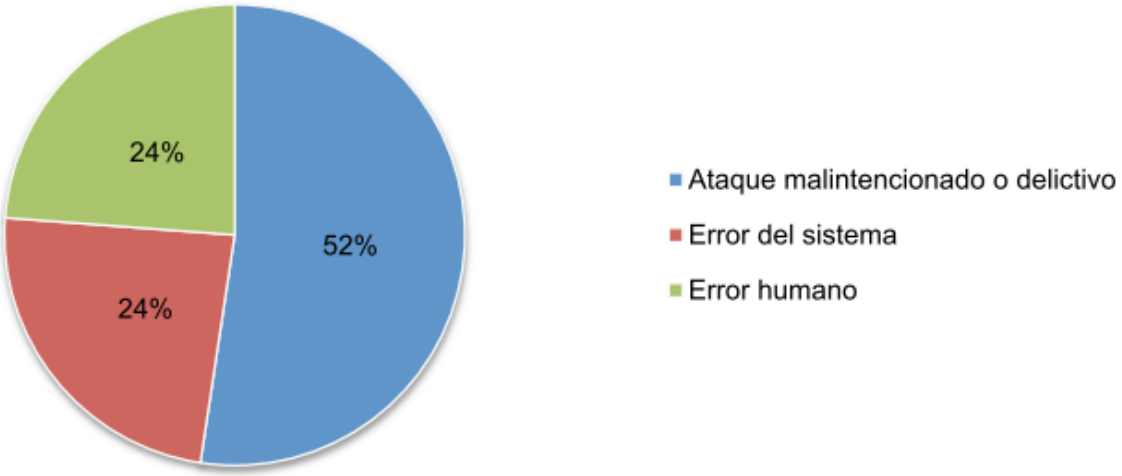


Figura 2 Diagrama de Porcentaje Errores Del Sistema

[3]

Según los resultados que las empresas dieron a conocer, los costes per cápita generados por estos incidentes ascienden a 244 dólares en los ataques malintencionados, 209 dólares los errores de sistema y 200 dólares los errores humanos.

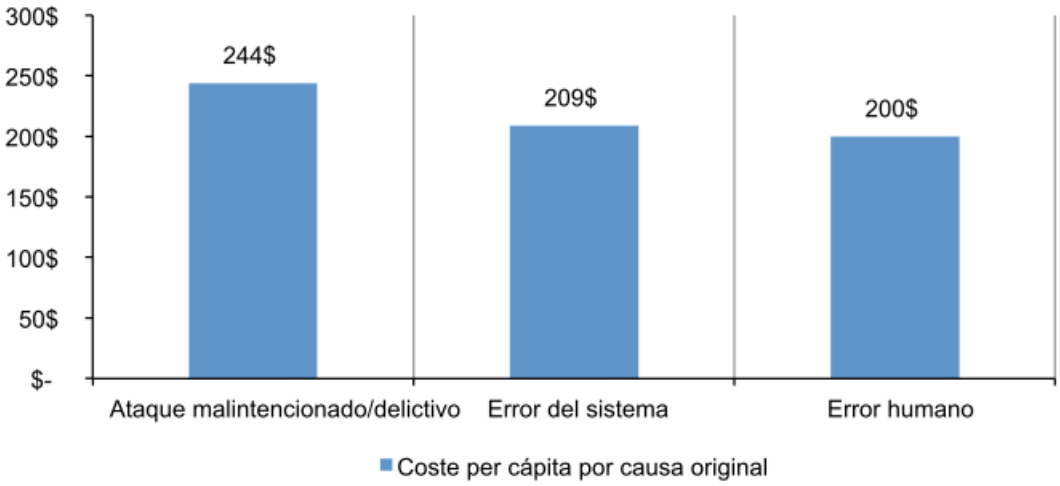


Figura 3 Coste per Cápita por Causa Original

[3]

La causa de estas estadísticas se debe a que las empresas cada vez necesitan sacar productos al mercado con mayor agilidad y es una carrera por mejorar la competitividad. Por este motivo dentro de los equipos de desarrollo

de software las metodologías ágiles son la solución a esta demanda, pero la creación de software ágil trae como consecuencia la generación de muchas fallas y defectos que tienen como probabilidad la aparición de vulnerabilidades de seguridad. Esto es debido a que la mala calidad del código conduce a un comportamiento impredecible y desde la perspectiva de un usuario, a menudo se manifiesta como una mala usabilidad, dando esto la oportunidad de que atacantes puedan encontrar estas vulnerabilidades y en su defecto afectar la integridad, autenticidad y disponibilidad del sistema.

Este afán que tienen constantemente las compañías de sacar sus productos al mercado hace que cada vez más la prioridad de involucrar la Seguridad dentro del ciclo de vida del desarrollo de software (SecSDLC) pierda interés, porque ven en la seguridad una tarea adicional que pone trabas y afecta negativamente el cronograma que los equipos de desarrollo tienen estipulados para las entregas. Esto se debe a que los equipos no están teniendo en cuenta que la seguridad es un problema de control de calidad y tampoco que ambas están relacionas con la eliminación del riesgo. [3]

Los estudios recientes que ha realizado la empresa IBM Security con respecto al tiempo necesario que necesitan las empresas para identificar y contener una brecha de seguridad en los datos y como este repercute en los costes, dieron como resultado que cuanto menos tarde una compañía en identificar y contener una brecha de seguridad, menor será el coste que tengan afrontar. Según la estadística las empresas que no tienen un plan de respuesta a incidentes (MTTI) el tiempo medio de detección de estos pueden llegar a más de los 200 días. Diferente es el panorama para aquellas empresas que sí lo tienen (MTTC), donde el tiempo medio de detección y contención son 55 días, es decir menos de dos meses. [3]

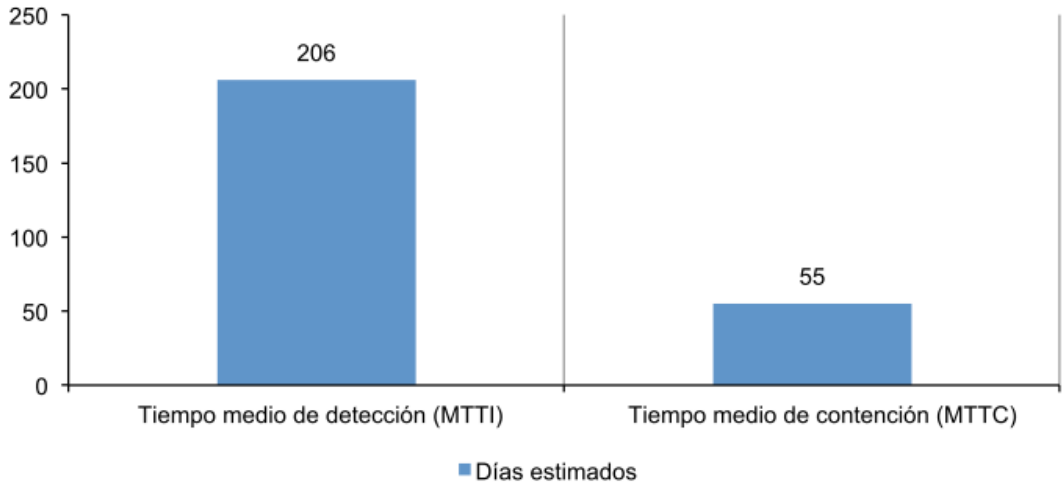


Figura 4 Tiempo Medio de Detección Vs Tiempo Medio Contención

[3]

Para las empresas que superan los 100 días, el coste medio de detección de una brecha de seguridad es de 8,70 millones de dólares, sin embargo para las que no superan los 100 días el coste medio es de unos 5,99 millones de dólares.

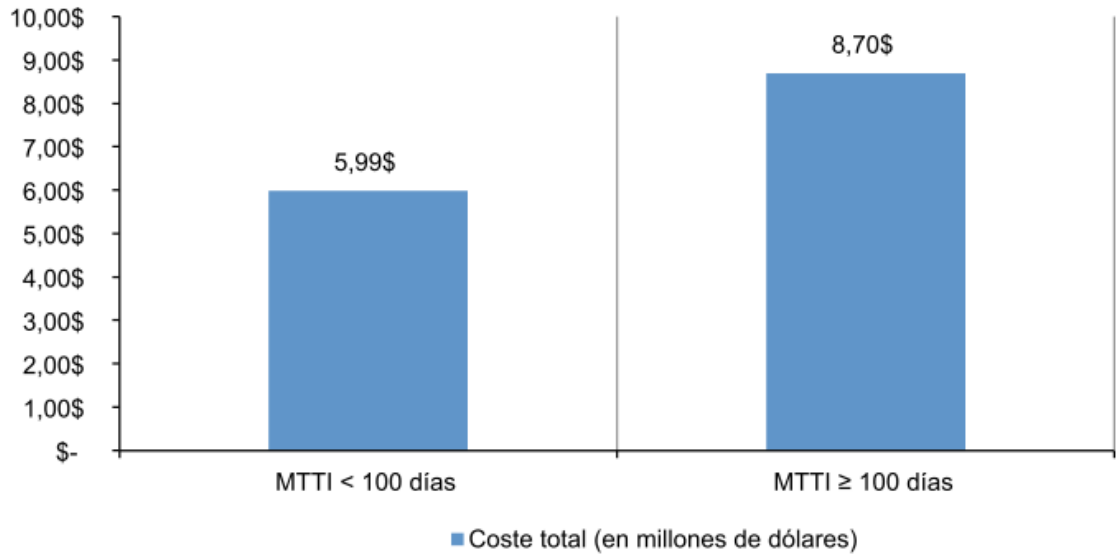


Figura 5 Costos Tiempo Medio de Detección Vs Tiempo Medio de Contención *[3]*

Para evitar pagar estos altos costes las empresas tienen la necesidad de incentivar la comunicación e integración de las áreas y para ello han optado por la metodología DevOps, pero han ignorado el SecDevOps que es el proceso de integración de buenas prácticas y metodologías de desarrollo seguro.

Cabe mencionar que actualmente las personas encargadas de la seguridad son pocas en comparación con las cantidades que hay en las demás áreas de IT. Por este motivo la seguridad no puede estar en todos los procesos del desarrollo de software y tampoco pueda seguir el ritmo rápido que exige las metodologías agiles.

Otros de los principales aspectos que ponen trabas a la seguridad son las formas de trabajo tradicionales que tienen los equipos de desarrollo, con respecto al versionado del software que se tiene en cada uno de los branch[3]. El siguiente ejemplo puede ilustrar la situación: dado un equipo de desarrollo que está codificando un nuevo producto, llegado al fin del reléase o sprint[4] 1 debe subir los cambios a su branch de desarrollo y, posteriormente, debe mergear los cambios a los branch de los equipos de seguridad y QA. Luego comienzan el realese 2 que involucra nuevos features. En esta etapa surge el siguiente interrogante ¿qué pasa con el tiempo que tiene seguridad para realizar el testing seguro de las aplicaciones? El tiempo que dedica Seguridad para probar las aplicaciones es el mismo que, en paralelo, utiliza Desarrollo para avanzar en el segundo reléase. El problema surge porque el equipo de seguridad ha terminado de probar la seguridad de la aplicación en su reléase 1 y cuando informa a Desarrollo sobre las incidencias encontradas, este equipo ya está en proceso el release2. Esto significa que las incidencias encontradas por el equipo de seguridad se encuentran desactualizadas con respecto a la versión actual del software, es así que desarrollo hará caso

[3] **Branch,** Se refiere a una línea de código fuente o rama, utilizada en el control de versiones de un software, por lo general en un equipo de desarrollo se crean diferentes ramas para alojar el código branch Dev, QA y Preproducción.

[4] **Sprint,** El corazón de Scrum es un Sprint, es un intervalo prefijado durante el cual se crea un incremento de producto "Hecho o Terminado" utilizable, potencialmente entregable. A lo largo del desarrollo hay Sprints consecutivos de duración constante.

omiso de estas incidencias y el equipo de seguridad perdió su tiempo en esta tarea.

La siguiente imagen ilustra la gran problemática de hacer uso de las actuales metodologías ágiles.

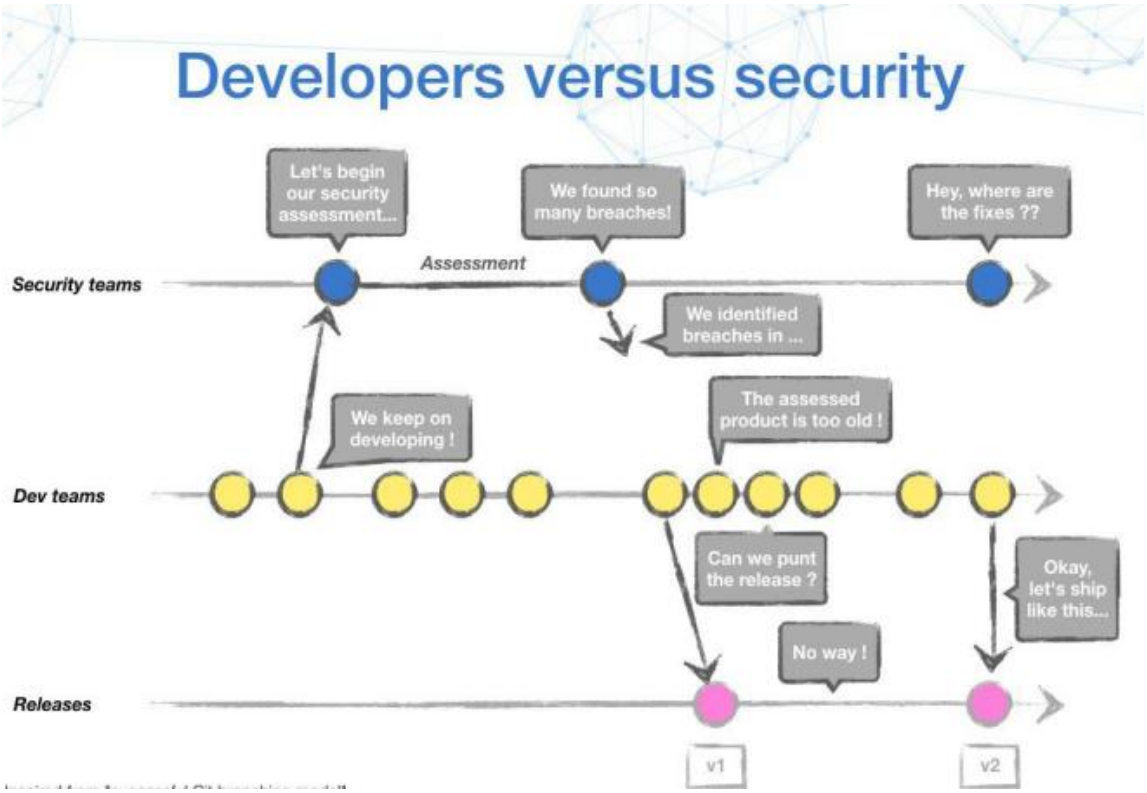


Figura 6 Branchs por Equipos *[4]*

1.4 Aportes teóricos y prácticos al campo

Según Wisseman, Stan [5]–"Los equipos de QA y seguridad definen requisitos funcionales y no funcionales que los desarrolladores deben cumplir. Estos requisitos funcionales y no funcionales son los cimientos para construir equipos de desarrollo con mentalidad de seguridad, y cuando los equipos de QA trabajan mano a mano con el personal de seguridad desde el principio, puede ser bastantes poderosos. Las pruebas de seguridad automatizadas deben ser vistas como un componente central en este proceso, ya que la

gerencia de desarrollo deberá reconocer que los defectos de calidad son puntos de entrada a las vulnerabilidades".

El presente trabajo de maestría tiene como finalidad realizar una guía técnica donde se pueda describir y detallar las tecnologías recomendadas por la OWASP Testing Guide. Estas permiten armar la infraestructura de un ambiente de integración continua que lleva a cabo la ejecución de pruebas funcionales y de seguridad de forma automatizada. Este proceso tiene como propósito obtener resultados y métricas que ayuden a detectar riesgos de seguridad tan pronto como sea posible durante el SDLC. De esta manera, los equipos de QA que no son expertos en temas de seguridad, generen un enfoque de automatización amalgamado basado en el modelo SecDevOps.

2.0 Alcance

El trabajo propuesto se enfocará en implementar una infraestructura de pruebas de seguridad en aplicativos web que permita aprovechar las pruebas de automatización funcional y colabore con los equipos de QA, Seguridad y Desarrollo en identificar incidencias de seguridad de manera temprana durante el SDLC. Los programas que se usarán para esta implementación serán de código abierto Open Source.

3.0 Objetivos generales

- Ofrecer una guía técnica y conceptual a los equipos de QA de cómo implementar una infraestructura para un ambiente de integración continúa basado en el modelo SecDevOps que ayude a llevar a cabo la ejecución de pruebas de seguridad automatizadas. Con el propósito de obtener resultados y métricas durante el ciclo de vida del desarrollo

de software que permitan generar una seguridad de la información proactiva.

4.0 Objetivos específicos

- Dar a conocer a los equipos de QA la importancia de actualizar el modelo DevOps a SecDevOps dentro de los procesos tradicionales de testing de software.

- Generar un mindset en los equipos de QA de la importancia de ayudar a la seguridad de la información dentro de los procesos del testing con el fin de generar una seguridad proactiva.

- Describir las tecnologías que permiten construir la infraestructura para un ambiente de integración continua que lleva a cabo la ejecución pruebas de seguridad automatizada basada en el modelo SecDevOps.

- Dar a conocer los detalles técnicos de cómo realizar las configuraciones de las tecnologías que permiten construir la infraestructura para un ambiente CI/CD.

- Realizar pruebas de conceptos donde se implementen una integración continua que permita realizar pruebas de seguridad automatizada, generación de reportes y creación de incidentes.

5.0 Metodología

En el desarrollo de este proyecto, se usará material bibliográfico relacionado con todos los aspectos referentes a las pruebas de seguridad en

servicios web expuestas por OWASP (Proyecto Abierto de Seguridad en Aplicaciones Web).

6.0 Plan de actividades

El plan de actividades se llevará a cabo de la siguiente manera:

Actividades	**Duración (Semanas)**
Investigación de Herramientas.	
Selección de aplicaciones recomendadas por Owasp que permitan llevar a cabo un pipeline basado en el enfoque SecDevOps.	1
Estado del arte de las aplicaciones.	1
Configuración de Herramientas	
Explicación del desarrollo guiado por comportamiento (BDD).	1
Pruebas de concepto: Escenarios de Seguridad escritos en BDD	1
Ejecución de pruebas funcionales y seguridad Automatizados.	1
Explicación de configuración de generación de reportes.	1
Explicación de configuración de integración de generación de incidencias.	1
Explicación de configuración de un Ambiente de Integración Continua.	1
Pruebas de Concepto: Armado de un ambiente de Integración continua para Pruebas de Seguridad.	
Conclusiones	1

Tabla 1 Cronograma de Actividades Tesis Maestría

7.0 Capítulo 1: DevOps una nueva tendencia en la creación de aplicaciones

7.1 Introducción

Es común en estos días escuchar con mucha frecuencia en las empresas el terminó Devops. En el pasado, los equipos de desarrollo no tenían la necesidad de realizar constantes cambios debido a que las aplicaciones eran sistemas aislados que tenían como función guardar grandes cantidades de información en sus bases de datos. Además, el personal encargado de utilizarlas no se quejaba respecto a su funcionalidad ni requería grandes cambios. Sin embargo, con advenimiento de la comunicación Mobile y la maduración de las aplicaciones web, los sistemas tradicionales fueron dándole paso a los actuales sistemas que involucran más al cliente con el negocio. Así estas aplicaciones deben contar con un alto nivel de rendimiento, debiendo ser fáciles de usar y poder cambiar constantemente de acuerdo a las necesidades del mercado. Para ello los equipos de desarrollo tienen que estar entregando continuamente y rápida versiones nuevas del producto, en pocas palabras esto es el "Enfoque DevOps".

7.2 ¿Por qué usar DevOps?

DevOps es un enfoque que promueve la colaboración entre líneas de negocio, desarrollo y operaciones de TI [6]. Es una funcionalidad empresarial que habilita la entrega continua, el despliegue continuo y la supervisión continua de aplicaciones. Reduce el tiempo necesario para tratar el feedback de los clientes. El desarrollo y las operaciones, e incluso las pruebas, antes se organizaban en silos, DevOps las reúne para mejorar la agilidad de modo que permite entregar código más rápido y alinearlo de manera más efectiva con los objetivos comerciales.

DevOps deriva de las metodologías ágiles actuales, pero podría decirse que este enfoque es en una metodología ágil con esteroides, lo que significa que es una la evolución de lo que hoy sabemos de las metodologías ágiles como Scrum o XP.

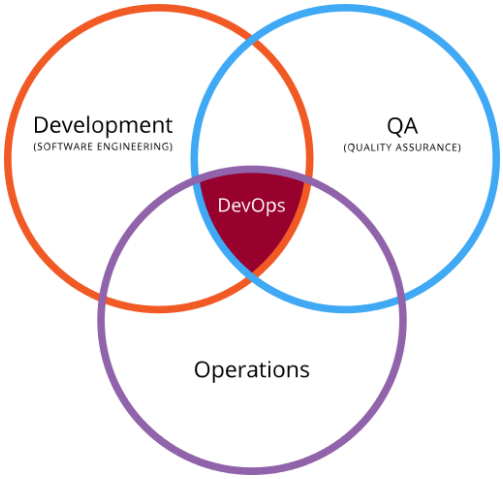


Figura 7 Diagrama DevOps

[7]

Los proyectos que usan el enfoque Devops tienen la ventaja de mejorar la comunicación y colaboración de las áreas de desarrollo, QA y Soporte de operaciones. Por otra parte, facilita la entrega continua de software gracias a la realización de pruebas en colaboración y la supervisión continua de los entornos de desarrollo, integración y transferencia. Además, corrige desalineaciones de objetivos de negocio, creando vínculos más cercanos entre desarrolladores y operaciones, también gracias a las herramientas facilitan la gestión de releases, el suministro de la infraestructura, la orquestación, la supervisión, la inclusión en contenedores, la virtualización y la automatización. [8]

8.0 Capítulo 2: SecDevOps el presente & futuro del aseguramiento de la calidad y seguridad de las aplicaciones

8.1 Introducción

Por lo general algunas compañías tienden a pensar que las pruebas de seguridad de aplicaciones es un requerimiento opcional dentro del SDLC, debido a que la seguridad es considerada como un costo que no tiene sentido pagarlo o invertir tiempo y recursos en etapas tempranas del desarrollo. Esto se debe a que las prioridades en los proyectos en gran parte son realizar entregas al cliente lo más frecuente posible y ven en la seguridad un obstáculo para poder llegar a sus objetivos de entrega. Es así que la implementación de la seguridad en etapas iniciales del proyecto de software es casi nula.

La necesidad de generar entregas frecuentes y de mejorar la comunicación y colaboración de trabajo entre los diferentes equipos de QA, Desarrollo y Operaciones hacen que el enfoque DevOps sea uno de los más utilizados hoy en día. Sin embargo, este enfoque por sí solo no es suficiente dado que cada vez que se realiza una entrega del software al cliente, el código cambia de manera frecuente. Esto puede ocasionar que las aplicaciones sean más propensas a tener muchos errores funcionales que derivan a errores de seguridad y esto es algo que todos en un equipo de desarrollo de software deben comprender que los errores funcionales van fuertemente agarrados de la mano de las posibles vulnerabilidades de seguridad, esto quiere decir que para poder garantizar la seguridad al igual que se garantiza la calidad de una aplicación, esta debe ser parte integral del proceso de Desarrollo, QA e Implementación de las aplicaciones, es por este motivo que nace el movimiento SecDevOps.

SecDevOps “es un conjunto de buenas prácticas diseñadas para ayudar a las organizaciones a implantar la seguridad en el corazón de sus procesos de desarrollo e implementación basados en el enfoque DevOps”. [9]

Es así como SecDevOps le da un plus al enfoque DevOps, adicionando durante todas las etapas del desarrollo de software la seguridad.

8.2 Importancia de tener SecDevOps dentro del SDLC

Los modelos de desarrollo de software en cascada obligaban a participar en cada una de las etapas al personal experto en seguridad. A diferencia de este modelo tradicional, el conjunto de buenas prácticas de SecDevOps, es aplicado en todo el desarrollo e integración continua del software. Estas prácticas ayudan a garantizar la seguridad durante todo el SDLC, pero este objetivo no se alcanza de la noche a la mañana. Muchas organizaciones tienen pocos ingenieros con perfiles de expertos en seguridad o directamente, no los tienen. Por este motivo los equipos de QA y Desarrollo deben ser capacitados con la finalidad de comprender aspectos técnicos de seguridad que permitan identificar vulnerabilidades tanto en lo técnico como lo funcional de las aplicaciones. [10]

Una de las ventajas de SecDevOps es que permite involucrar y colaborar entre los perfiles de desarrollo, QA y Operaciones en torno a la seguridad. Esto significa que si en una compañía existiera personal con perfil en seguridad de la información no garantizaría la seguridad durante todo SDLC. Los analistas de seguridad no siempre pueden hacer revisiones de código, o tener el conocimiento completo de una aplicación a nivel funcional y pruebas como lo tienen los analistas de QA o revisar todos los cambios que los analistas con perfil de Operaciones pueden hacer en la configuración del sistema.

Vale la pena mencionar que en el informe de estado DevOps de Puppet de 2018, la compañía señala que los equipos de alto rendimiento de DevOps “Dedican un 50% menos de tiempo a solucionar problemas de seguridad que los de bajo rendimiento” [11], por lo tanto una compañía que adopte el enfoque

SecDevOps va tener más productividad y con un riesgo mínimo de sufrir fallas en su seguridad.

8.3 AppSec y QA testing como un equipo integrado

El enfoque DevOps desdibuja cada vez más el límites entre desarrollo, prueba y operaciones, dado que el objetivo principal es generar una colaboración entre todos los equipos, por este motivo pensar que los analista QA solo deben estar involucrados en las actividades de calidad funcional y no participar en otras actividades como las de seguridad va contra las buenas prácticas de SecDevOps. Dentro de las organizaciones se cometen a diario el error de pensar que el personal de QA no tiene el conocimiento técnico para entender y reportar los errores de seguridad y por ello solo el personal de seguridad debe estar involucrado únicamente en estas actividades. De ahí que el control de calidad sea incompatible con las pruebas de seguridad básicas.

En esencia el trabajo realizado por equipos de seguridad es el mismo trabajo que hacen los equipos de QA. La diferencia se presenta en el tipo de pruebas que realizan y por lo tanto, la seguridad de las aplicaciones puede depender del equipo de QA y no sería necesario requerir del 100% del trabajo de un especialista en seguridad.

8.4 SecDevOps dentro del SDLC

Vale la pena mencionar la evolución que han tenido las metodologías de desarrollo, con respecto a la metodología tradicional como la watterfal desde el punto de vista de los clientes, el producto nunca llegaba de acuerdo a lo que esperaban. Siempre faltaba algo y esto generaba bastante frustración

en los equipos de desarrollo. Sin embargo desde el punto de vista de seguridad esta metodología era bastante organizada, y cómoda durante las primeras fases, tales como Análisis y Diseño, se definían los requisitos de seguridad, esto implicaba saber cómo iba a funcionar la aplicación, qué tipos de datos iba a utilizar, dónde iba estar instalada y de acuerdo a ello eran implementados los mecanismos de seguridad necesarios, luego se realizaba un seguimiento en las fases de implementación y pruebas con el fin de garantizar que los mecanismos se estaban llevando a cabo, en forma posterior, en las fases de despliegue, se verificaba que no hubiesen vulnerabilidades graves mediante pruebas de intrusión por medio de herramientas especiales y pruebas manuales, finalmente en la fase de mantenimiento si eran solicitados por alguna ley o normativa se realizaban pruebas periódicas una vez al año [10].

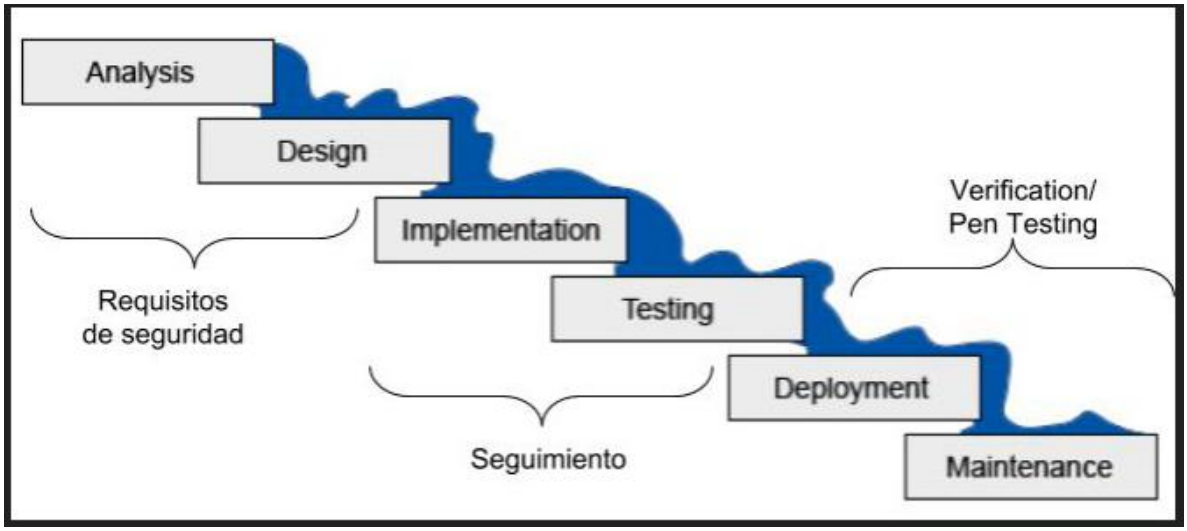


Figura 8 Modelo Watterfall SDLC *[4]*

Con la llegada de las metodologías ágiles las fases de desarrollo cambian y hacen que el diseño de pruebas de seguridad no sea tan claro, debido a que el producto se define por etapas y los equipos de seguridad nunca llegan a tener una visión real del mismo. Las pruebas de seguridad son realizadas en cada release y las vulnerabilidades encontradas se reportan cuando Desarrollo se encuentra trabajando en un reléase posterior.

SecDevOps entonces llega como una solución a las problemáticas surgidas en el modelo Secure SDLC tradicional, modelo con muchas falencias y que se ha convertido en un obstáculo para que los equipos puedan hacer entregas continuas.

A continuación, se dará a conocer una tabla comparativa del enfoque SecDevOps y Secure SDLC.

Secure SDLC	SecDevOps
División	Colaboración
Trabajo Manual	Automatización
Lento	Rápido
Estructurado	Ágil

Tabla 2 SSDLC Vs SecDevOps

[12]

Como se pude apreciar en la tabla Secure SDLC, los procesos son divididos, lo que implica también que cada perfil trabaje en islas separadas con sus propios procesos, por otro lado SecDevOps tiene un enfoque diferente y, como ya se ha mencionado, todos los perfiles trabajan bajo los mismos procesos y de manera colaborativa. El trabajo en Secure SDLC es un trabajo manual, la mayoría de actividades en este modelo son realizadas de esta manera, por el contrario SecDevOps prioriza mucho en el tema de la Automatización, “Automatizar Todo” es la frase principal de este enfoque, lo que permite dar una gran agilidad a todas las actividades, de esta manera un equipo de desarrollo puede entregar más rápido el software por partes, tarea que era muy complicada con el modelo Secure SDLC dado que sus procesos son lentos y estructurados.

8.5 Integración, despliegues y entrega continua

Tener dentro de una organización implementada las buenas prácticas del enfoque SecDevOps es de alto valor en estos días llenos de hackeos. Para llevar a cabo las buenas prácticas de SecDevOps es necesario considerar que dentro de las metodologías ágiles utilizadas siempre se encuentre la seguridad como un feature dentro del backlog[5] de trabajo, en el caso de estar aplicando metodologías ágiles como Scrum cada Product Owner que dirige un equipo debe trabajar en conjunto con los demás Product Owner. Manteniendo una tarea en común la “Seguridad”.

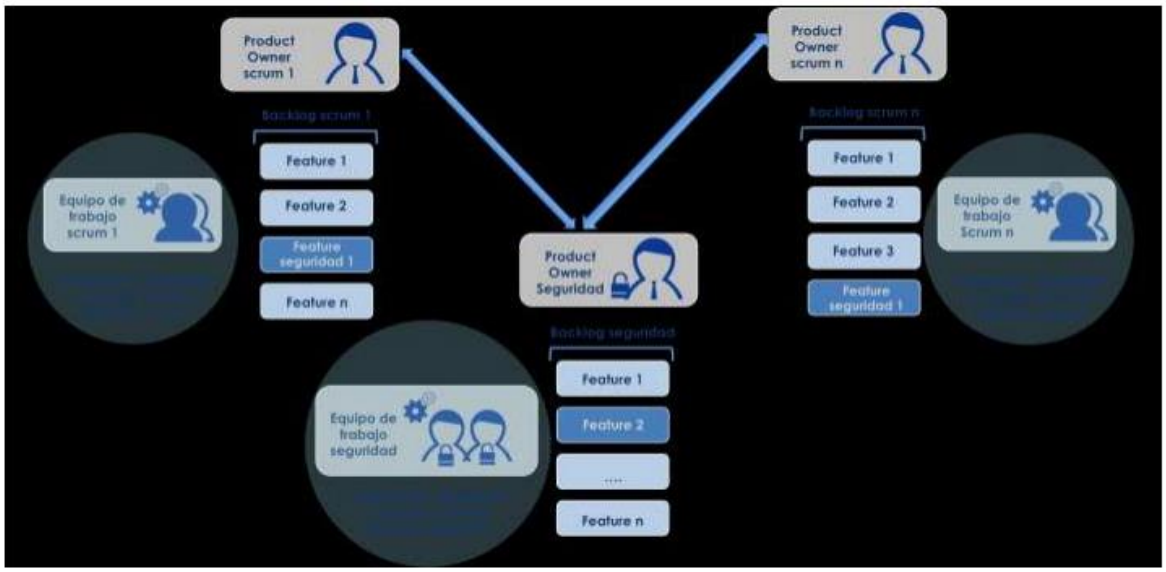


Figura 9 Actividades Metodologías Agiles

[13]

Las organizaciones que utilizan este tipo de metodología ágiles necesitan velocidad en las entregas y que las nuevas características o features que se implementan estén más rápido en producción, todo esto hace que sean más competitivas, pero cada entrega debe garantizar la implementación de la seguridad.

[5] **Backlog**, Se refiere a una lista ordenada de todo el trabajo pendiente. Contenido. Dependiendo del método ágil utilizado, los elementos incluidos en el **Backlog** se denominan ítems, historias de usuario, unidades de trabajo, etc.

Para llevar a cabo estas tareas es necesario recurrir a los Pipelines cuyo objetivo principal es automatizar la creación de nuevas versiones de los productos, ejecutar todas las pruebas y por último generar informes. Por otro lado, como segunda instancia está la integración continua (CI) que se centra principalmente en el camino que recorre el código desde el desarrollador hasta un determinado repositorio.

Otro de los puntos a tener en cuenta es el Despliegue continuo que se encarga de desplegar directamente en un determinado entorno de ejecución las nuevas características o feature una vez que hayan pasado satisfactoriamente por la batería de pruebas funcionales y de seguridad. Por último, está la entrega continua (CD) que es la encargada de pasar a producción de forma manual una vez que todas las pruebas han sido correctas.

Dado que la integración continua, las pruebas continuas, la implementación continua y la entrega continua son los principales aspectos técnicos de DevOps, para seguir dando avance es necesario aclarar estos conceptos bien importantes:

8.6 ¿Qué es la Integración Continua en DevOps?

La integración continúa y popularmente abreviada ‘CI’ es un proceso importante en DevOps, también en considerado como un conjunto de procesos que son llevados a cabo mediante un pipeline denominado ‘Build Pipeline’ o ‘CI Pipeline’ [14].

En las prácticas de DevOps existe una única herramienta de control de versiones para el equipo de Desarrollo y Operaciones, donde el código de todos es alojado en una base de código maestro y esto permite que el equipo pueda trabajar en paralelo.

Por lo tanto, la integración continua en DevOps no es más que fusionar (Mergear) el código de los desarrolladores individuales en la copia maestra del código a la rama principal donde se mantiene el control de versiones. La siguiente imagen ilustra lo anterior mencionado.

Figura 10 Repositorio Central De Código Fuente

[13]

El proceso del CI se compone de los siguientes pasos:

1. Fusión de todo el código de los desarrolladores a la línea Principal.
2. Ejecución del build .
3. Compilación del código y creación del build.
4. Ejecución de pruebas unitarias.

La integración continua es un proceso que combina (merges) todo el código de los desarrolladores en un repositorio central y valida cada una de las combinaciones con una compilación de pruebas automatizadas.

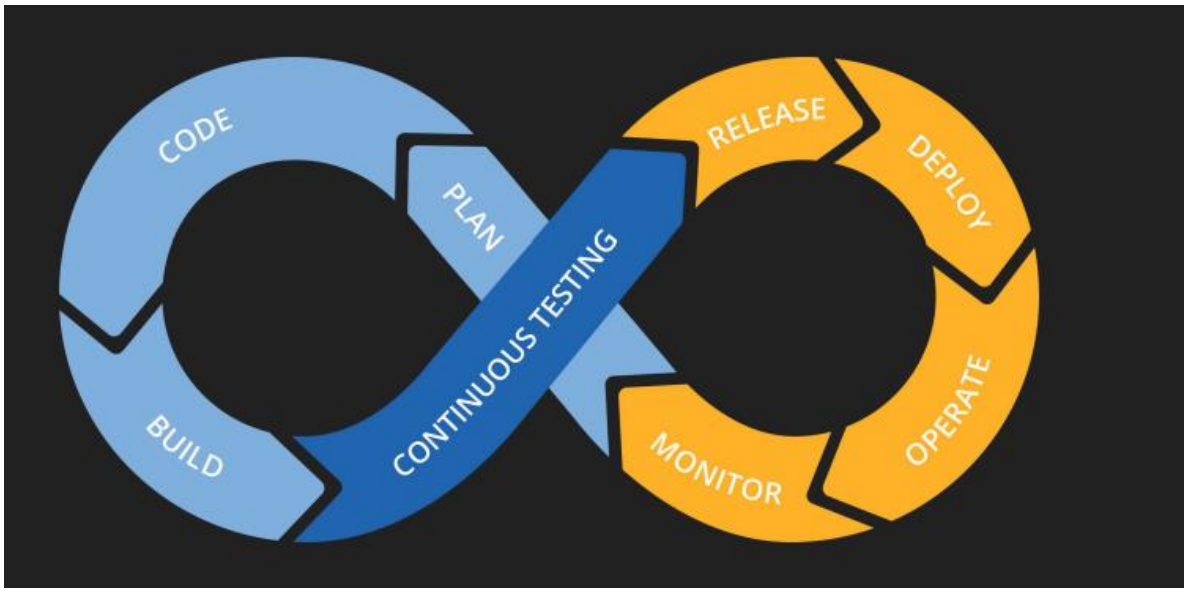


Figura 11 Enfoque DevOps

[15]

A nivel técnico funcional cuenta con un servidor para la integración continua que aloja una herramienta CI integrada con una herramienta de control de versiones para el registro de código, cuando un desarrollador sube los cambios al repositorio central, la herramienta de control de versiones ayuda a combinar el código y a su vez se comunica con las herramienta CI, las cuales activan y compilan en conjunto todas las pruebas, estas pueden ser las pruebas unitarias, las de análisis de código estático y las automatizadas bien sea a nivel de front-end de las aplicaciones, como a nivel de APIS[6] y de Seguridad.

Entre las herramientas más importantes usadas que permiten llevar a cabo las pruebas automatizadas se encuentra Jenkins, por otra parte, esta Bitbucket y GitHub como herramientas de control de versiones de código, por otro lado, tenemos las herramientas que permiten realizar las pruebas de unitarias entre ellas están TestNG, NUnit. En cuanto a las pruebas de análisis estático de código fuente se encuentra la herramienta más usada, SonarQube y por último para llevar a cabo las pruebas de seguridad se encuentra

[6] **API**, Es una sigla que significa Interfaz de programación de aplicaciones.

OwaspZap desarrollada por la comunidad de Owasp, el cual permite actualizar el enfoque devOps a SecDevOps.

Por lo tanto, es posible decir que el proceso de CI es un proceso automatizado sin ninguna intervención manual y se ejecuta en unos pocos segundos o minutos.

8.7 ¿Cuáles son los beneficios del CI? [14]

El principal beneficio del CI es la rápida respuesta que un equipo de desarrollo obtiene en poco tiempo, al ser un proceso automatizado, minimiza el error humano al reducir las largas fusiones de código manuales que inducen a errores.

Cualquier persona en un equipo de Desarrollo y QA puede verificar su código y combinarlo en un repositorio central, sin esperar a que otros completen su codificación, esto ayuda a eliminar la dependencia hacia otros y optimizar tiempo. Por lo tanto, los miembros de un equipo no tienen por qué esperar a que sus compañeros terminen sus tareas, lo que ayuda al trabajo en paralelo.

Cada vez que se sube y se combina código con el repositorio central del control de versiones, este no se queda únicamente allí alojado si no que automáticamente se está calificando de inmediato a través de la compilación y ejecución del conjunto de pruebas automatizadas configuradas, es en este momento que todas las pruebas automatizadas a nivel de front, de API y de seguridad, junto a las pruebas estáticas de condigo fuente, se ejecutan validando los nuevos cambios e informando por medio de reportes al equipo el estado actual a nivel de código fuente, funcional y de seguridad.

La posibilidad de perder el código es nula porque el código es de todos los miembros del equipo y se encuentra registrado en el repositorio maestro o central.

Cabe mencionar que todo el proceso de compilación, creación y prueba se ejecuta en pocos segundos, por lo tanto es mucho más rápido, ahorra mucho tiempo y lo más importante es que ayuda a lograr el objetivo de DevOps de entregar más rápido en un periodo de pocas horas.

8.8 ¿Qué es la Entrega Continua en DevOps?

La entrega continua es el proceso mediante el cual se envía actualizaciones del software a los ambientes productivos por medio de entregas pequeñas y en ciclos de tiempo cortos. Basado en el enfoque DevOps, el equipo siempre estará listo para hacer entregas en cualquier momento a producción [16].

La entrega continua entonces se puede decir que es un pipeline o el ciclo de vida del código, donde el código recientemente desarrollado o actualizado por el equipo de software, es probado en diferentes etapas, mediante pruebas manuales funcionales y seguridad como automática, una vez pasadas este tipo de pruebas es instalado en el ambiente de producción.

El objetivo principal de la entrega continua es construir, probar y liberar al cliente de forma ágil más rápida, frecuente y en ciclos cortos, todo esto con el menor de los costes, tiempo, confiabilidad y teniendo como prioridad la calidad y seguridad como un proceso automatizado.

Cada organización define su pipeline de CI/CD mediante la inclusión de diferentes fases de pruebas dependiendo de las necesidades del proyecto. Incluir pruebas manuales funcionales y de seguridad y automatizadas es indispensable para poder conservar el enfoque SecDevOpS.

8.9 ¿Cuáles son los beneficios del CD? [14]

Entre los principales beneficios que existen en el uso de CD es el aumento del número de entregas.

Minimiza el riesgo de fallo en la producción, dado que los CD son implementaciones pequeñas y frecuentes, eliminan el mayor riesgo de fallas en la producción.

Reduce el trabajo manual, a menos que exista un requisito obligatorio de intervención humana, todo en el pipeline, de principio a fin, está automatizado. Así, se reducen muchos trabajos manuales.

Aumenta la confianza en el equipo. La entrega continua aumenta la confianza en el equipo y el equipo estará siempre preparado para la "entrega a producción" y su mente estará constantemente conectada a la calidad y la velocidad que se espera en la producción.

Permite al equipo automatizar todo. La entrega continua permite enriquecer tanto el desarrollo como las operaciones para automatizar todo lo que está en el pipeline, lo que incluye actividades de desarrollo y operaciones, desencadenamiento, construcción, pruebas unitarias, implementación, definición de configuraciones de infraestructura y entorno como código, niveles más altos de pruebas (funcionales y de seguridad, rendimiento, interfaz de usuario, etc.).

Permite una retroalimentación más rápida, la entrega continua, al ser un ciclo de implementación corto, ayuda al equipo a obtener una retroalimentación más rápida sobre la entrega, que no solo proviene del entorno de desarrollo, sino también del entorno de producción y, por lo tanto, hace que las entregas de software sean bajas. Actividad de estrés, negocio habitual para el equipo.

8.10 Pipeline CD

Como se mencionó anteriormente el objetivo de la CD es aumentar el número de entregas de actualizaciones de un producto de software al ambiente productivo a través de un proceso automatizado, pero esto puede ser posible a través del pipeline.

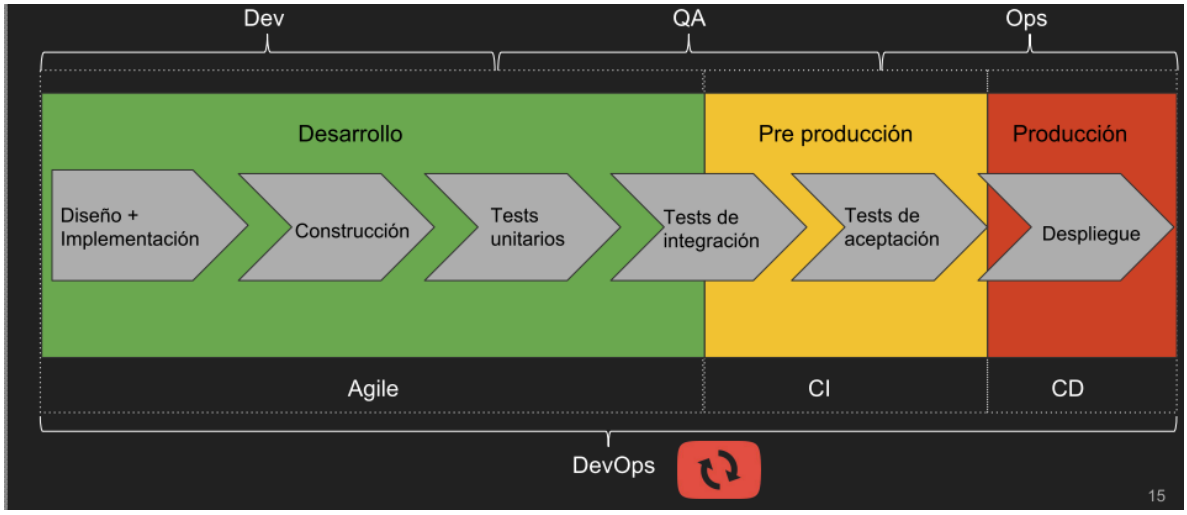


Figura 12 Pipeline DevOps

[4]

El pipeline es una nueva forma de trabajar en el mundo de DevOps en la CI. El pipeline divide el proceso de entrega de software en etapas. Cada etapa tiene una responsabilidad diferente con el fin de validar las nuevas funcionalidades y evitar que los errores afecten a los usuarios. Los pipelines ayudan a los equipos a tener una visibilidad del flujo de cambios en las entregas nuevas de funcionalidades en un software [15].

La configuración de un pipeline estándar no existe, sin embargo, en un CD normal estarían las siguientes etapas: construcción de los binarios de la automatización e integración continua (Build Automation), automatización de pruebas y automatización de despliegue.

En los siguientes títulos se darán a conocer en que consiste cada una de las fases.

8.11 Build automation e integración continua [15]

La primera fase del pipeline es la construcción de los binarios que son los entregables que pasarán a las etapas posteriores. Es decir que, cada vez que un desarrollador implementa nuevas features o funciones se integran en la base de código central de forma continua, se construye y se ejecutan las pruebas por unidad. Esta fase indica de manera continua el estado del código a los equipos de desarrollo.

8.12 Automatización de pruebas [15]

Es una etapa (stage) muy importante donde se prueba la calidad del software. Las nuevas versiones de la aplicación pasan por un proceso de pruebas automatizadas a nivel funcional, de rendimiento y de seguridad todo esto con el objetivo de validar que la aplicación cumpla con todas las cualidades deseadas a nivel funcional y no funcional. En esta etapa se involucran diferentes tipos de pruebas automatizadas y también manuales.

8.13 Deployment automation [15]

Es la entrega del software totalmente automatizada, que ha cumplido con cada una de las anteriores etapas, de esta manera cada funcionalidad nueva desarrollada que llegue a esta última etapa se implementa con toda la confianza [15].

Sin embargo, es necesario dar más detalle de los tipos de pruebas que se pueden llevar a cabo dentro de un pipeline para llegar entender como poder armar cada una de las etapas de Calidad de Software:

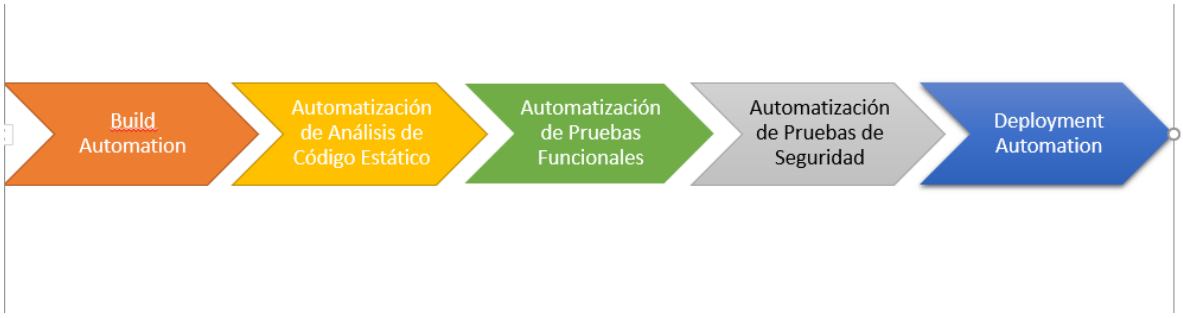


Figura 13 Pipeline QA

8.14 Los tipos de prueba dentro de un pipeline CD

Dentro del pipeline CD existen dos tipos de pruebas que son llevadas a cabo para asegurar la calidad a nivel funcional y de seguridad de una aplicación, entre ellas están BlackBox Testing y WhiteBox Testing.

8.15 BlackBox testing

Este tipo de prueba se ejecuta sin tener en cuenta a nivel interno como funciona el sistema y están más enfocadas en la salida generada por una entrada de información en el sistema. Son consideradas como las pruebas funcionales que abarcan las pruebas de seguridad [17].

8.16 WhiteBox testing

En este tipo de pruebas se tiene en cuenta los mecanismos internos de un sistema. Se lleva a cabo análisis de estructura en el código fuente, se realiza también un análisis de complejidad ciclomatica[7] [17]. Con respecto a la seguridad se realiza un análisis estático de código haciendo uso de diferentes herramientas que ayudan a encontrar vulnerabilidades dentro del código fuente.

Dentro de estas categorías de prueba existen otros tipos de pruebas entre ellas se pueden mencionar las funcionales, del sistema, de estrés y de seguridad.

8.17 Pruebas funcionales / pruebas de aceptación (UAT)

Estos tipos de prueba son utilizados para validar el software y verificar que el comportamiento del sistema sea de acuerdo con los requerimientos o features solicitados por el usuario.

[7] **Complejidad Ciclomatica**, es una métrica del software en ingeniería del software que proporciona una medición cuantitativa de la **complejidad** lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.

8.18 Pruebas del sistema

Asegura que al poner el software en diferentes entornos (por ejemplo, sistemas operativos) todavía funcione.

8.19 Pruebas de estrés

Este tipo de pruebas evalúa cómo se comporta el sistema en condiciones desfavorables.

8.20 Pruebas de seguridad

Las pruebas de seguridad en conjunto con las anteriores pruebas descritas tienen como objetivo identificar las principales vulnerabilidades que una aplicación pueden llegar a tener con el fin de que sean mitigadas de manera proactiva antes de que la aplicación se encuentre en un ambiente productivo.

9.0 Capitulo III: Implementando SecDevOps a partir de la CI/CD

9.1 Introducción

En este capítulo tiene como misión dar a conocer aspectos a nivel de diseño, conceptuales y técnicos de las principales herramientas open source que permiten llevar a cabo una efectiva implementación de la Seguridad mediante el uso de un ambiente de integración y entrega continua CI/CD.

En la medida en que se desarrolla este capítulo se estarán dando a conocer las herramientas a nivel conceptual, las acciones que cada una de

ellas tiene para ofrecer y adicionalmente análisis prácticos que permiten dar una demostración de la funcionalidad de cada una de ellas. Adicionalmente se dará a conocer aspectos técnicos de cómo llevar a cabo la configuración de jobs para integrar estas herramientas de seguridad en un ambiente de CI/CD y cómo generar reportes que puedan estar informando de manera automatizada a los miembros del equipo de QA. De esta manera los miembros del equipo QA pueden estar siempre al tanto del estado a nivel de seguridad de la aplicaciones que están en proceso de testing y gracias a ello, puedan informar a los equipos de desarrollo para la corrección de estas fallas haciendo uso de herramientas de gestión de incidentes.

9.2 Clasificación de vulnerabilidades de aplicaciones web basadas en Owasp

Antes de dar inicio a este capítulo es necesario dar una introducción acerca de la comunidad de Owasp, qué es y cuáles son las vulnerabilidades de seguridad más importantes a tener en cuenta en el momento que se desee implementar seguridad en un ambiente de CI/CD.

La Fundación Open Web Aplication Security Project (OWASP) es una organización sin fines de lucro establecida en 2004. OWASP es una comunidad abierta dedicada a la concepción, desarrollo, adquisición, operación y mantenimiento de aplicaciones confiables para la seguridad de sistemas web y para el uso de cualquier organización que lo requiera. Una de las grandes ventajas de esta comunidad es el tener un grupo numerable de profesionales expertos en seguridad que producen documentación en donde se exponen buenas prácticas en el proceso de las pruebas de seguridad dando a conocer nombre de herramientas que pueden ser utilizadas por cualquier usuario que lo requiera. El propósito principal de esta fundación, según ellos declaran en su página web oficial, es “ser la comunidad global más prospera que impulse la visión de la evolución de la seguridad y protección del software mundial”.

Owasp realiza periódicamente la clasificación de los fallos de seguridad más críticos. Este documento es conocido como el Owasp Top 10 y consiste en una lista de los 10 riesgos de seguridad más críticos para las aplicaciones web según su criterio. Los datos con los que se realiza esta clasificación son basados en la experiencia de expertos de seguridad a nivel mundial, de compañías consultoras y fabricantes de herramientas especializadas en la detección de vulnerabilidades.

Owasp Top 10 es un documento de alto valor en el mundo de la seguridad y su finalidad es incentivar al desarrollo seguro de aplicaciones web. La clasificación de las diez categorías de vulnerabilidades se ha seleccionado de acuerdo con el porcentaje de posibilidad de uso de en un ataque y a la estimación del impacto de la aplicación atacada.

A continuación, se detallan cada una de las vulnerabilidades presentes en la clasificación OWASP Top 10 2017:

Inyección: Las fallas de inyección, como SQL, NoSQL, OS o LDAP ocurren cuando se envían datos no confiables a un intérprete, como parte de un comando o consulta. Los datos dañinos del atacante pueden engañar al intérprete para que ejecute comandos involuntarios o acceda a los datos sin la debida autorización [18].

Pérdida de autenticación: Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son implementadas incorrectamente, permitiendo a los atacantes comprometer usuarios y contraseñas, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios (temporal o permanentemente) [18].

Exposición de datos sensibles: Muchas aplicaciones web y APIs no protegen adecuadamente datos sensibles, tales como información financiera, de salud o Información Personalmente Identificable (PII). Los atacantes pueden robar o modificar estos datos protegidos inadecuadamente para llevar

a cabo fraudes con tarjetas de crédito, robos de identidad u otros delitos. Los datos sensibles requieren métodos de protección adicionales, como el cifrado en almacenamiento y tránsito [18].

Entidades externas XML: Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML. Las entidades externas pueden utilizarse para revelar archivos internos mediante la URI o archivos internos en servidores no actualizados, escanear puertos de la LAN, ejecutar código de forma remota y realizar ataques de denegación de servicio (DoS) [18].

Pérdida de control de acceso: Las restricciones sobre lo que los usuarios autenticados pueden hacer no se aplican correctamente. Los atacantes pueden explotar estos defectos para acceder, de forma no autorizada, a funcionalidades y/o datos, cuentas de otros usuarios, ver archivos sensibles, modificar datos, cambiar derechos de acceso y permisos, etc. [18].

Configuración de seguridad incorrecta: La configuración de seguridad incorrecta es un problema muy común y se debe en parte a establecer la configuración de forma manual, ad hoc o por omisión (o directamente por la falta de configuración). Son ejemplos: S3 buckets abiertos, cabeceras HTTP mal configuradas, mensajes de error con contenido sensible, falta de parches y actualizaciones, frameworks, dependencias y componentes desactualizados, etc. [18].

Secuencia de comandos en sitios cruzados (XSS): Los XSS ocurren cuando una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada; o actualiza una página web existente con datos suministrados por el usuario utilizando una API que ejecuta JavaScript en el navegador. Permiten ejecutar comandos en el navegador de la víctima y el atacante puede secuestrar una sesión, modificar (defacement) los sitios web, o redireccionar al usuario hacia un sitio malicioso [18].

Desacralización insegura: Estos defectos ocurren cuando una aplicación recibe objetos serializados dañinos y estos pueden ser manipulados o borrados por el atacante para realizar ataques de repetición, inyecciones o elevar sus privilegios de ejecución. En el peor de los casos, la deserialización insegura puede conducir a la ejecución remota de código en el servidor [18].

Componentes con vulnerabilidades conocidas: Los componentes como bibliotecas, frameworks y otros módulos se ejecutan con los mismos privilegios que la aplicación. Si se explota un componente vulnerable, el ataque puede provocar una pérdida de datos o tomar el control del servidor. Las aplicaciones y API que utilizan componentes con vulnerabilidades conocidas pueden debilitar las defensas de las aplicaciones y permitir diversos ataques e impactos [18].

Registro con monitoreo insuficiente: El registró y monitoreo insuficiente, junto a la falta de respuesta ante incidentes permiten a los atacantes mantener el ataque en el tiempo, pivotear a otros sistemas y manipular, extraer o destruir datos. Los estudios muestran que el tiempo de detección de una brecha de seguridad es mayor a 200 días, siendo típicamente detectado por terceros en lugar de por procesos internos [18].

9.3 Owasp Zap [19]

Es una herramienta desarrollada por OWASP y se caracteriza por ser una de las herramientas de seguridad open source más populares del mundo. Diariamente recibe mantenimiento de cientos de usuarios voluntarios internacionales. Permite encontrar automáticamente vulnerabilidades de seguridad en aplicaciones web y es considerada como una herramienta que

deja una buena experiencia en el momento de realizar pruebas de seguridad manual. Su diseño y configuración permite ser usado tanto por usuarios expertos, con un alto nivel de conocimiento de este tipo de herramientas, así como por desarrolladores y personal de QA o usuarios en general, que no están familiarizados o son nuevos en la realización de pruebas de penetración en aplicaciones web. Puede realizar distintos tipos de análisis y ataques al configurar perfiles específicos para ajustarlos a las características de la aplicación que se desea analizar. Es utilizado como un escáner automático, en su conjunto, pero también presenta una serie de herramientas individuales que facilitan la búsqueda de vulnerabilidades de forma manual.

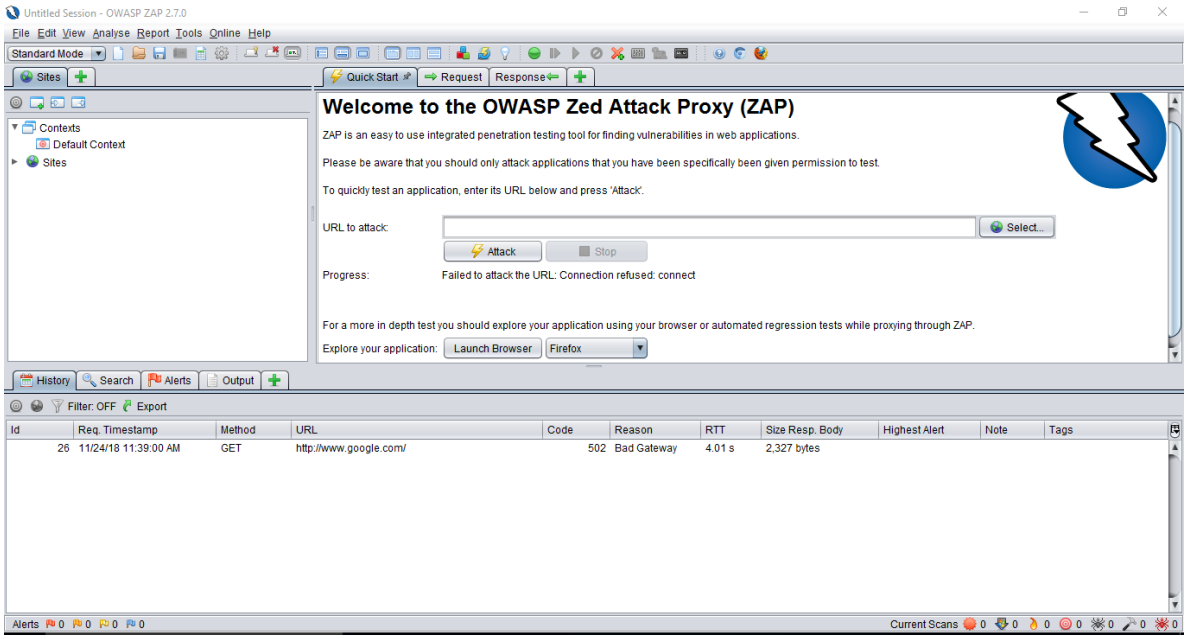


Figura 14 GUI Owasp Zap

Con OWASP ZAP se pueden realizar dos tipos de análisis de forma independiente o conjunta:

Escaneo activo: es el modo más utilizado por quienes emplean esta herramienta. El escaneo activo aplica políticas de búsqueda que pueden ser configuradas por el usuario o puede utilizar sencillamente la política por defecto que hace un análisis completo de la aplicación web en busca de vulnerabilidades.

Escaneo pasivo: en este modo de análisis OWASP ZAP etiqueta de manera automática aquellas respuestas por parte de la aplicación web, en base un grupo de reglas específicas del usuario. La herramienta no realiza ningún análisis intrusivo.

Complementos: OWASP ZAP soporta la instalación de complementos para agregar mayor funcionalidad a la herramienta.

Alertas: Diferenciación de alertas de vulnerabilidades mediante el uso de gráficos con colores que determinan los niveles de gravedad de cada vulnerabilidad hallada (Alta, Media, Baja, Información, Falso Positivo)

Anti CSRF Tokens: Permite la detección del uso de tokens en el envío de formularios, aumentando de esta forma la dificultad de realizar ataques CSRF. Con el uso de esta funcionalidad OWASP ZAP puede identificar estos tokens y almacenarlos conjuntamente con la URL que lo generó, para usarlo en ataques de este tipo.

API: OWASP ZAP provee de un API a través de la cual se puede interactuar con la herramienta mediante programación. Soporta los formatos JSON, HTML y XML. Con el uso de esta API se puede acceder a funcionalidades principales como Escaneo activo y Spider

Autenticación: Soporta distintos métodos de autenticación para el acceso hacia las aplicaciones web objeto de análisis. Los principales son el Método Manual y el basado en formularios. El primero permite al usuario ingresar a la aplicación mediante la introducción de sus credenciales durante la fase de captura y navegación proxy. El Basado en Formulario es usado cuando la autenticación se realiza mediante el envío de un script a través de una petición GET. El script debe contener el par credencial de Usuario/Contraseña otorgado por el usuario.

Break Points: con esta funcionalidad OWASP ZAP puede interceptar todas las peticiones y respuestas para posteriormente modificarlas. Con

esta funcionalidad se puede realizar un bypass en la validación del usuario en la aplicación y obtener sus credenciales.

Contexts: permite asociar diferentes tipos de URL's con un mismo sitio.

Data Driven Content: con la utilización de Contexts, se reduce el número de peticiones en una misma página de un sitio.

Filtros: puede realizar filtros sobre cada petición y respuesta, durante el reconocimiento del sitio, utilizando la función de Proxy.

Exclusión de URL's(forma global): con el uso de expresiones regulares ZAP ignora las URL's que coincidan, con estas expresiones, dentro de la aplicación durante todo el proceso de análisis.

Sesiones HTTP: almacena y utiliza distintas sesiones, mediante Cookies dentro de una misma aplicación, forzando a que todas las peticiones se hagan en una sesión particular sin destruir ninguna otra.

Proxy interceptor: funcionalidad principal de la herramienta que permite ver y almacenar un registro de todas las peticiones que se hacen hacia una aplicación web y sus respectivas respuestas.

Interfaz: ZAP permite cambiar el aspecto de su interfaz para cumplir determinados roles y facilitar el acceso hacia funcionalidades propias de cada rol (Modo Seguro, Modo Protegido, Modo Estándar, Modo Ataque).

Reglas: OWASP ZAP soporta reglas para análisis activo o pasivo, todas las reglas que utiliza ZAP son complementos propios de la herramienta, su actualización es sencilla a través del administrador de complementos.

Alcance: OWASP ZAP permite determinar el alcance de búsqueda en la aplicación, utilizando el grupo de URL's relacionadas en la característica de contexto. Por defecto el alcance es nulo, es decir, no tiene límite.

9.4 Diseño y arquitectura de la seguridad en un ambiente CI/CD

Para lograr lo anteriormente mencionado es necesario armar la arquitectura que se detalla en la siguiente imagen:

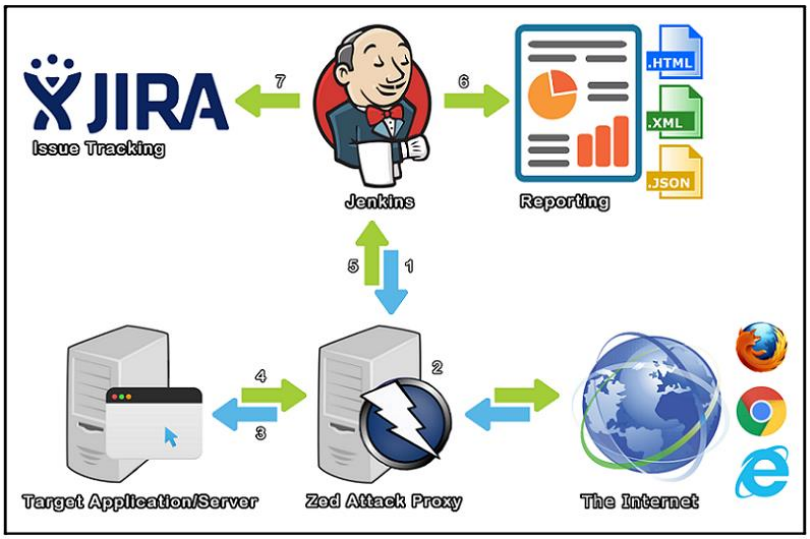


Figura 15 Arquitectura de CI/CD Seguro

[20]

Esta imagen detalla una arquitectura de cómo se puede llevar a cabo de una manera sencilla y sin que requiera tanto esfuerzo por parte de un equipo de QA y Dev con respecto a la implementación de la seguridad dentro de un ambiente CI/CD.

Es necesario dar una explicación de esta arquitectura, para lo cual se supone que existe un equipo de miembros de QA probando una aplicación de manera manual o automatizada. Esta aplicación está siendo probada en los diferentes navegadores existentes en el mercado bien puede ser Chrome, Firefox o Edge. Todas las peticiones que el QA realiza a medida que prueba la aplicación pasan a través de un proxy que mapea todas estas acciones.

Como lo detalla la siguiente imagen, el proxy al que hace referencia es a Owasp Zap. Este proxy es configurable, es decir que es posible configurar las pruebas de seguridad que sean necesarias; sin embargo, la imagen hace referencia a la comunicación de un Job en Jenkins con la API de este proxy.

Figura 16 Auditoria de Seguridad Con ZAP *[20]*

De manera continua la aplicación está siendo auditada a nivel de seguridad por Zap. A medida que surgen nuevas características en el desarrollo, van siendo revisadas por el proxy, el cual va dando a conocer reportes en diferentes formatos, por ejemplo, json, xml y html.

Figura 17 Integración Jenkins con Zap API *[20]*

Estos reportes dan a conocer el estado actual a nivel de seguridad de la aplicación y ayudan a que los equipos de desarrollo puedan tomar contramedidas tempranas antes de que la aplicación salga a un ambiente productivo. Esto es importante, dado que la corrección de errores funcionales y de seguridad encontrados en ambientes operacionales vale mucho más que los encontrados en ambientes de prueba.

Figura 18 Generación de Reportes de Seguridad

[20]

Es así que los equipos de calidad QA pueden tener una herramienta eficaz y automatizada que permite informar de cualquier incidente en seguridad. Por este motivo, el ambiente de CI/CD Jenkins se integra con un gestor de incidentes, para este caso Jira, y por medio de esta herramienta se notifican los incidentes de seguridad o funcionales a los equipos de Dev.

Figura 19 Integración Con Jira (Gestor de Incidentes)

[20]

9.5 Análisis práctico: Integración de Owasp Zap en un ambiente CI/CD [21]

En este análisis práctico la idea es dar a conocer cómo empezar a implementar SecDevOps dentro de un ambiente de Integración y Entrega continua CI/CD. Para ello la idea es configurar un Job en Jenkins que permita

interactuar con la API de Zap y ejecute las pruebas de seguridad de forma automatizadas.

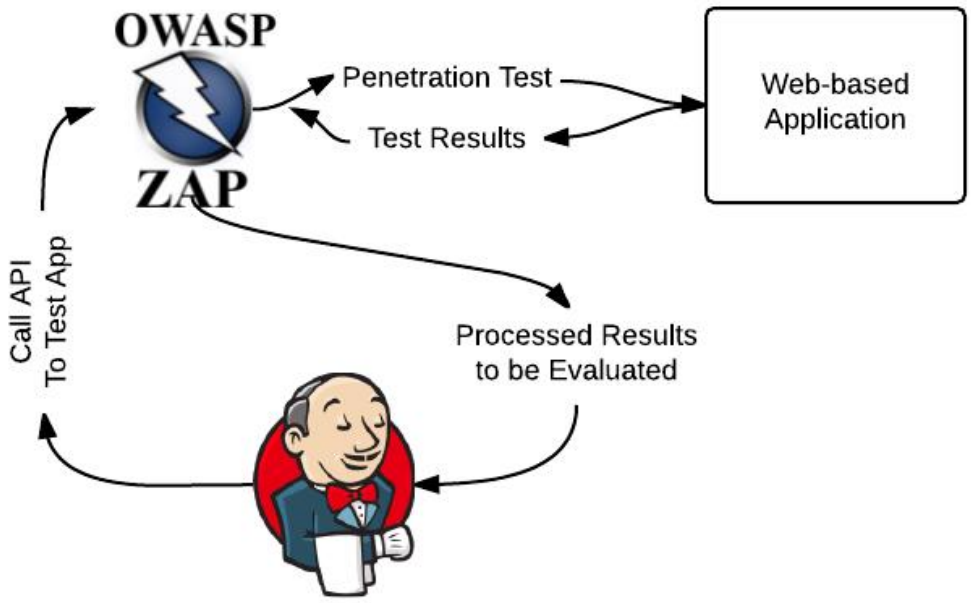


Figura 20 Jenkins Integración Con ZAP API y Auditoria de Seguridad

[20]

9.5.1 Descripción de hardware utilizado

Las descripciones técnicas del equipo en el cual se realizó la prueba de concepto son:

Sistema Operativo: Windows 10 Home Single Languaje

Procesador: AMD A10-5750M APU with Radeom(tm) HD Graphics 2.50 GHz

Memoria RAM: 8.0 GB

9.5.2 Descripción del software utilizado

Jenkins war v 2.161

Servidor de Aplicaciones Apache Tomcat 8.5

Owasp Zap 2.7

Altoro Mutual Web Vulnerable de IBM

9.6 Configuración de pruebas automáticas en Jenkins con Owasp Zap

La siguiente imagen muestra un diseño en que se puede llevar a cabo la integración de un ambiente de CI/CD, como lo es Jenkins, con la aplicación para pruebas de seguridad Owasp Zap.

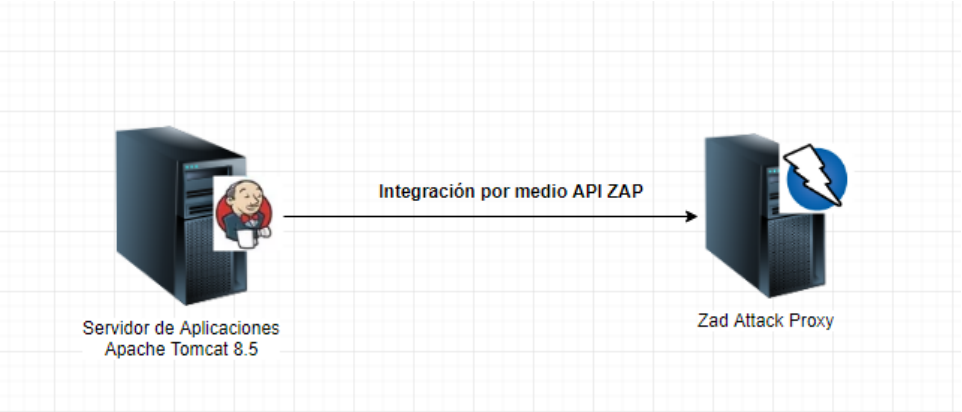


Figura 21 Integración Jenkins con API ZAP

9.7 Creación de un Job para la integración con Owasp Zap

9.7.1 Primer paso

Para poder integrar Zap con Jenkins es necesario instalar el plugin oficial de Owasp Zap. Para esto es necesario acceder Administrador de Jenkins / Administrador de plugins. Se debe buscar el plugin Oficial de Owasp Zap y proceder con su instalación.

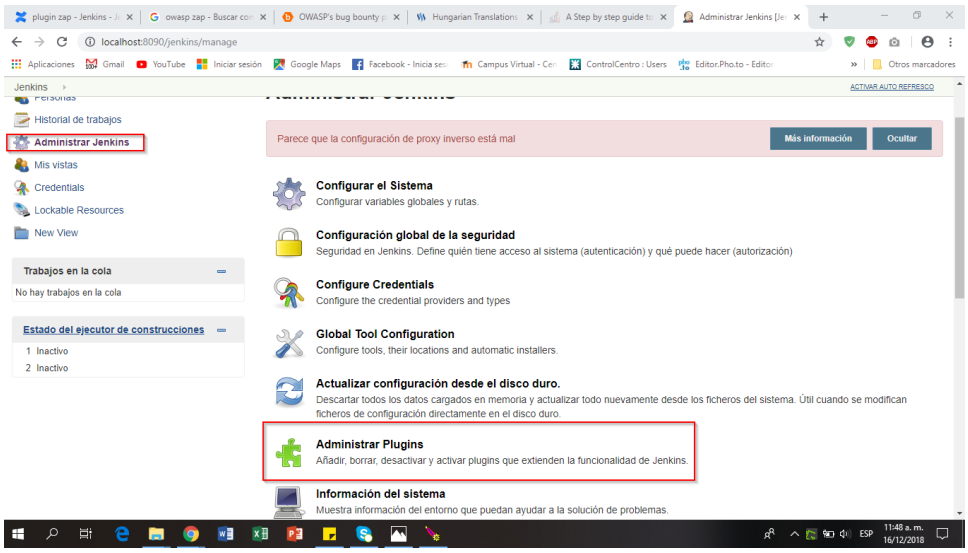


Figura 22 Menú Administrar Plugin

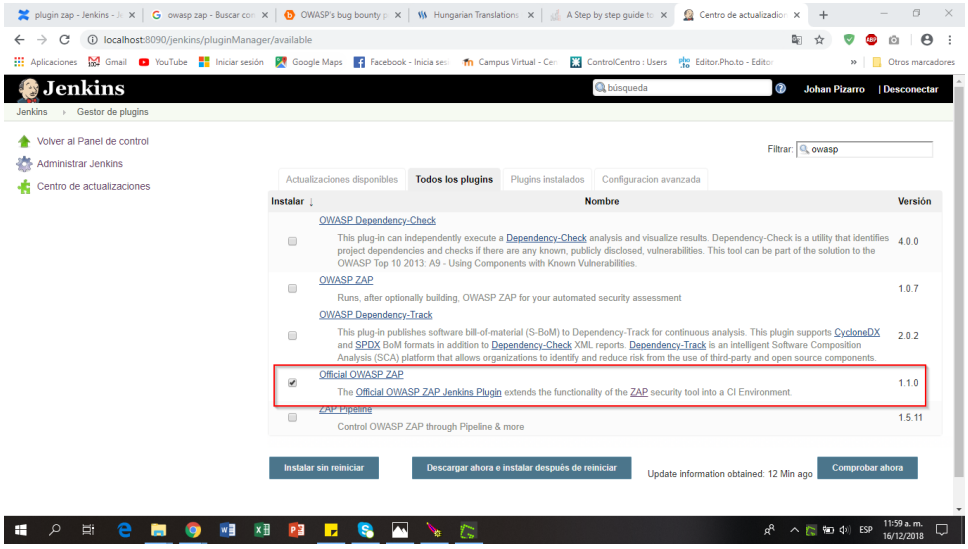


Figura 23 Plugin Official Owasp ZAP

9.7.2 Segundo paso

Cuando se ha instalado el plugin lo siguiente a realizar es configurar el host zap y puerto. Para ello se ingresa al Administrador de **Jenkins/**

Configuración Sistema y en estos campos se deberá ingresar la IP del servidor donde está instalado el proxy ZAP y su puerto.

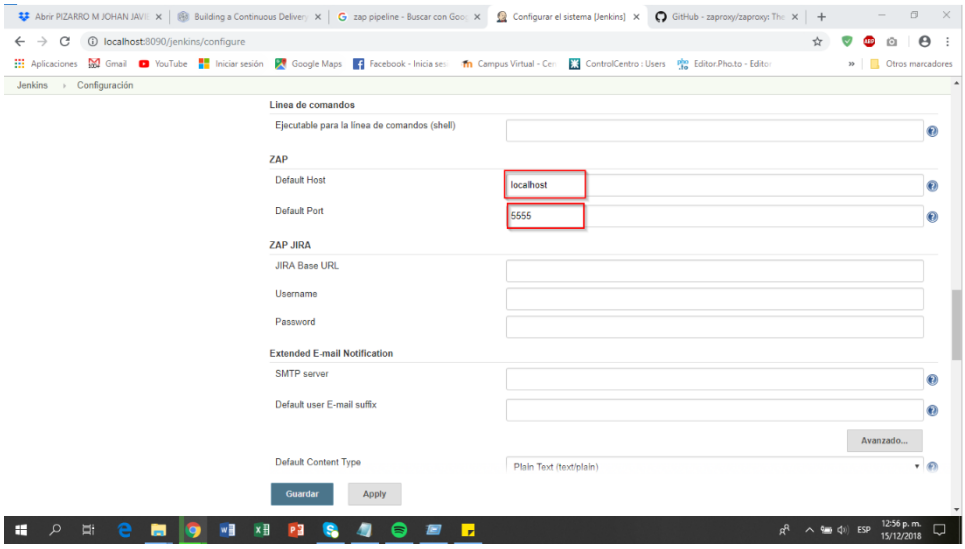


Figura 24 Configuración del Puerto de Escucha Para ZAP

9.7.3 Tercer paso

Luego se debe crear un jobs en Jenkins. Para ello es necesario ir a Crear una Nueva Tarea/ Ingresar Nombre de Tarea / Crear un Proyecto de Estilo Libre

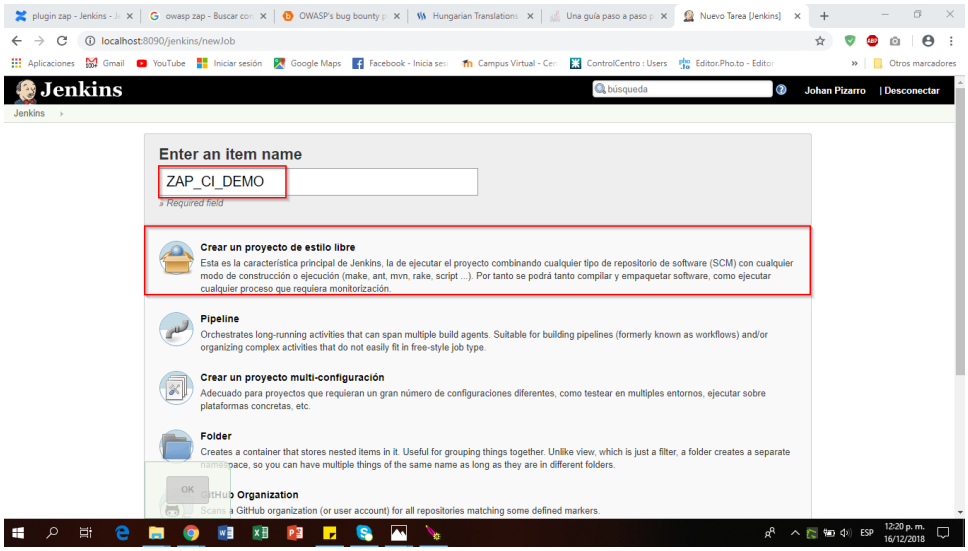


Figura 25 Creación de un Job

9.7.4 Cuarto paso

En este paso es necesario crear nuestro Espacio de Trabajo o workSpace que es el fichero donde se crearán todas las configuraciones de nuestro jobs. Para ello, primero se guarda el job sin necesidad de realizar ninguna configuración sobre él y, luego, es necesario ir a "Construir Ahora" con el fin de crear el workSpace.

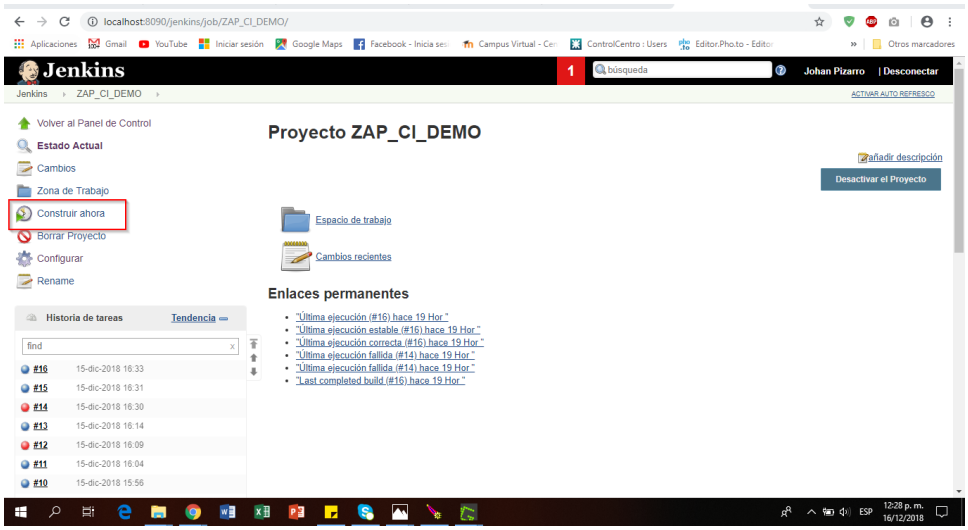


Figura 26 Build del Nuevo Job

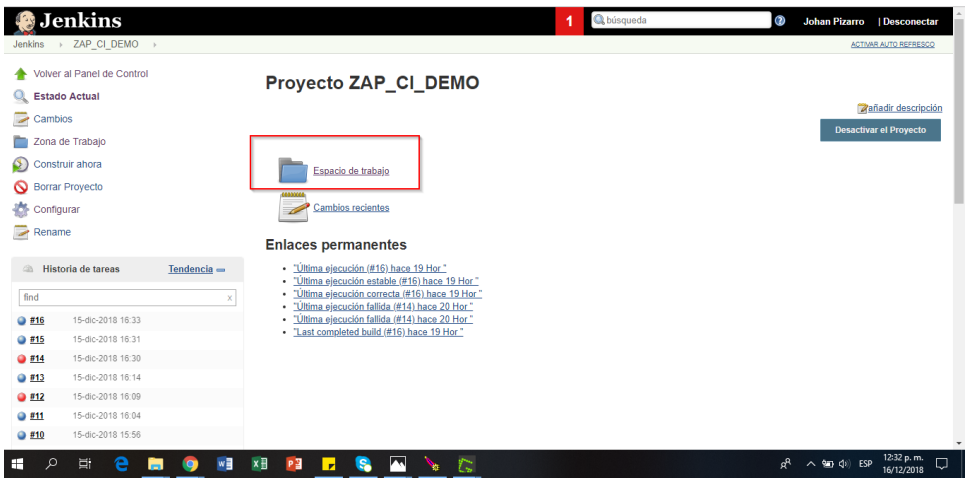


Figura 27 Creación del WorkSpace Para Nuevo Job

9.7.5 Quinto paso

En este paso se procede a configurar el Job. Se accede a Configurar y dentro de la configuración de Job se selecciona la pestaña Ejecutar.

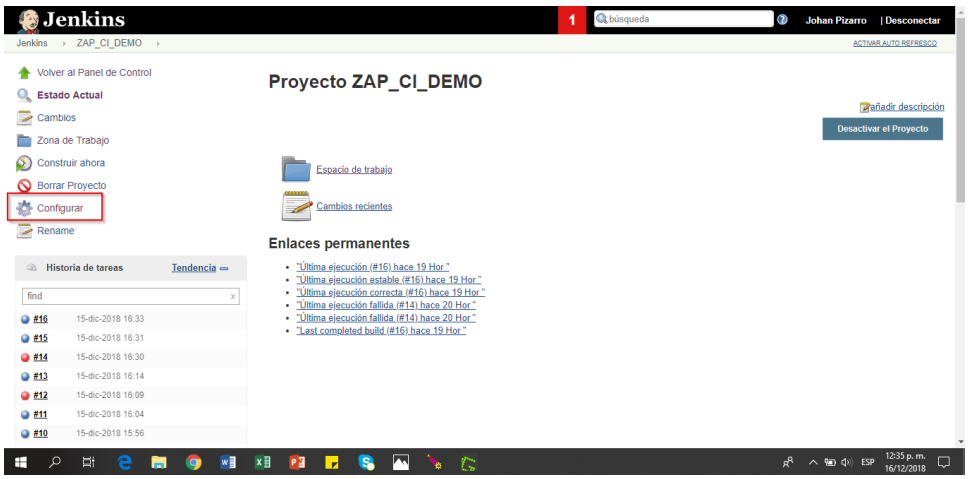


Figura 28 Menú Configuración

Allí se debe ingresara al apartado Ejecutar Zap. Este paso tiene la finalidad de que Jenkins sepa cómo iniciar ZAP y dónde ubicar el directorio instalado de ZAP.

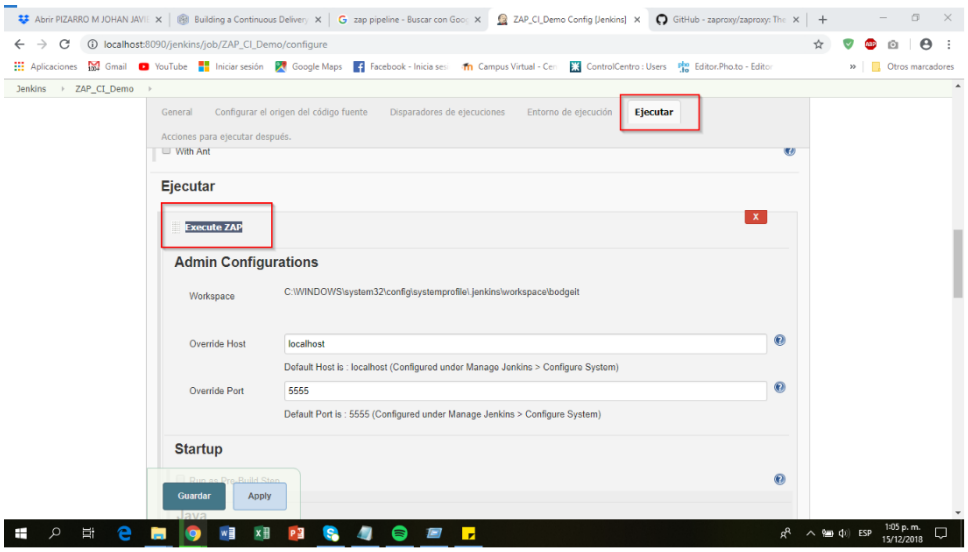


Figura 29 Configuración de Administrador para ZAP

9.7.6 Sexto paso

En este paso es necesario indicarle a Jenkins la dirección de donde se tiene instalado la aplicación ZAP, para ello es necesario ir a Administrar Jenkins/ Configurar el Sistema/ Ir a Propiedades Globales/ Checkear Variables de Entorno. Allí se debe ingresar el nombre de la variable de entorno y la ruta donde se encuentra alojado el Owasp Zap y guardar los cambios.

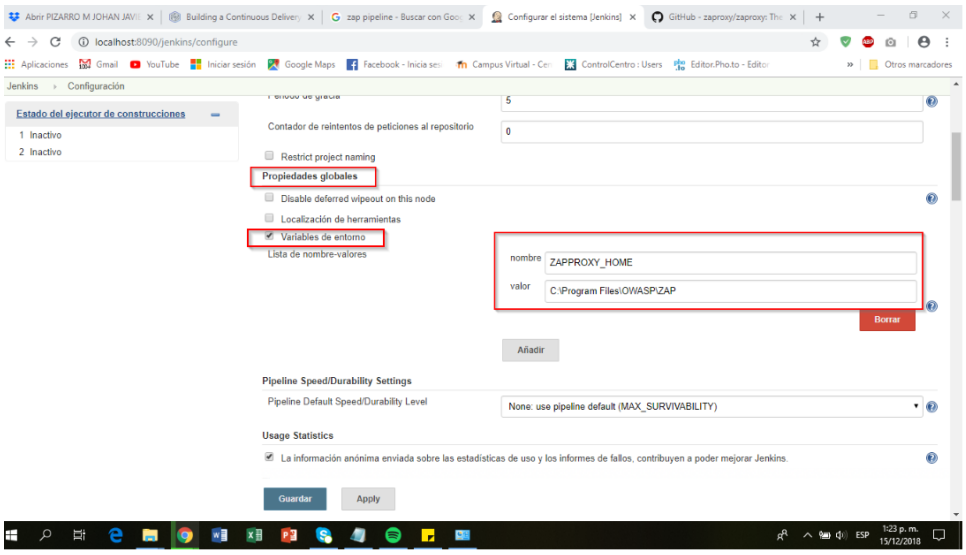


Figura 30 Configuración Path de Origen ZAP

9.7.7 Séptimo paso

Nuevamente se debe ingresar a la configuración del Job y en el apartado Método de Instalación seleccionar el radio buttom Directorio de Instalación de Zap y Seleccionar la variable de entorno previamente configurada.

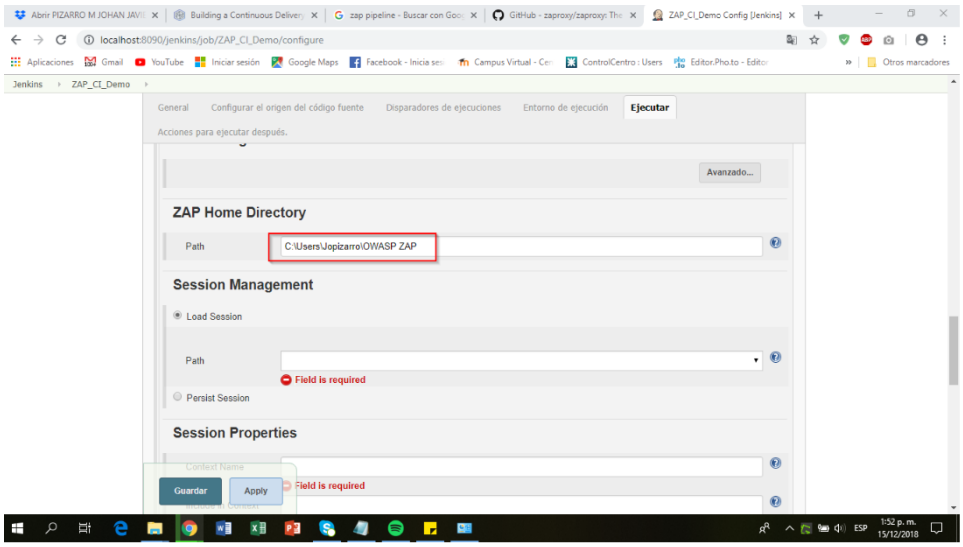


Figura 31 Aplicar el Path de Origen ZAP

9.7.8 Octavo paso

Hasta el momento ya se tiene configurado la integración de Zap con Jenkins de tal manera que Zap se active durante el proceso de construcción o build. En este paso se debe proporcionar la URL sobre la que se realizarán las pruebas. Para ello es necesario ingresar en la pestaña Ejecutar y crear una Session Properties.

A continuación, se proporciona el nombre de la aplicación y la URL dentro del campo Context Name y debajo en los campos Include In Context y Exclude from Context se deben ingresar los parámetros que no se tendrán en cuenta en el escaneo activo (Active Scan) y la exploración activa (Spidering)

Por ejemplo, se quiere incluir dentro del contexto http://demo.testfire.net/* donde http://demo.testfire.net es la aplicación destino y * indica todas las rutas a través de la aplicación.

Luego como opcional se puede armar una expresión regular de lo que en la URL deseo que se excluya del contexto.

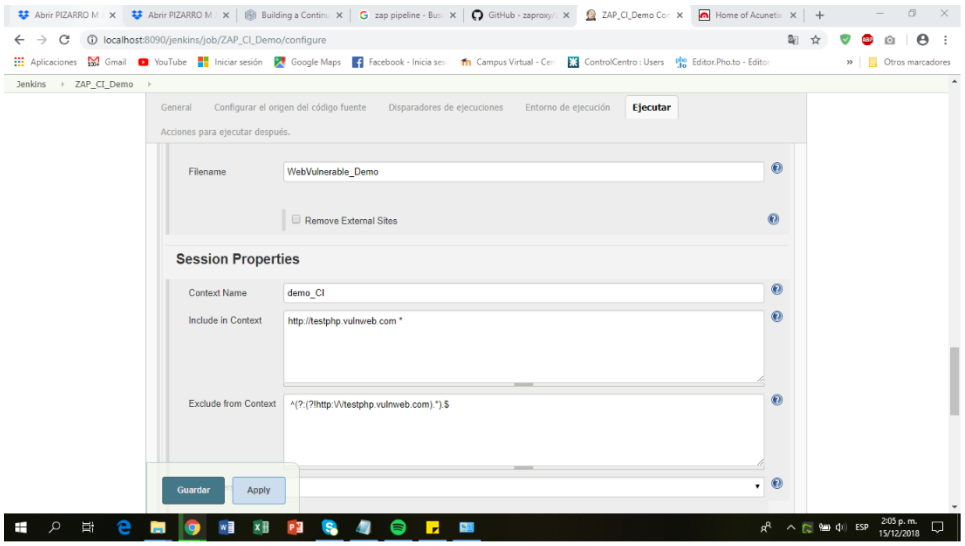


Figura 32 Excluir del Contexto

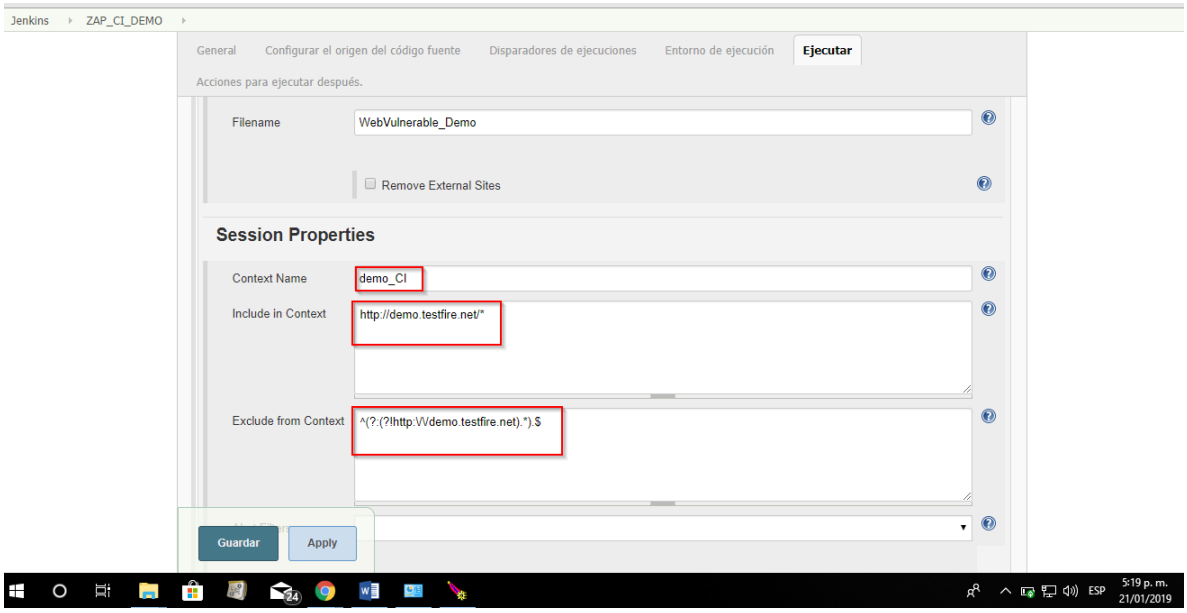


Figura 33 Configuración de la Session Properties

9.7.9 Noveno paso

En este paso se trata de indicar cuál será el modo de ataque que le indicaremos a Zap para que realice sobre la aplicación de prueba. Aquí es necesario acceder al apartado Attack Mode, ingresar la URL en el campo starting point. Ppara generar un rastreo de la aplicación se selecciona la

opción Spider Scan. Por otro lado, se debe habilitar el Active Scan y seleccionar la política de la lista despegable, si es la primera vez que se ejecuta ZAP en Jenkins, este provee una política por defecto.

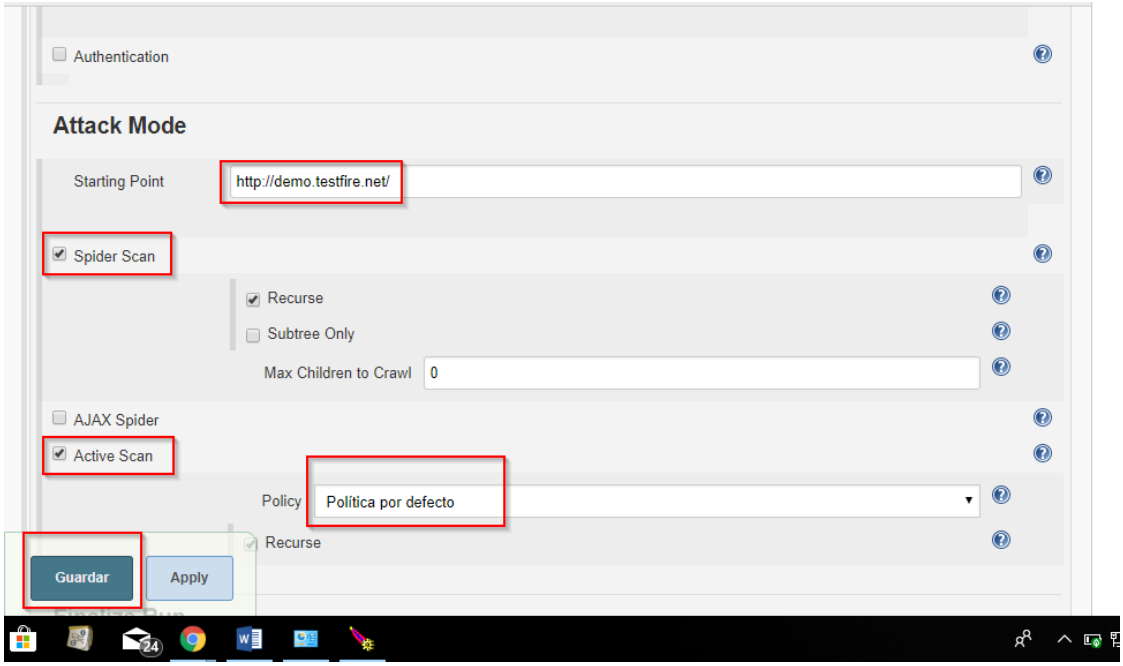


Figura 34 Configuración del Modo de Ataque

9.7.10 Décimo paso

Esta es una de las configuraciones más importantes dado que aquí se configura la forma en que Jenkins permite comunicarse con Zap con el objetivo de generar reportes. Para eso es necesario ir a la pestaña Post Build Actions y seleccionar Finalize Run. Luego se debe habilitar la casilla Generate Reports y proporcionar un nombre del archivo único que se genera por cada iteración.

JENKINS_ZAP_VULNERABILITY_REPORT _ $ {BUILD_ID}

JENKINS_ZAP_VULNERABILITY_REPORT_: Este es un prefijo constante

$ {BUILD_ID} : Esta es la variable de entorno Jenkins, que siempre es única en la compilación actual.

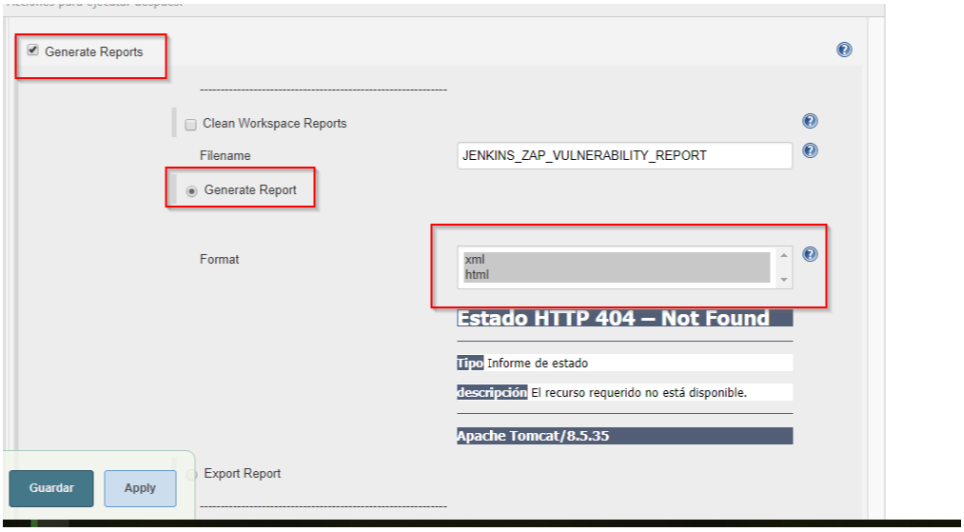


Figura 35 Configuración del Report

9.7.11 Onceavo paso

Por último, se configura los tipos de archivos que se desean almacenar y se indica el nombre del directorio donde se desea alojar los reportes.

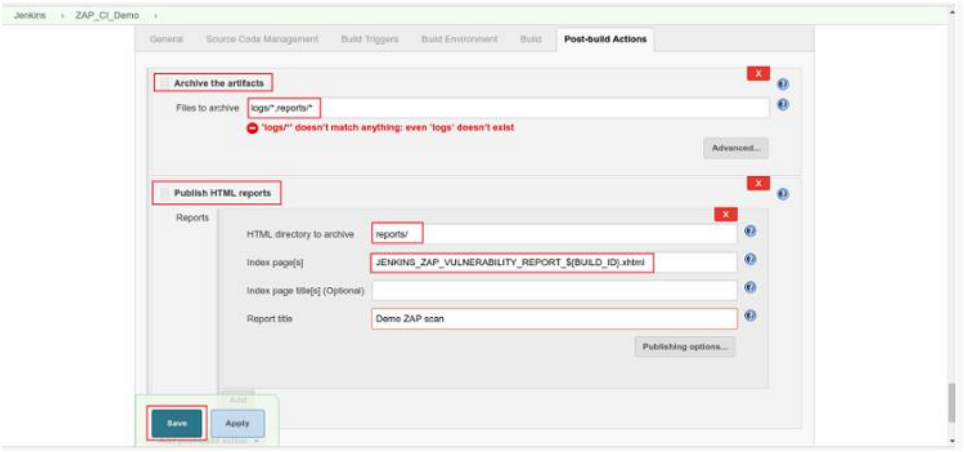


Figura 36 Configuración del Path del Report

9.7.12 Ejecución del job en Jenkins

Cuando todo está configurado es posible ejecutar el job y ver cómo este llama a la API de Zap y empieza a generar las pruebas de seguridad que se configuraron y los respectivos reportes.

En la imagen a continuación se puede apreciar lo que arroja en consola del job en el momento que empieza a compilar las acciones configuradas en los pasos previos.

Figura 37 Construcción del Job Completo

En esta imagen se puede apreciar el momento en que ejecuta las políticas por defecto de Zap y el sitio web al cual se configuró para realizar las pruebas de seguridad automatizadas, dando a conocer por cada prueba el log del status de la prueba, las alertas encontradas.

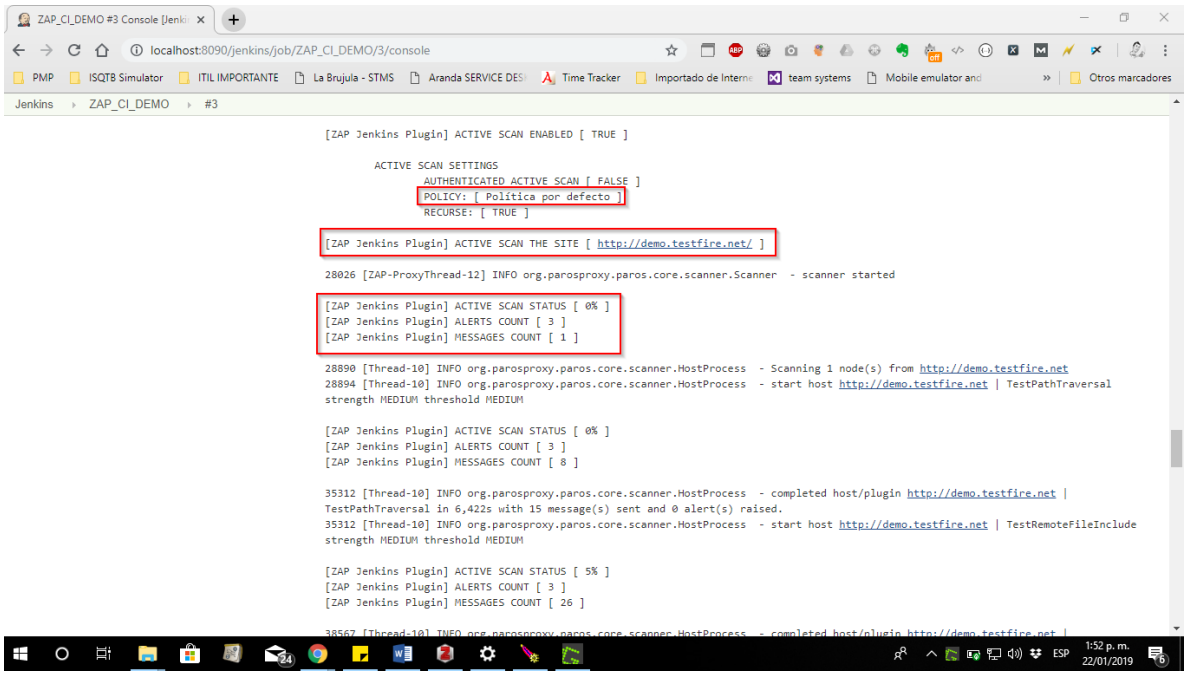


Figura 38 Log de Estatus de las Pruebas

9.7.13 Generación de reporte de vulnerabilidades

Esta es una de las funcionalidades más importantes en toda esta integración, dado que es aquí donde Zap identifica todas las vulnerabilidades de seguridad dentro del CI/CD. Además, lo da a conocer por medio de un reporte que se actualiza cada vez que es ejecutado el job,cuyo nombre en este laboratorio es “ZAP_CI_DEMO”. La idea consiste en que cada vez que exista un cambio en el código de una aplicación, por medio de un pipeline en Jenkins este corra e integre los cambios y llame al nodo o Job “ZAP_CI_DEMO” que ejecuta las pruebas de seguridad de forma automática y genera los reportes actualizados.

En la siguiente imagen se puede apreciar como job termina su ejecución de manera exitosa.

Figura 39 Ejecución Exitosa

En esta imagen es posible apreciar la ubicación que aloja Jenkins del reporte de vulnerabilidades cuyo nombre para esta prueba de laboratorio es “Demo Zap Scan”.

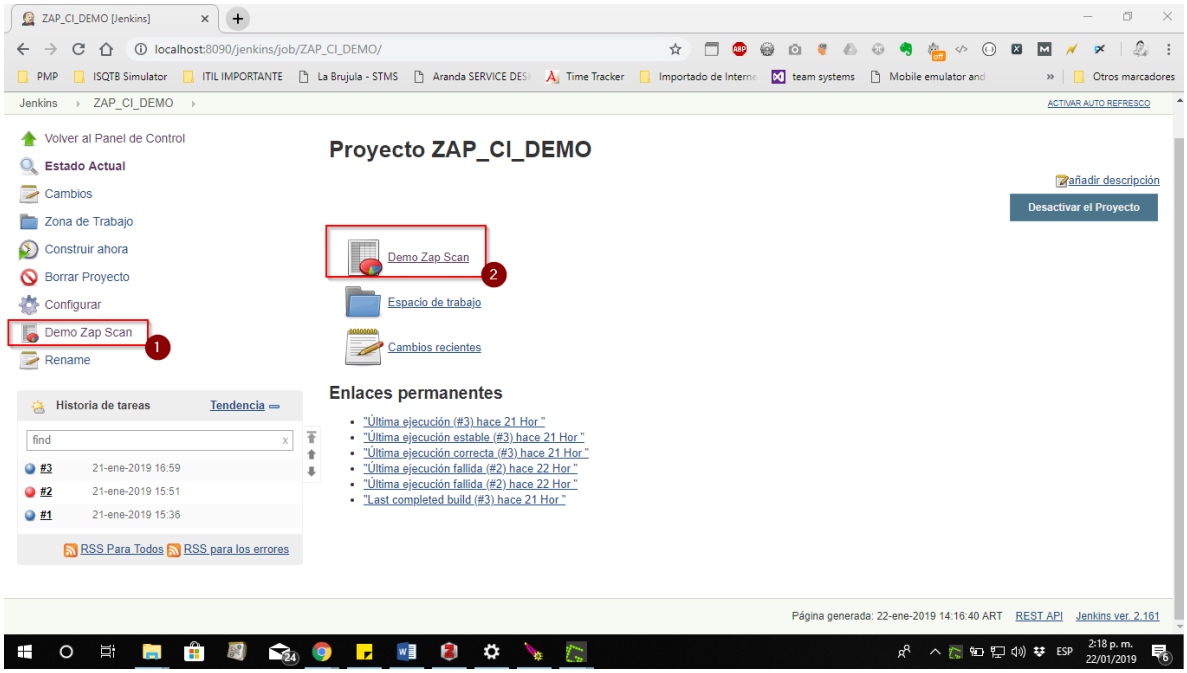
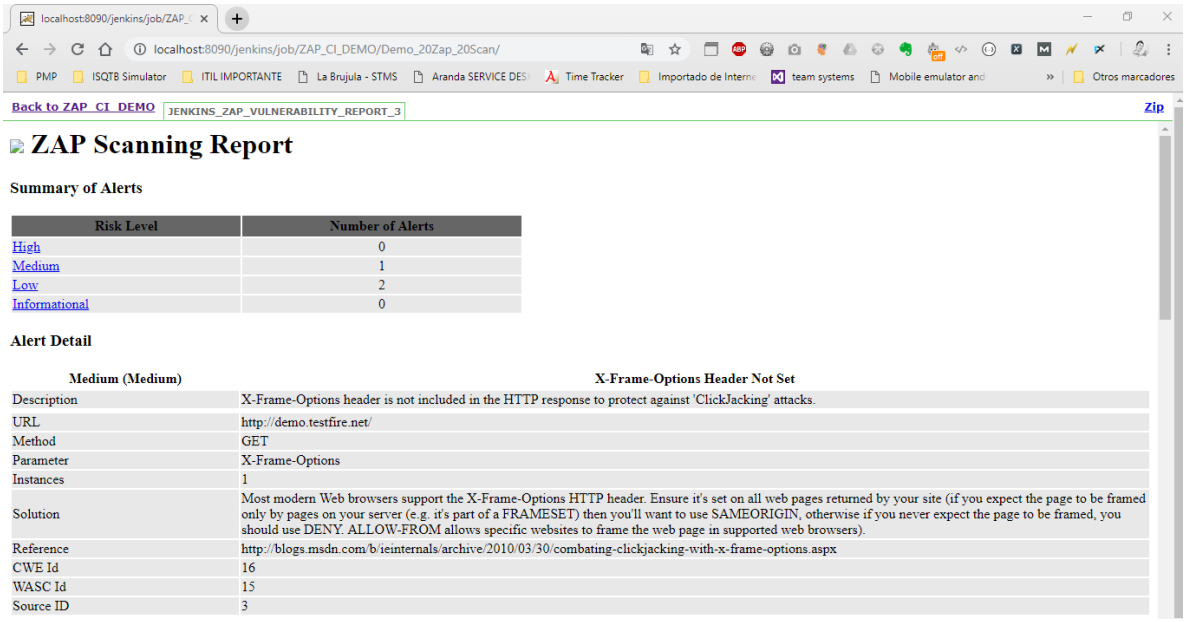


Figura 40 Generación del Reporte

Al ingresar al reporte se muestran las detecciones de seguridad realizadas por la herramienta Owasp Zap. Se encuentran clasificadas de acuerdo a su nivel de riesgo (Alto, Medio, Bajo, Informativo). Por otro lado, da a conocer los detalles de la detección con una breve descripción, la URL del sitio que se está realizando el análisis y la descripción de la solución que permite generar las contramedidas correspondientes para mitigar las alertas.



Back to ZAP_CI_DEMO | JENKINS_ZAP_VULNERABILITY_REPORT_3 | Zip

ZAP Scanning Report

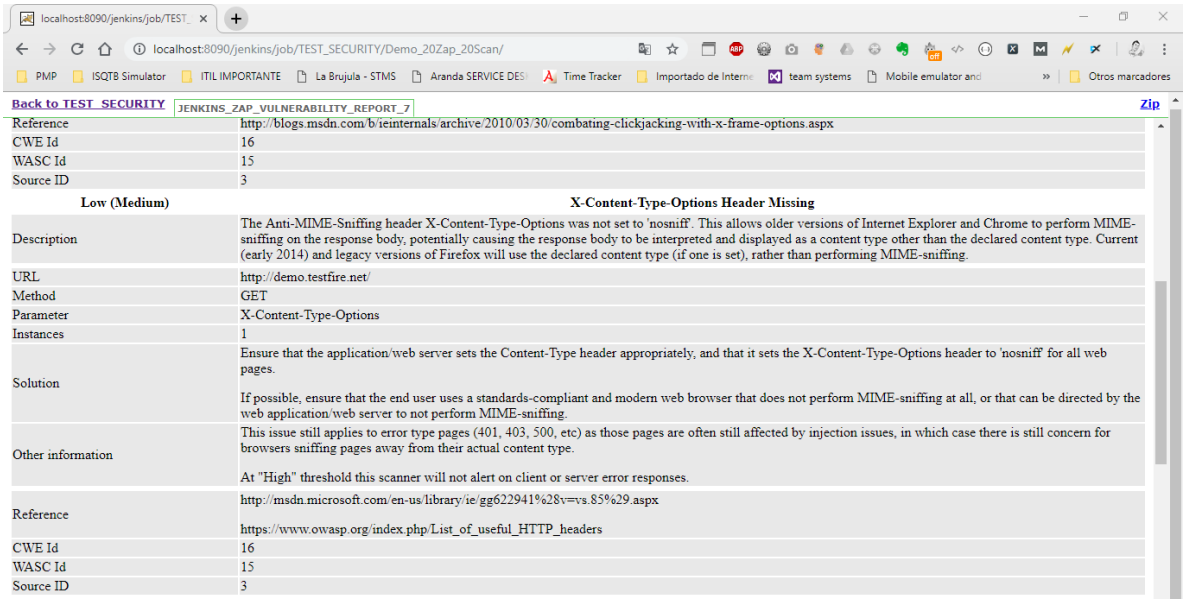
Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	1
Low	2
Informational	0

Alert Detail

Medium (Medium)	X-Frame-Options Header Not Set
Description	X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.
URL	http://demo.testfire.net/
Method	GET
Parameter	X-Frame-Options
Instances	1
Solution	Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Figura 41 Reporte de Vulnerabilidades



Back to TEST_SECURITY | JENKINS_ZAP_VULNERABILITY_REPORT_7 | Zip

Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	X-Content-Type-Options Header Missing
Description	The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
URL	http://demo.testfire.net/
Method	GET
Parameter	X-Content-Type-Options
Instances	1
Solution	Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
Other information	This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At "High" threshold this scanner will not alert on client or server error responses.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

Figura 42 Detalle del Reporte de Vulnerabilidades

9.8 ¿Cómo implementar un Pipeline en Jenkins?

Esta pregunta surge en el momento de implementar la seguridad dentro del CI/CD. Existen diversas maneras de implementar un pipeline. La herramienta Jenkins provee lo que se llama Jenkins Pipeline.

9.9 ¿Qué es Jenkins Pipeline?

Jenkins Pipeline es un grupo de eventos o Jobs interconectados entre sí que forman una secuencia de actividades. En otras palabras, Jenkins Pipeline es una combinación de plugins que soportan la integración e implementación de pipelines de entrega continua haciendo uso de Jenkins. Un pipeline tiene un servidor de automatización extensible que permite crear pipelines simples o complejos estos últimos haciendo uso de código, por medio de pipeline DSL (Lenguaje específico de dominio) [20].

9.10 ¿Cómo funcionan los Pipelines de entrega continua en Jenkins?

Un Pipeline en Jenkins es un grupo de eventos o Jobs que están interconectados unos con otros dentro de una secuencia.

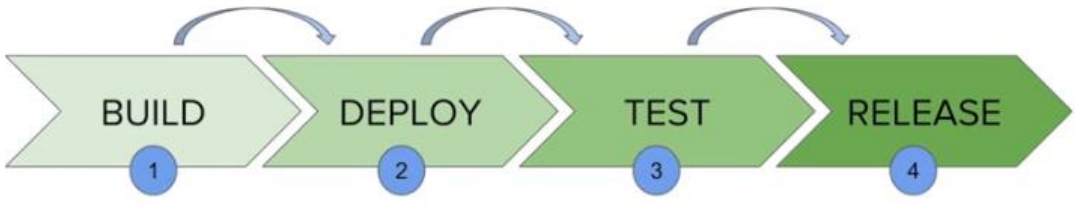


Figura 43 Pipeline Jenkins

[22]

La imagen representa un pipeline de entrega continua en Jenkins. Contiene un grupo de estados llamados Build, Deploy, Test y Release. Cada

Estado está relacionado con otro estado y cada uno tiene sus eventos, que funcionan en una secuencia llamada Delivery Pipeline.

Un Delivery Pipeline es una expresión automatizada que permite visualizar el proceso de obtención de software para el control de versiones. Esto significa que cada cambio realizado en el software pasa por una serie de procesos complejos en su camino al lanzamiento o released. También implica desarrollar software de manera confiable y repetible, que sea verificado a través de múltiples etapas de prueba y despliegue.

9.11 ¿Qué es un Jenkins file?

Los pipelines en Jenkins se pueden definir mediante el uso de un Jenkins File, un archivo de texto donde se almacena el código. Esto permite definir un pipeline haciendo uso de un lenguaje específico del dominio (DSL) escribiendo en código todos los stages.

9.12 Beneficios de usar Jenkins File

A continuación, se mencionan los principales beneficios de usar Jenkins File.

Es posible la creación de pipelines automáticos para todos los branches y ejecutar los pull request con solo un Jenkins File.

Es posible revisar el código dentro del Pipeline.

Se puede auditar el Jenkins Pipeline.

El Pipeline puede ser modificado por varios usuarios.

Un Jenkins File puede ser configurado mediante interfaz gráfica o por medio de código.

9.13 ¿Por qué se debe Implementar Jenkins Pipeline? [20]

Gracias a la implementación de Jenkins Pipeline es posible la creación de múltiples Jobs automatizados que ejecuten diferentes casos de usos y arranquen los pipelines.

Entre los beneficios obtenidos al usar Jenkins pipeline se mencionan los siguientes.

Jenkins Pipeline se implementa como un código que permite a múltiples usuarios editar y ejecutar procesos pipeline.

Los Pipelines son robustos. En caso que el servidor de CI/CD sufra un reinicio imprevisto, el pipeline pude reanudarse automáticamente.

Es posible pausar el proceso del Pipeline y hacer que espere para reanudarse hasta que haya cambios en el software.

Jenkins File soporta varios proyectos y puede correr múltiples Jobs al mismo tiempo e incluso usar pipelines en bucles.

9.14 Análisis práctico: Implementación de Jenkins Pipeline [22]

La intención de esta prueba de laboratorio es dar a conocer cómo se puede implementar un Jenkins Pipeline.

9.14.1 Descripción de hardware utilizado

Las descripciones técnicas del equipo en el cual se realizó la prueba de concepto son:

Sistema Operativo: Windows 10 Home Single Languaje
Procesador: AMD A10-5750M APU with Radeom(tm) HD Graphics 2.50 GHz
Memoria RAM: 8.0 GB

9.14.2 Descripción del Software utilizado

Jenkins war v 2.161
Build Plugin Pipeline 1.5.8

9.15 Configurando un Pipeline en Jenkins

Para esta prueba de laboratorio se decidió configurar un pipeline ejemplo, en donde se puede dar a conocer un stage, con sus diferentes steps.

En anteriores capítulos se había dado a conocer un pipeline con una mínima cantidad de steps que debería tener un ambiente de integración y entrega continua. Esta prueba de laboratorio estará basada en la implementación de este pipeline:

Figura 44 Pipeline Básico En un Ambiente CI/CD

Para el Aseguramiento de La Calidad

9.15.1 Primer paso: instalar Build Pipeline Plugin

La idea es empezar a construir el siguiente pipeline en Jenkins, para ello es necesario instalar el plugin “Build Pipeline”, con el fin de generar una vista de los pipelines que se vayan creando.

Es necesario dirigirse al menú Administrar Jenkins / Administrar Plugins como lo muestra la siguiente imagen.

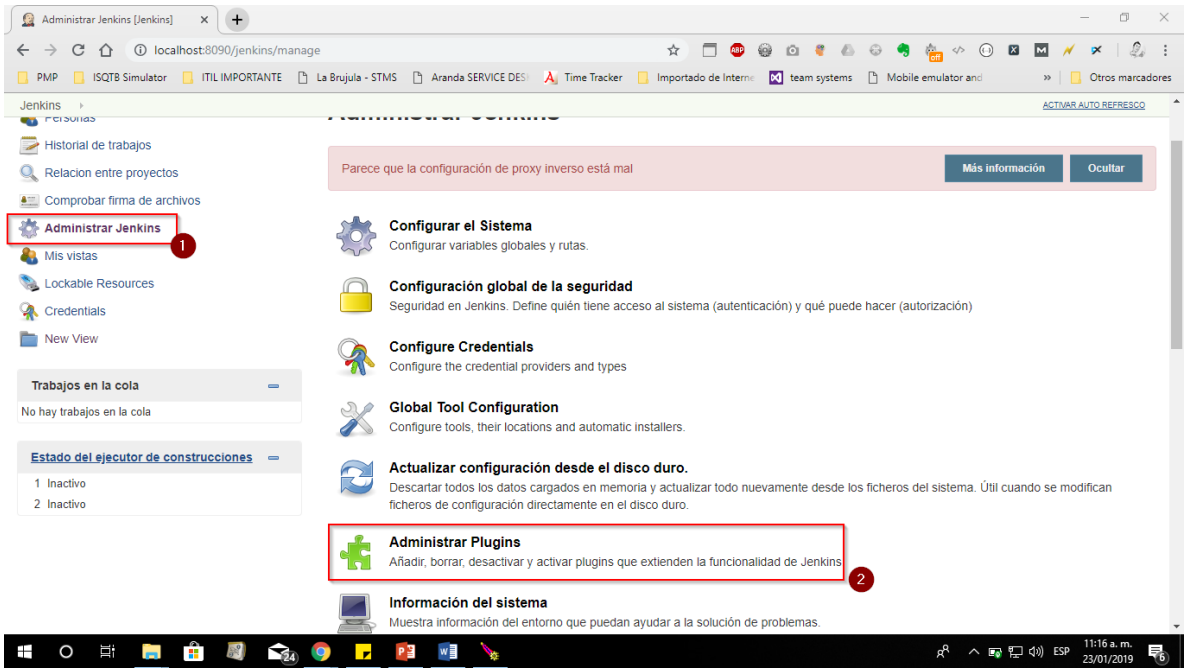


Figura 45 Administrador de Jenkins

Luego dentro de Administra plugins, es necesario filtrar por el plugin Build Pipeline Plugin.

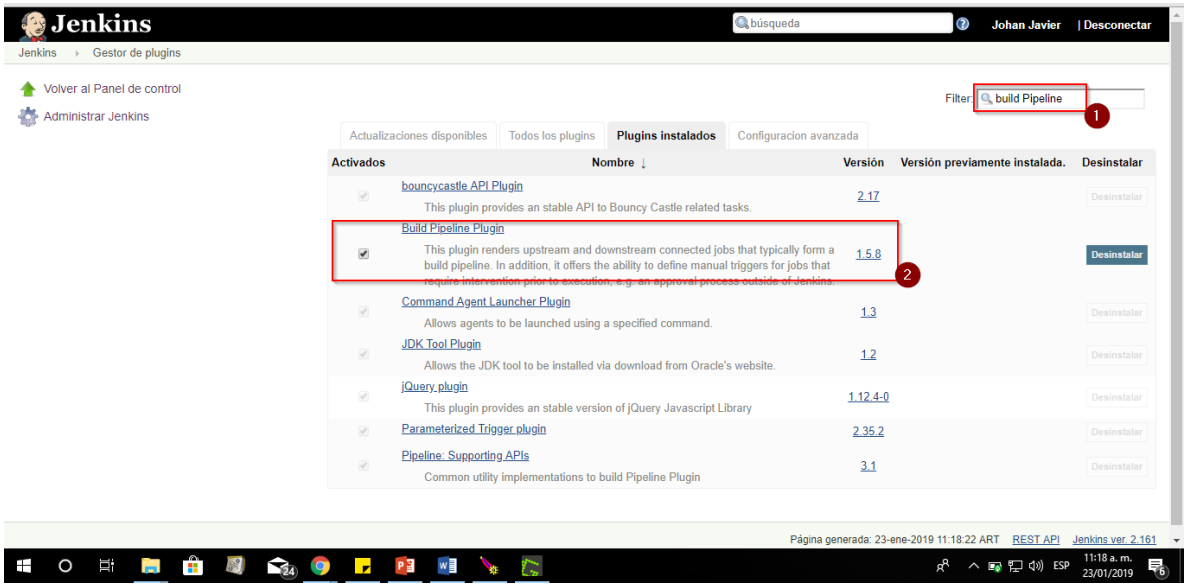


Figura 46 Build Pipeline Plugin

En caso de que no tener el plugin build pipeline previamente instalado, este debe aparecer en la pestaña “Todo los plugins”.

9.15.2 Segundo Paso: creación del Jenkins Pipeline

Para ello es necesario crear la vista Build Pipeline plugin. Esto se puede hacer ingresando una nueva vista como lo muestra la siguiente imagen.

Figura 47 Creación del Jenkins Pipeline

Posteriormente se debe ingresar el nombre que se colocará al Pipeline. Para esta prueba de laboratorio se lo designó “Pipeline_TesisMaestria” y se selecciona la opción “Build Pipeline View” como lo muestra la siguiente imagen.

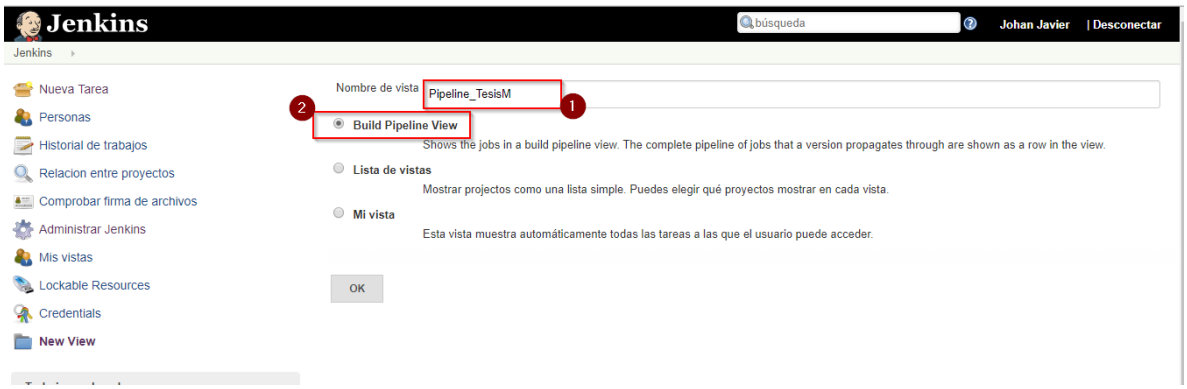


Figura 48 Creación de la Vista Pipeline

Lo siguiente que se debe hacer es empezar a crear los stages o Jobs que van a hacer parte del pipeline. Para ello es necesario de crear una nueva tarea mediante el menú “Nueva Tarea” como lo indica la siguiente imagen.

Figura 49 Creación de la Nueva Tarea

El primer Jobs que se genera permite crear los binarios de las aplicaciones que estemos integrando, para lo cual se pondrá como nombre de ejemplo “Sample Build” y posterior a ello se debe presionar la opción “Crear un Proyecto de Estilo Libre”

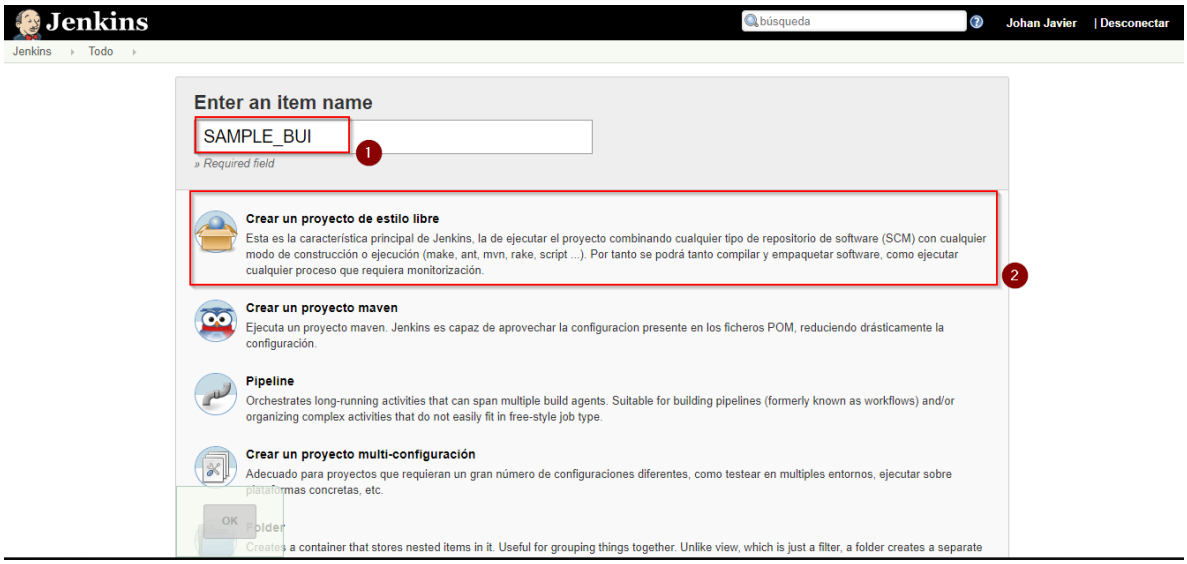


Figura 50 Nombre de la Nueva Tarea

Esto mismo se debe hacer con los demás stages o Jobs que conforman el pipeline como se muestran en la siguiente imagen.

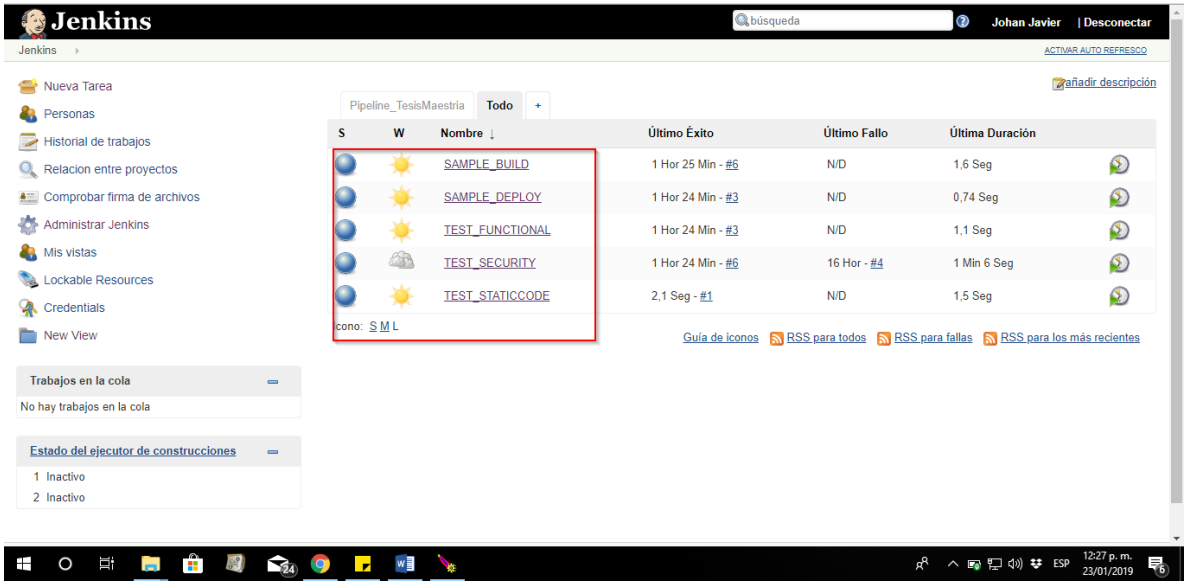


Figura 51 Jobs Creados Para La Ejecución del Pipeline

Una vez creados todos los Jobs que conformarán el pipeline, es necesario definir cuál será el statage o job inicial del pipeline, para lo cual se ingresa a la vista del pipeline y luego se accede a configurar.

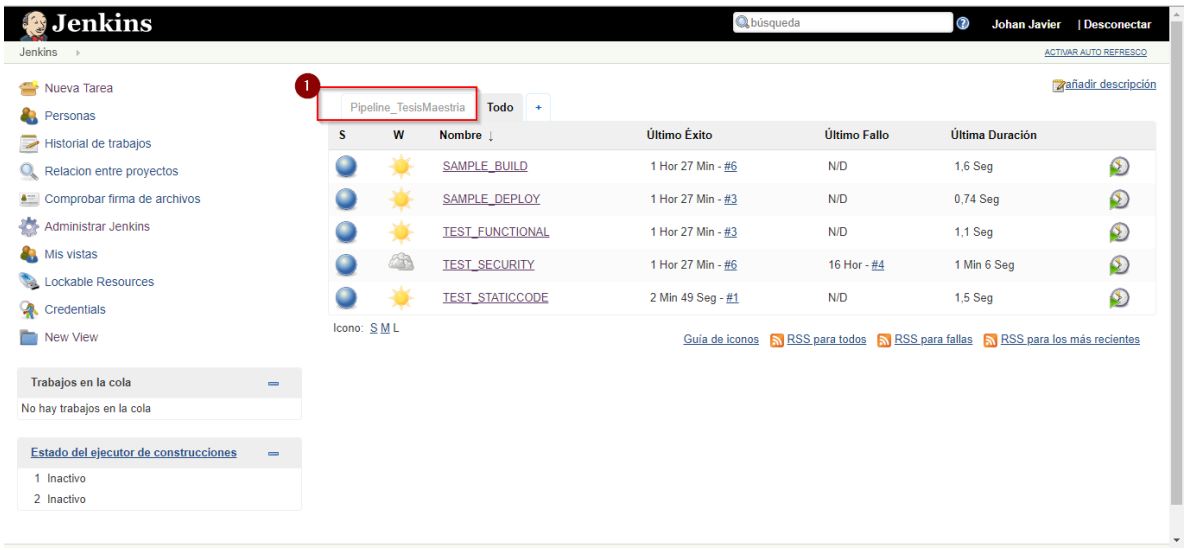


Figura 52 Vista del Pipeline

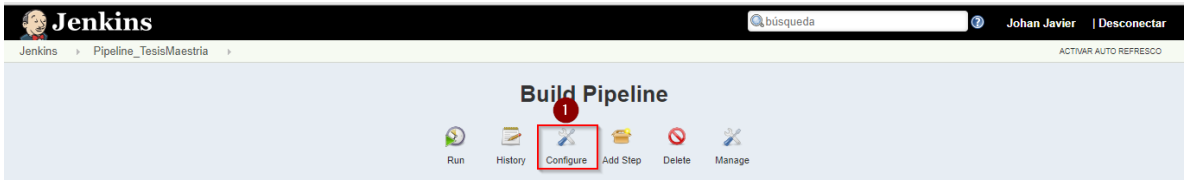


Figura 53 Herramientas Pipeline

Dentro de la configuración se define como el primer stage “Sample Build” que, como se mencionó anteriormente, es el primer job que creará los binarios, para definirlo como el primero se debe ingresar al apartado “Pipeline Flow” y allí seleccionarlo como el inicial, como muestra la siguiente imagen.

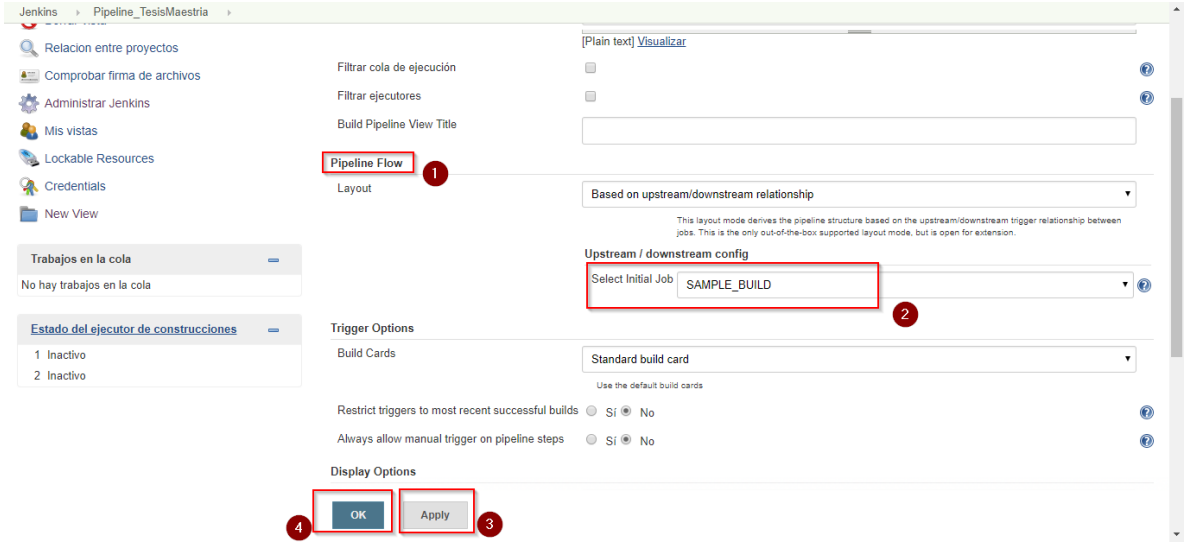


Figura 54 Configuración del Pipeline

Es así que el pipeline empieza a tener como primer stage el SAMPLE_BUILD, como lo muestra la siguiente imagen.

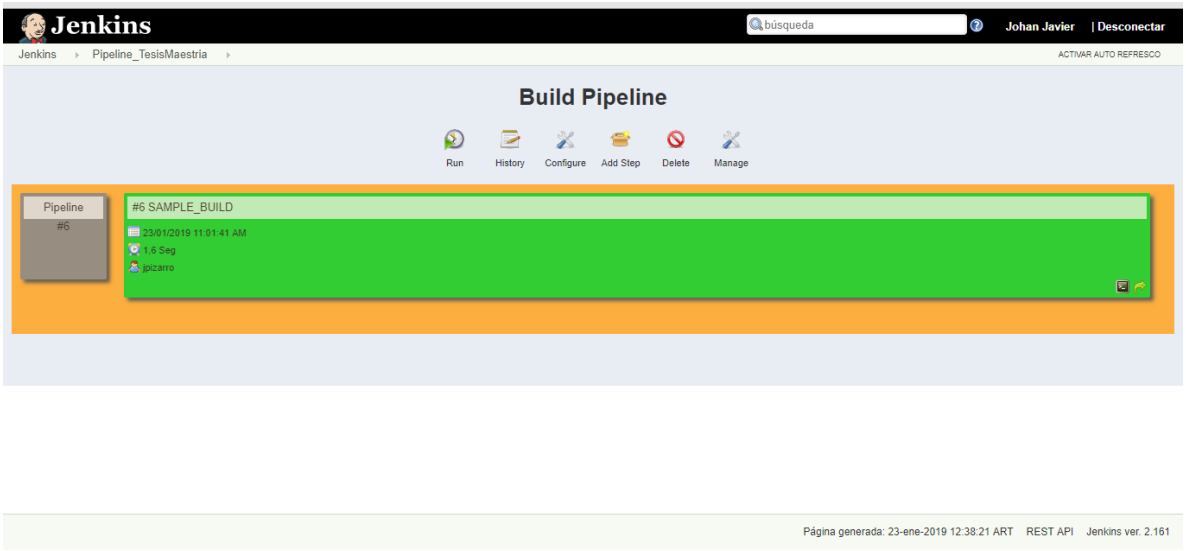


Figura 55 Primer Stage

Luego se continuará armando el pipeline de acuerdo a la imagen #. Es necesario ingresar al segundo stage llamado “TEST_STATICCODE”. Para ello accederemos a este job y en la opción configurar se deberá acceder al apartado “Disparadores de Ejecuciones” y allí seleccionar la opción “Construir tras otros proyectos”. En ese lugar se ingresa nuestro job padre, en este caso es SAMPLE_BUILD, que será nuestro stage antecesor en el pipeline.

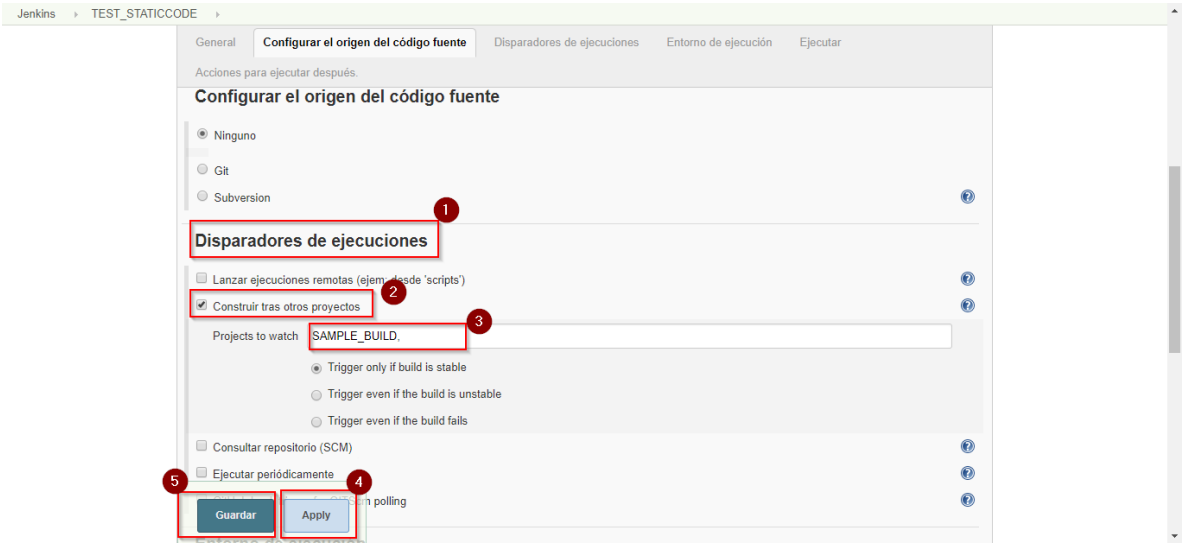


Figura 56 Configuración del Segundo Stage

En esta instancia, al ingresar a la vista del Build Pipeline, es posible apreciar cómo se van creando los stages, en este caso SAMPLE_BUILD y TEST_STATICCODE. Esto le indica a Jenkins que primero deberá crear los binarios y luego deberá hacer un análisis de código estático del proyecto.

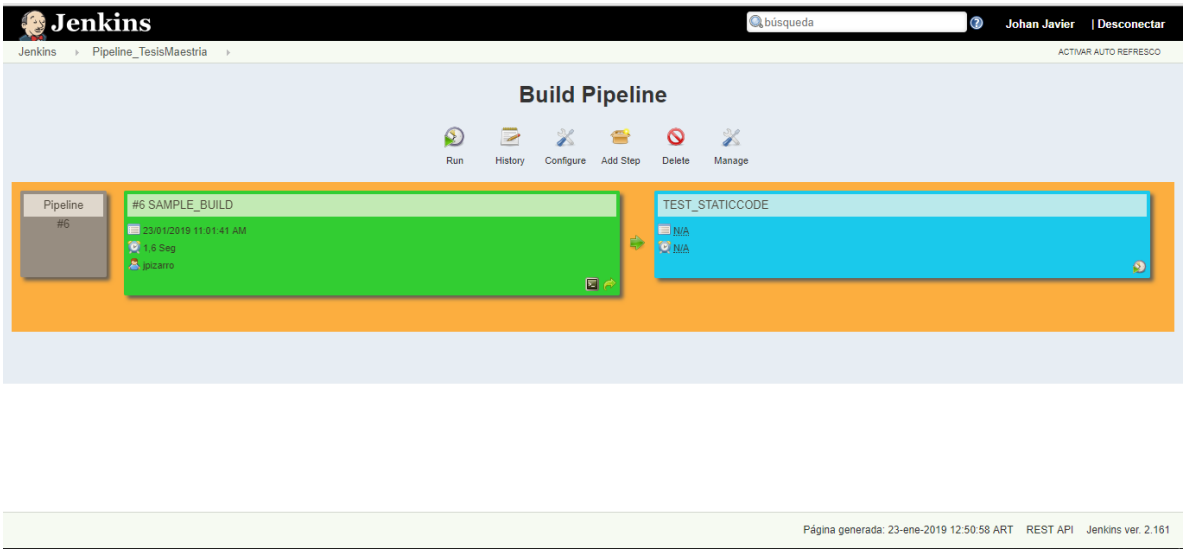


Figura 57 Vista del Segundo Stage en el Pipeline

Esto mismo se deberá hacer con los demás Jobs. Cabe recordar que en el anterior laboratorio se creó un job para la integración con Owasp Zap con el fin de realizar pruebas de seguridad sobre un ambiente CI/CD. En el

caso de este laboratorio se ha reutilizado y se le ha cambiado el nombre CI_DEMO_ZAP a TEST_SECURITY.

Una vez que se realicen las configuraciones a todos lo Jobs, es posible ver cómo queda armado todo el pipeline con el stages de pruebas de seguridad. Por lo tanto, una aplicación que se encuentre en desarrollo y sufre cambios, cada uno deberá pasar primero por los stages dentro del pipeline. En primer lugar, se deberá pasar por la creación de los binarios y una vez que se hayan creado, se realizan pruebas de análisis de código estático. Luego pasará por un conjunto de pruebas automatizadas funcionales, y más tarde por un conjunto de pruebas de seguridad automatizadas. Si todo sale bien, puede activar el job que permite deployar la aplicación en un ambiente productivo.

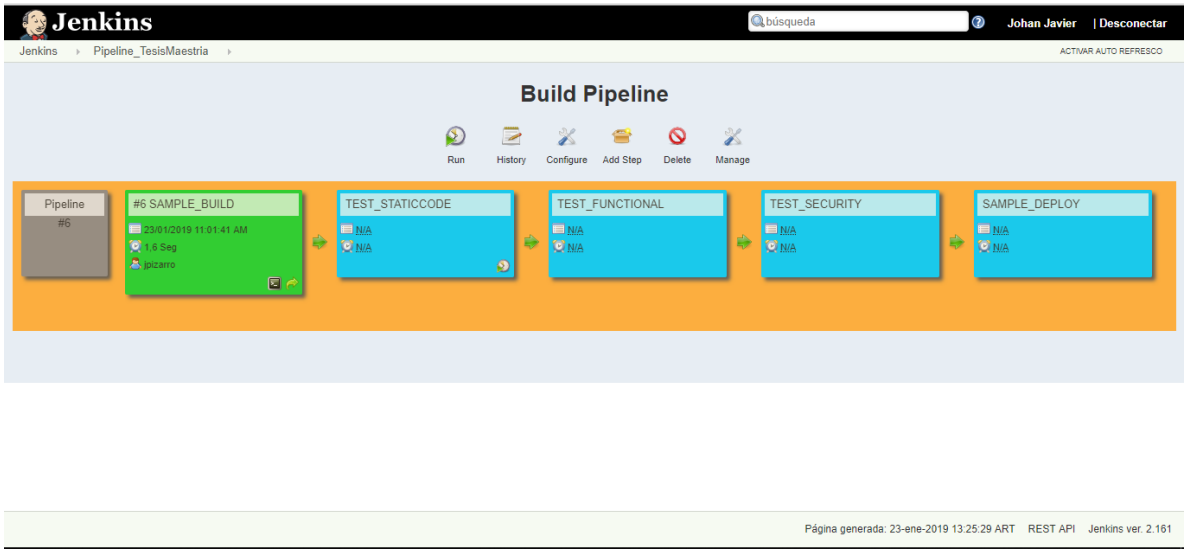


Figura 58 Vista Completa del Pipeline con Todos Sus Stages

De manera automática se puede configurar la corrida de los pipelines. También es posible de manera manual, presionando la opción “Run”. Se verá la corrida de cada uno de los stages y la consola. El color azul indica que el job se encuentra pendiente de ejecución, el color verde indica que el job se ha ejecutado de forma exitosa y el color rojo indica que el job tuvo inconvenientes al ejecutarse.

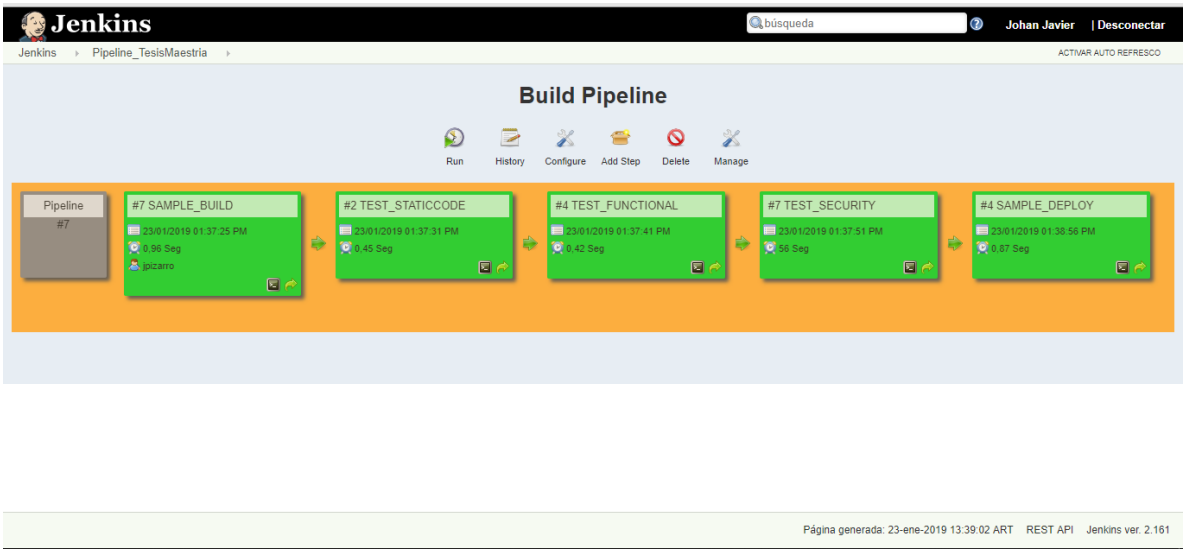


Figura 59 Ejecución Exitosa de todos los Stages En El Pipeline

Esta es una de las formas en que se puede crear un pipeline en Jenkins usando interfaz gráfica, las formas más complejas requieren crear un archivo Jenkins File con código.

Es así que un equipo de Dev y QA puede garantizar la seguridad dentro de un ambiente de CI/CD y cada miembro del equipo en especial el equipo QA puede estar al tanto de las detecciones de seguridad que pueden aparecer constantemente cada vez que el código es integrado y entregado. De esta manera es posible agregar SecDevOps en nuestro SDLC [23]

10 Capitulo: Automatizando la seguridad en el proceso de calidad de Software

10.1 Introducción

En el proceso de llevar a cabo la calidad de los sistemas, los miembros de QA se dedican a realizar en primer lugar un análisis de requerimientos. En segundo lugar, se encuentra la etapa de diseño de pruebas y posteriormente

la ejecución manual de esos casos de pruebas funcionales. Por último, el reporte de las vulnerabilidades encontradas.

Sigue la fase de llevar a cabo las pruebas de regresión que consiste en realizar pruebas con el fin de corroborar que las funcionalidades desarrolladas anteriormente aun sigan funcionando. Esta actividad, en lo posible, se trata de hacer de manera automatizada. Sin embargo, lo que actualmente se ha automatizado son características funcionales dejando de lado aspectos tan importantes como la seguridad. Por este motivo en este capítulo la intención es dar a conocer la manera en que las áreas de QA pueden aprovechar esta etapa de automatización funcional y complementarla con la automatización de la seguridad haciendo uso de tecnologías recomendadas por Owasp. Estas tecnologías permiten realizar este proceso de manera exitosa. En primer lugar, se llevará a cabo una prueba de concepto para dar a conocer cómo utilizar estas tecnologías, con el objetivo que se pueda aprovechar más la etapa de automatización, basado en el enfoque SecDevOps, y se tenga en cuenta la seguridad durante todo el ciclo de calidad del software.

10.2 Proceso de prueba funcional y seguridad propuesto

Este es un proceso simple de pruebas propuesto en esta tesis con el fin de involucrar a los equipos de QA y Dev en el aseguramiento de la calidad funcional y de seguridad. Como se ha mencionado, la seguridad no se tiene en cuenta en etapas tempranas del desarrollo de software. Una de las causas mencionadas es el desconocimiento que tienen los equipos de cómo implementar una seguridad proactiva durante este proceso. Para ello se ha propuesto un flujo basado en el efoque DevOps donde todos los perfiles colaboran unos con otros y no trabajan de manera aislada. Suele pasar en muchos proyectos actuales que los perfiles de seguridad trabajan aislados de los equipos de Dev y QA. Además, aplican seguridad de manera reactiva. Es decir que aplican seguridad ya cuando una aplicación esta en modo productivo, del lado del cliente. Entonces si en el proceso de desarrollo de una

aplicación no consideraron aspectos básicos de seguridad, lo más probable es que cuando la aplicación se encuentre en modo productivo estará expuesta a un sin número de fallas de seguridad.

Explicando un poco en que consiste el flujo, la idea es que los especialistas de seguridad solo si en caso de que una empresa considere necesario tenerlos, porque esta tarea tambien puede ser realizada solamente por los Devs y QA que comprendan que la seguridad es una obligación durante sus procesos y aprendan sobre ella, pero suponiendo que una compañía tiene a un experto en seguridad, esta persona estara encargada de brindar capacitación sobre las principales vulnerabilidades, tambien estara encargada de brindar asesoria y acompañamiento durante todo el ciclo de desarrollo de software.

Los miembros de los equipos de QA son las personas que deben entender más sobre seguridad, porque en este caso son ellos los encargados de garantizar que el producto tenga la menor probabilidad de fallos funcionales y de seguridad. Para ello deberán armar un buen plan de pruebas donde puedan abordar un gran porcentaje de cobertura funcional y de seguridad con base al diseño y desarrollo de casos de prueba que puedan ser ejecutados de manera manual y automatizada, de acuerdo a la tecnologia que más se adapte a las necesidades del proyecto. Actualmente en el mundo de la automatización de pruebas existen dos formas: el uso de BDD usando tecnologias como Cucumber y la tradicional forma de automatización de pruebas con Selenium para la generacion casos de pruebas funcionales. A ello se le debe adicionar la aplicación Owasp Zap que permite generar un escaneo por cada una de las pruebas automatizadas, lo que al final genera un reporte que se notifica al equipo de Devs por medio de una herramienta de gestor de incidentes con el fin de que puedan analizarlos y resolverlos.

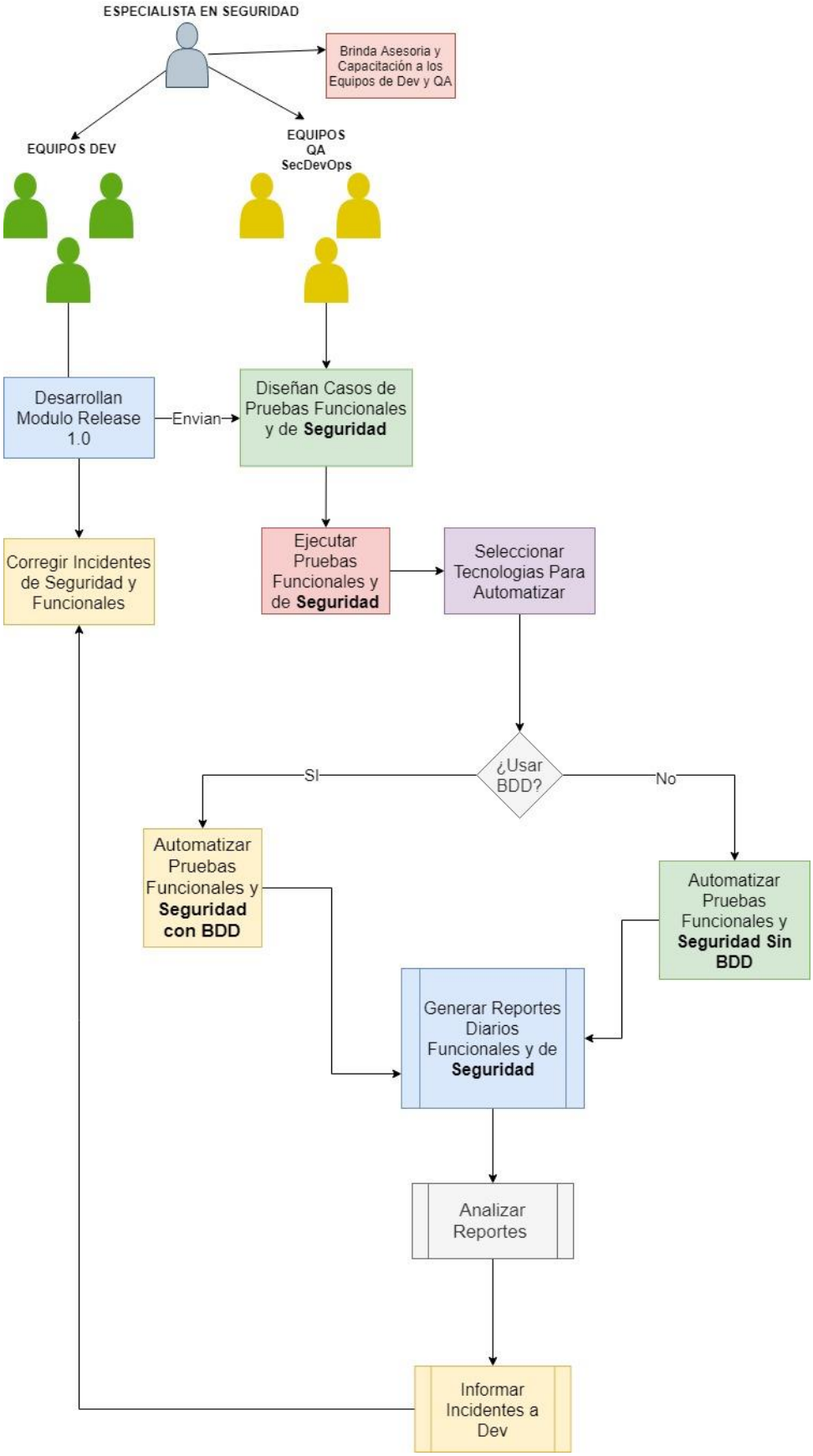


Figura 60 Diagrama de actividades propuesto para implementar SecDevOps

10.3 Pruebas de seguridad automatizadas basadas en BDD [23]

Las actuales herramientas utilizadas para realizar pruebas de seguridad sobre aplicaciones, por si solas son nada más que simples escáneres de seguridad que se dedican a encontrar vulnerabilidades. Sin embargo, estos escáneres no son capaces de encontrar errores funcionales de seguridad. En contraste con lo anterior, las pruebas funcionales están basadas en un concepto calificativo en el cual indican dos salidas si la prueba acierta o la prueba falla y detalla la evidencia del error. No obstante, los escáneres de seguridad solos tienen una respuesta en caso de encontrar una vulnerabilidad, pero en caso contrario no responden nada. Por este motivo, se necesitan siempre especialistas en seguridad dedicados a revisar los reportes generados con el fin de encontrar falsos positivos e indicar cuáles incidentes aplican y cuáles no. En general, estos criterios de análisis no quedan documentadas ni codificados y solo se quedan en las mentes de estos especialistas. Como solución a ello es necesario aplicar el desarrollo guiado por comportamiento, cuyas siglas son BDD.

El desarrollo guiado por comportamiento o en su acrónimo en inglés Behavior Driver Deveploment BDD, es un proceso que proviene de la evolución de la programación guiada por pruebas cuyas siglas son TDD[8]. Consiste en escribir pruebas antes del código, pero en vez de ser pruebas unitarias a nivel de código son pruebas a nivel del negocio cuyo objetivo es escribir requerimientos que hacen parte de las historias de usuario[9], haciendo uso de un lenguaje Gherkin. Tiene como principal objetivo involucrar al personal interesado técnico y no técnico ya que por su estructura es un lenguaje entendible por humanos y ordenadores.

[8] **TDD(Test Driven Development),** Es una técnica de diseño e implementación software incluida dentro de la metodología XP (Extreme Programing), permite obtener cobertura de pruebas altas, su principal objetivo es que el desarrollador diseñe pruebas unitarias antes de empezar a desarrollar el código.

[9] **Historias de usuario,** Son documentos utilizados dentro de las metodologías agiles que permiten la especificación de requisitos y pruebas de validación provenientes de las solicitudes de los clientes.

Como anteriormente se mencionó, en la etapa de pruebas de seguridad los requerimientos de seguridad, por lo general, están en las mentes de los especialistas del área, o en su defecto en algún documento escrito que puede ir perdiendo actualización, a medida que el desarrollo avanza. Es así como BDD entra como una solución a este problema ya que permite escribir estos requerimientos por medio del lenguaje Gherking de forma dinámica y a medida que el desarrollo avanza, es posible actualizar y modificar estos requerimientos. Esto tiene muchas ventajas. Entre ellas es la de integrar de forma efectiva los procesos de pruebas funcionales de QA con las pruebas de seguridad y tener una trazabilidad de los requerimientos.

Los equipos de QA con necesidad de integrar dentro de sus actividades la seguridad y empezar a encaminarlas a un enfoque SecDevOps, deberán entender que la automatización es primordial para apoyar este enfoque. Los requisitos de seguridad y los criterios de aceptación tendrán que ser expuestos y entendidos por el resto de los equipos. Además, estos requisitos estarán definidos en un solo lugar de tal manera que puedan ser probados bajo demanda en cualquier momento, o incluso de manera continua.

10.4 BDD Security Framework [24]

Para llevar a cabo la integración con las pruebas de seguridad, dentro de un equipo de QA, entre las herramientas open source más populares se encuentran BDD Security Framework, que utiliza el lenguaje Gherking, y las sintaxis Given, When, Then para la descripción de requerimientos. Gracias a que el framework es compatible con Cucumber y JBehave permite la ejecución de pruebas de aceptación automatizadas escritas en BDD, por otra parte el mismo es compatible con Selenium Web Driver y con Owasp Zap. Más adelante se darán a conocer aspectos más técnicos del uso de estas tecnologías y la explicación de su importancia al momento de automatizar la seguridad.

Como se puede apreciar en la siguiente imagen, la arquitectura del framework está compuesta por 3 capas. La primera contiene los BDD Security Stories que se comunican con la capa dos, Java Core, y esta se interconecta con la capa de Selenium Steps donde están localizados los componentes de la aplicación.

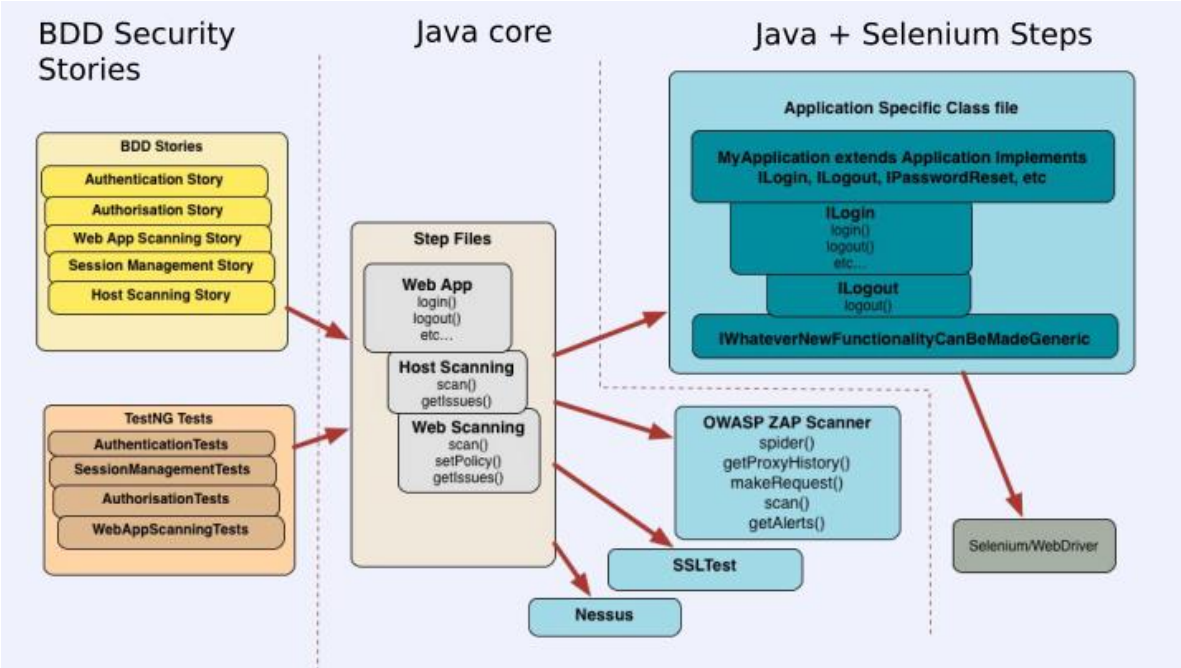


Figura 61 Arquitectura de BDD Security Framework

[25]

Otras características importantes que se pueden mencionar son que es un framework Open Source que ayuda actualizar DevOps a SecDevOps, se puede integrar al pipeline CI/CD, y permite generar reportes de seguridad para que fácilmente puedan ser vistos y entendidos por los involucrados en el negocio y personal de seguridad.

El framework automatiza las pruebas de seguridad manuales que se realizan con la aplicación Owasp Zap, de manera que hace uso de la API de ZAP para acceder a sus funciones.

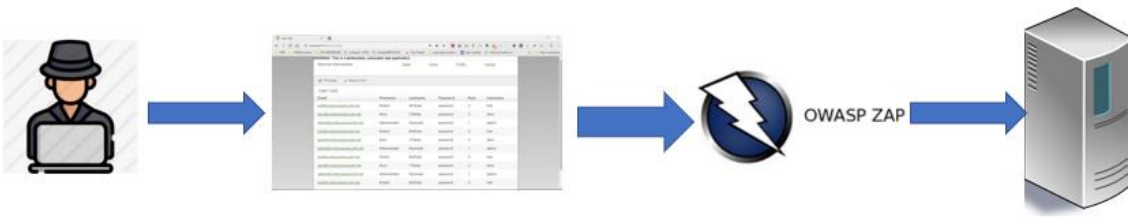


Figura 62 Pruebas Manuales de Seguridad con Owasp Zap

También hace uso de la API de Selenium Web Driver con el fin de automatizar las pruebas de seguridad y funcionales. De esta manera se controlan las posibles vulnerabilidades que la aplicación pueda tener.

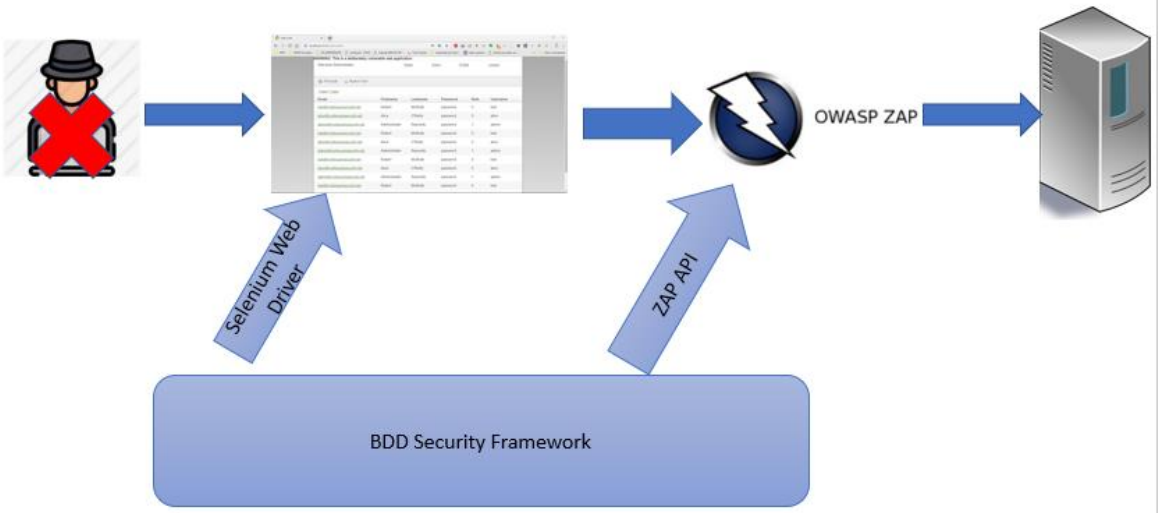


Figura 63 Pruebas de seguridad Automatizadas con Owasp Zap

10.5 Análisis práctico y pruebas de concepto para BDD Security Framework [24]

En este punto se describirá detalladamente los componentes de hardware y Software que serán utilizados para la realización de la prueba de concepto.

10.6 Descripción de hardware utilizado

Las descripciones técnicas del equipo en el cual se realizó la prueba de concepto son:

Sistema Operativo: Windows 10 Home Single Languaje
Procesador: AMD A10-5750M APU with Radeom(tm) HD Graphics 2.50 GHz
Memoria RAM: 8.0 GB

10.7 Descripción de software utilizado

Los siguientes programas fueron utilizados para poder instalar y hacer uso de BDD Security Framework:

Java Development Kit JDK Versión “1.8.0_121”, Este software permite proveer las herramientas necesarias para el desarrollo y creación de programas escritos en lenguaje Java. [26]

Eclipse IDE Versión “eclipse-jee-2018-09-win32-x86_64”, Es una plataforma de software compuesta por un conjunto de herramientas de programación de código abierto y multiplataforma. [27]

Git Bash Versión 2.19.0-64-bit Para Windows, consola para la gestión de versiones del código.

Git HuB. Es una forja para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de computadora. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, [28].

Gradle, es un sistema de automatización de construcción de código abierto que construye sobre los conceptos de Apache Ant y Apache Maven e

introduce un lenguaje específico del dominio basado en Groovy en vez de la forma XML utilizada por Apache Maven para declarar la configuración de proyecto [29].

Buildship Gradle Integration 2.0 es una herramienta que permite automatizar la construcción de proyectos similar a las tecnologías como Maven y Ant, pero a diferencia de estas dispone una gran flexibilidad que permite trabajar con diferentes lenguajes de programación.

Cucumber Eclipse Plugin, es una herramienta de software utilizada por los programadores de computadoras para probar otros programas. Ejecuta pruebas de aceptación automatizadas escritas en un estilo de desarrollo basado en el comportamiento. El enfoque central de Cucumber BDD es su analizador en lenguaje claro llamado Gherking. [30]

Gherking, es un lenguaje común que alguien sin conocimientos técnicos en programación puede fácilmente leer y entender, e igualmente puede aportar en la creación de escenarios. Estos escenarios son almacenados en archivos con extensión ".feature".

Un archivo ".feature" está especificado de la siguiente manera:

Feature: nombre de la funcionalidad que vamos a probar, el título de la prueba.
Scenario: habrá uno por cada prueba que se quiera especificar para esta funcionalidad.
Given: marca el contexto, las precondiciones.
When: especifica las acciones que se van a ejecutar.
Then: especifica el resultado esperado, las validaciones a realizar.

Cada archivo contempla diferentes escenarios de prueba. En la siguiente imagen se muestra cómo se vería implementado un escenario.

Código Fuente
Feature: Test Login MHO **@Login** **Scenario:** Login MHO **Given** Open browser and start MHO **When** I enter a valid "jpizarro" and valid "aaa" **Then** User Should be able to login succesfully

Tabla 3 Feature Login

Para este Escenario, la idea es validar el login de una aplicación ingresando unas credenciales correctas y al final validar que efectivamente el login es exitoso. Cabe mencionar que el lenguaje Gherking tiene diferentes idiomas, si bien el inglés es uno de los más utilizados también soporta el lenguaje español.

Selenium Web Driver, es la tecnología más popular utilizada para automatizar aplicaciones web y mobile. Esta herramienta tiene una API[10] muy sencilla de usar que permite mediante ciertos comandos interactuar con el navegador web de modo que se pueda simular el uso de una aplicación web por parte del usuario final [31].

Altoro Mutual es un Banco Online vulnerable publicado por IBM Corporation con la finalidad de demostrar la eficacia de diferentes herramientas de detección de vulnerabilidades de aplicaciones web y defecto del sitio [32].

RopeyTasks, es una simple aplicación web sencilla construida con las siguientes vulnerabilidades: Blind HQL injection, XSS, CSRF, Case Sensitive passwords, No SSL, Falta de indicadores HttpOnly y seguros en las cookies de sesión [33].

10.8 Configurando BDD Security Framework

En esta sección la intención es dar a conocer los comandos necesarios para poder configurar el framework. Para hacerlo es necesario contar con el programa git Bash que permitirá realizar una copia exacta de branch de master. En esta prueba de concepto el framework se encuentra ubicado en la siguiente dirección de https://github.com/continuumsecurity/bdd-security.git , por medio del comando “git clone” es posible realizar una copia del mismo.

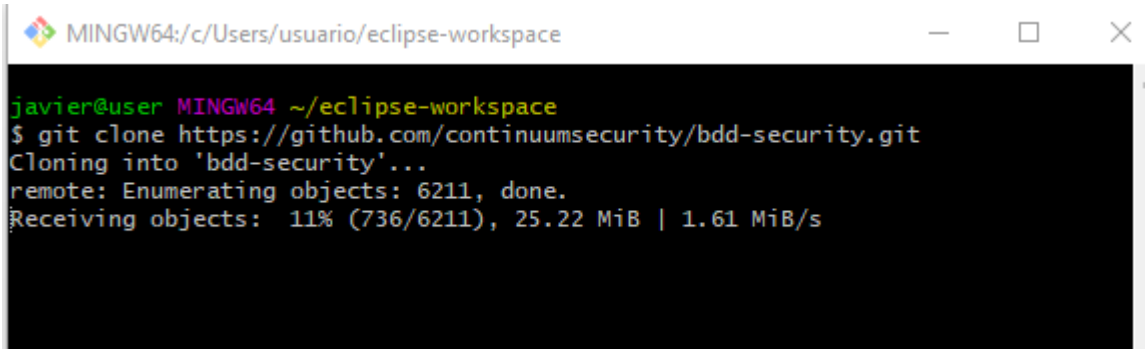


Figura 64 Descarga de BDD Security Framework

Para utilizar el framework es necesario tener instalado Eclipse IDE y el JDK. Como se detalla en la imagen anterior el framework fue alojado dentro del fichero denominado “eclipse-workspace”.

El siguiente paso es tener el Eclipse IDE abierto. En primera instancia eclipse no tendrá disponible el plugin Gradle por lo que es necesario instalarlo desde el eclipse Market Place que es el repositorio de plugins. Se encuentra ubicado accediendo al menú principal - menú Help - Eclipse MarketPlace y allí buscar gradle.

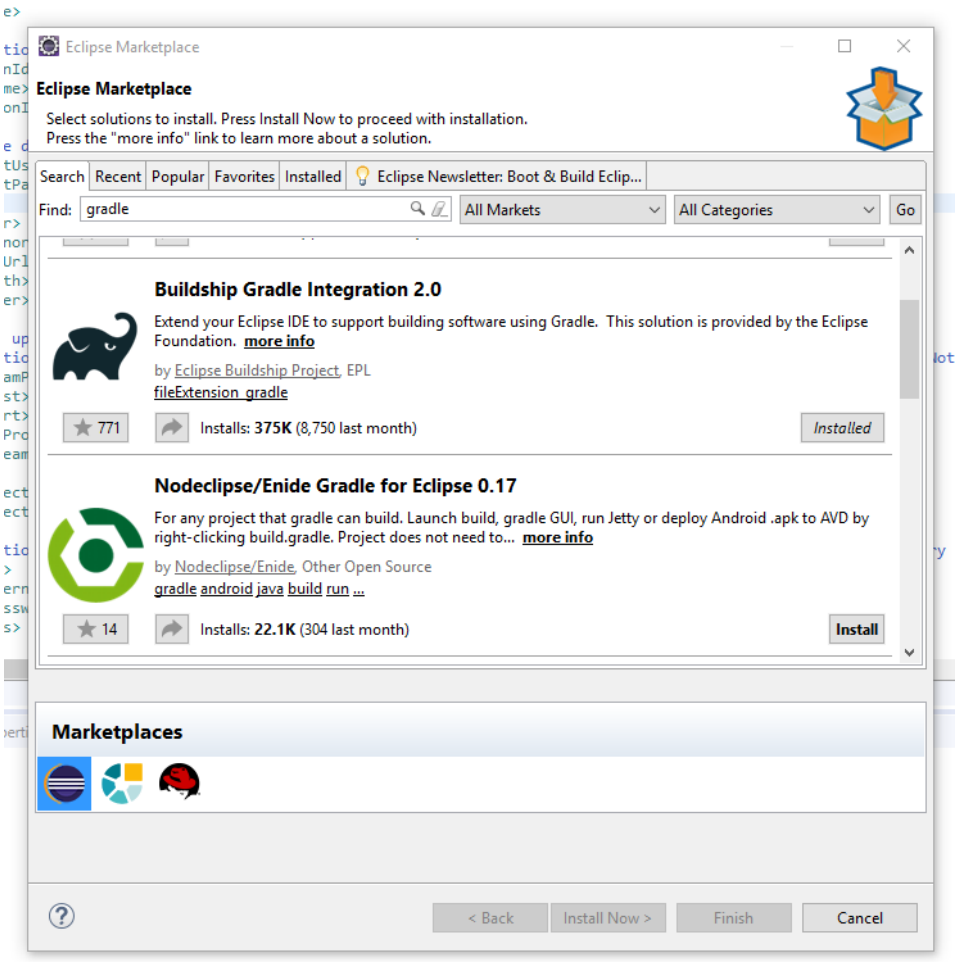


Figura 65 Instalación de Gradle en Eclipse IDE

Una vez instalado gradle es posible realizar la importación de BDD Security Framework. Es necesario volver a la barra principal - menú File – Import y buscar “Existing Gradle Project”.

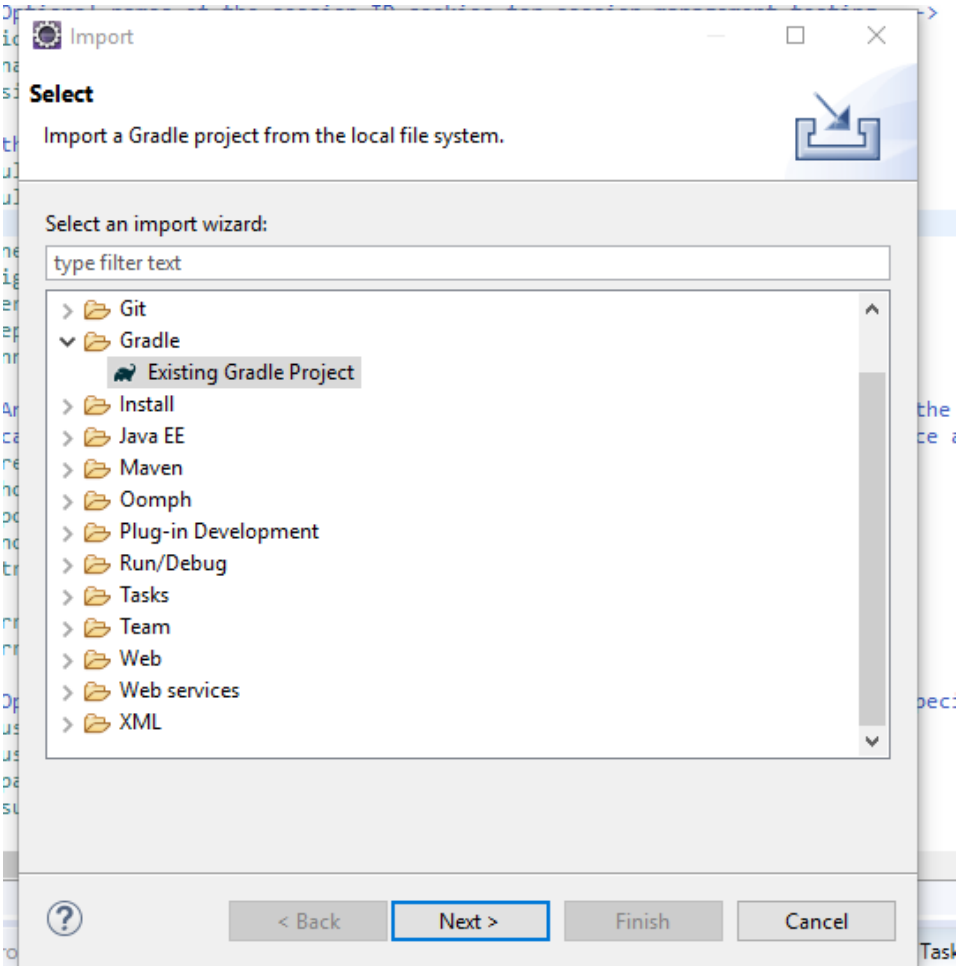


Figura 66 Creación de Proyecto Gradle en Eclipse IDE

Al final, se selecciona el patch donde se encuentra almacenado el framework y se selecciona finish.

Una vez que se importó el framework, se realiza la instalación de Cucumber, con el fin de poder crear nuestros escenarios BDD alojados en archivos .feature. Se realiza accediendo nuevamente a menú Help - Eclipse MarketPlace y se busca Natural Actual Versión.

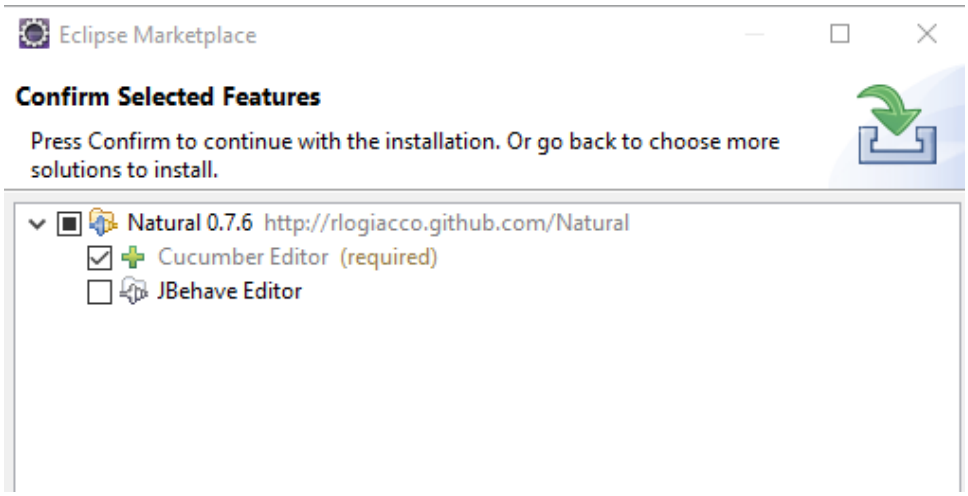


Figura 67 Instalación de Cucumber en Eclipse IDE

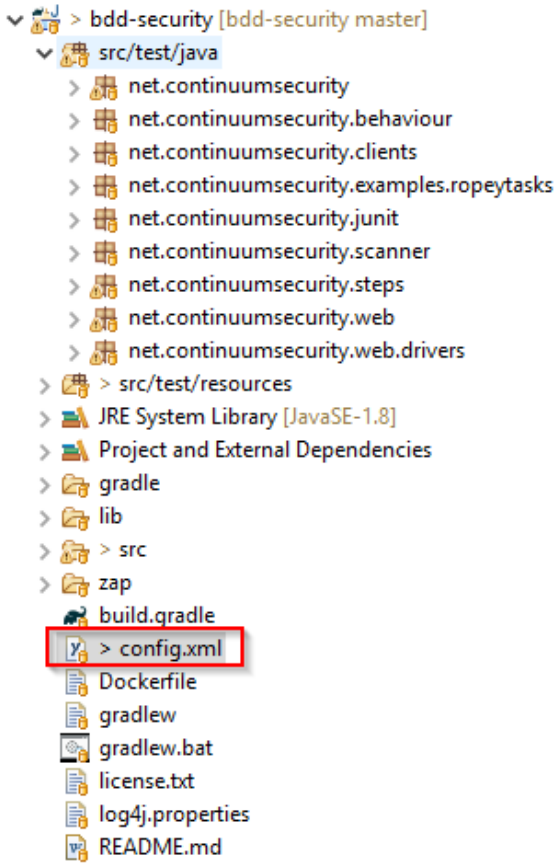


Figura 68 Archivo Config.xml

Una vez que se han instalado estos plugin es necesario configurar el archivo “config.xml” Este archivo aloja toda la configuración del framework

referente a las clases implementadas y credenciales utilizadas en los archivos ".features"

Para esta prueba de concepto es necesario definir dentro del archivo "config.xml" la url en la cual se van a realizar las pruebas de seguridad automatizadas. Por otra parte, se debe definir la ruta del driver que se utiliza. Esto significa que por cada explorador Firefox, Chrome o IExplorer se necesita indicar su ruta, según del browser en que se ejecutarán las pruebas.

Código Fuente:

Config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app>
		<!-- ARCHIVO CONFIG PARA EL BDD SECURITY FRAMEWORK -->
	<!-- Este archivo tiene las configuraciones necesarias para poder utilizar
		el BDD SECURITY FRAMEWORK durante el desarrollo de la Tesis de Maestría -->
		<!-- Configurado por Johan Javier Pizarro Martinez -->

<!—Esta línea de código sirve para indicar el tipo de webdriver que usaremos en este caso el webdriver es para el navegador Chrome -->
<defaultDriver
	path="src/test/resources/drivers/chromedriver.exe">Chrome</defaultDriver>

<!—Aquí se debe ingresar la URL Base de la Aplicación a Probar -->
<baseUrl>http://localhost:8080/</baseUrl>

<!—Aquí viene el nombre del paquete de aquellas clases que necesitemos usar Selenium para la Automatización (Identificación de Locators en el DOM de la Aplicación)
-->
```

```
<class>net.continuumsecurity.examples.ropeytasks.RopeyTasksApplication</class>

<!—Nombres opcionales de las cookies de ID
 de Sesión para la prueba-->
        <sessionIds>
                <name>JSESSIONID</name>
        </sessionIds>

<!—Aquí se deben ingresar el usuario y password
que necesitara la aplicación para la prueba -->
        <defaultUsername>admin</defaultUsername>
        <defaultPassword>password</defaultPassword>

</web-app>

<!—Aquí se debe ingresar el puerto de escucha y la IP
para correr las pruebas haciendo uso de la instancia OWASP ZAP -->
    <proxy>
      <host>127.0.0.1</host>
      <port>8888</port>
      <api></api>
   </proxy>
```

Tabla 4 Código Fuente del Config.xml

Por otro lado, es necesario indicar en el archivo de configuración aquellas clases creadas donde se encuentren los locators que se han identificado para generar la automatización.

10.9 Creación de scripts para automatización de la seguridad

Agregar seguridad a las pruebas automatizadas es posibles gracias a BDD Security Framework. En esta sección se dará a conocer la forma en que se puede automatizar y al mismo tiempo realizar las pruebas de seguridad.

10.10 Instalando aplicación vulnerable

Antes de empezar a utilizar BDD Security Framework es necesario conocer a la aplicación que vamos automatizar las pruebas funcionales y al mismo tiempo las de seguridad. Por ello para esta prueba de concepto fue necesario contar con una aplicación vulnerable denominada Ropeytasks que es una sencilla aplicación web vulnerable utilizada para pruebas de seguridad, entre las vulnerabilidades que incluye la aplicación se encuentran:

Blind HQL Injection
XSS
CSRF
Case insensitive passwords
No SSL
Lack of HttpOnly and secure flags on session cookies

Ropeytasks se puede encontrar en la siguiente dirección [33] y haciendo uso de git bash y por medio del comando “git clone ” es posible realizar una copia de este.

```
$ git clone https://github.com/continuumsecurity/RopeyTasks.git
Cloning into 'RopeyTasks'...
remote: Enumerating objects: 369, done.
remote: Total 369 (delta 0), reused 0 (delta 0), pack-reused 369
Receiving objects: 100% (369/369), 122.34 MiB | 5.18 MiB/s, done.
Resolving deltas: 100% (147/147), done.
```

Figura 69 Descarga de RopeyTasks

Es necesario ejecutar su .jar con el nombre “ropeytasks.jar” con ayuda del siguiente comando “java -jar ropetask.jar”

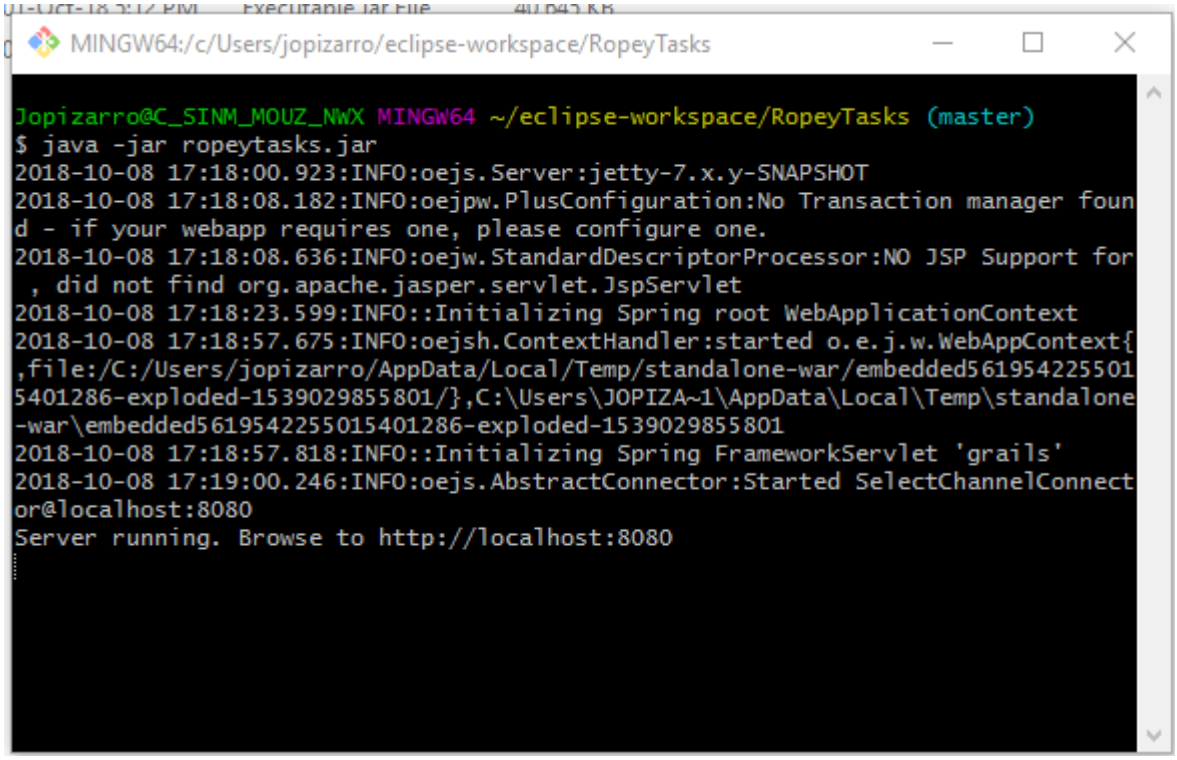


Figura 70 Instalación de RopeyTasks

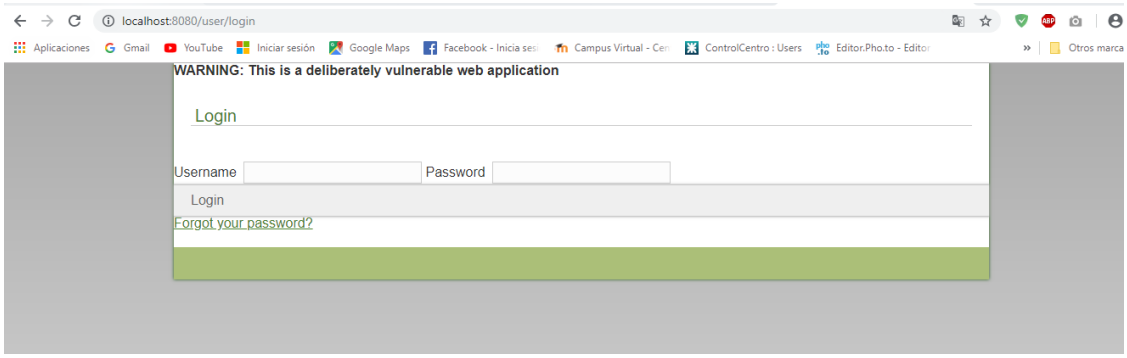


Figura 71 GUI de RopeyTasks

Por medio de esta aplicación se llevará a cabo las pruebas de seguridad automatizadas que estarán como Escenarios de prueba BDD.

10.11 Escenarios BDD security

Como en anteriores secciones se ha explicado que hay diferentes formas de llevar a cabo las pruebas automatizadas que incluyen pruebas funcionales y de seguridad y una de ellas es hacer uso del enfoque de pruebas BDD. BDD Security Framewok tiene prediseñadas un conjunto de pruebas de seguridad haciendo uso del lenguaje Gherkin. Estos escenarios se encuentran alojados dentro del proyecto en la ruta /bdd-security/src/test/resources/features, como lo indica la siguiente imagen:

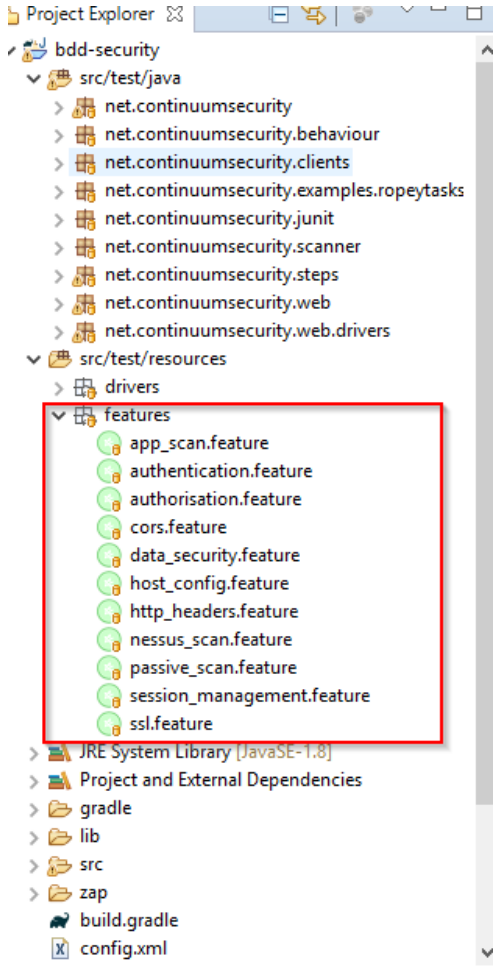


Figura 72 Features Prediseñados de BDD Security Framework

Los escenarios están basados en las pruebas de seguridad que como mínimo deben tenerse en cuenta en el momento de revisar la seguridad en

una aplicación. Estos escenarios están basados en las fases de pruebas de seguridad de aplicaciones.

10.12 Fases de pruebas de seguridad en aplicaciones

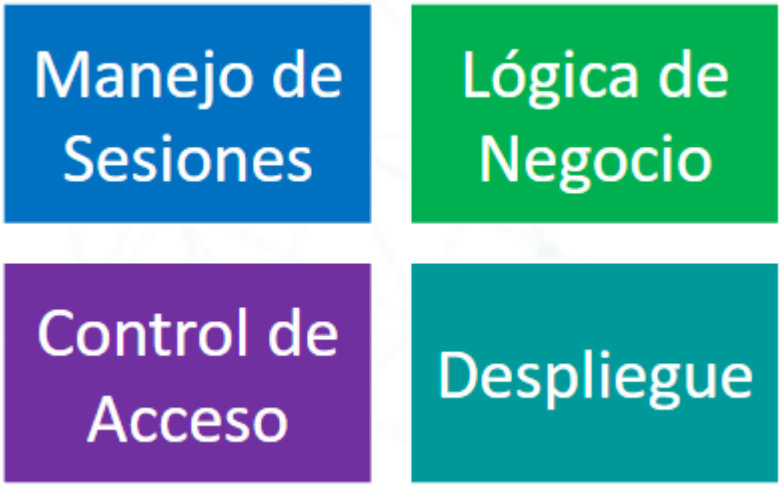


Figura 73 Fases de pruebas de Seguridad En Aplicaciones

Estas fases de pruebas están basadas en las buenas prácticas expuestas por Owasp con el propósito de generar técnicas que permitan en el momento del testing garantizar un anillo más de seguridad a las aplicaciones. En primera instancia estas pruebas pueden ser realizadas a nivel manual por el QA y posteriormente generar scripts que permitan automatizar estas fases de prueba siempre teniendo en cuenta que el enfoque DevOps indica que todo este automatizado.

Siguiendo con la descripción del framework, este da a conocer algunos escenarios prediseñados. Por ejemplo, en el proceso de automatización de pruebas funcionales por lo general el QA lleva a cabo la construcción de scripts que permiten automatizar las pruebas funcionales para el login. Sin embargo, solo se llevan a cabo pruebas funcionales, pero no de seguridad y para llevar a cabo este tipo de pruebas es posible hacer uso del archivo

“authetication.feature” en el cual están escritas los diferentes escenarios que permiten automatizar las pruebas de seguridad en el login y cabe resaltar la gran ventaja de usar un lenguaje entendible para cualquier persona no técnica.

La idea de la creación de un archivo.feature está basado en el proceso ágil donde por cada funcionalidad se define un feature, es decir, una característica que debe tener esa aplicación. Lo mismo sucede con la seguridad donde se definen una cantidad de features y por cada una se consideran diferentes escenarios de pruebas **“Scenario”.**

En estas fases el objetivo es identificar errores de seguridad y funcional de las aplicaciones. Para dar ejemplos de pruebas de seguridad en esta fase se hace uso de la aplicación Ropeytasks, para lo cual se definen diferentes casos de prueba. Estas pruebas son llevadas en primera instancia a nivel manual y luego automatizadas con ayuda de BDD Security Framework.

10.13 Manejo de sesiones

Se refiere a la fase de pruebas que se encarga de verificar la seguridad referente a:

- Inicio de Sesión
- Gestión de Contraseñas
- Actualización de datos
- Gestión de la Sesión

A continuación, se da a conocer algunos escenarios de pruebas que permiten describir y ejecutar de forma manual este tipo de pruebas.

10.14 Pruebas manuales

A continuación, se proponen para esta tesis algunos ejemplos de casos de prueba de seguridad que un equipo de QA debe tener en cuenta en proceso de pruebas de calidad de una aplicación:

CP_ID: CA01 **Caso De Prueba :** Validar Mensajes de Sesión
Pasos - Ingresar a la aplicación Ropeytasks - Ingresar al Login - Ingresar un usuario y/o password inválido
Verificar que: La sesión debe evitar la aparición de este tipo de mensaje: "Login for User failed: invalid password" "Login failed, invalid user ID" "Login failed; account disabled" "Login failed; this user is not active"

Tabla 5 Caso de Prueba Para Validar Mensajes de Sesión

10.15 Ejecución de caso de prueba CA01

Para la ejecución, se han tomado los siguientes datos de prueba:

Datos de Prueba 1: Se ha ingresado un usuario inválido con password correcta:

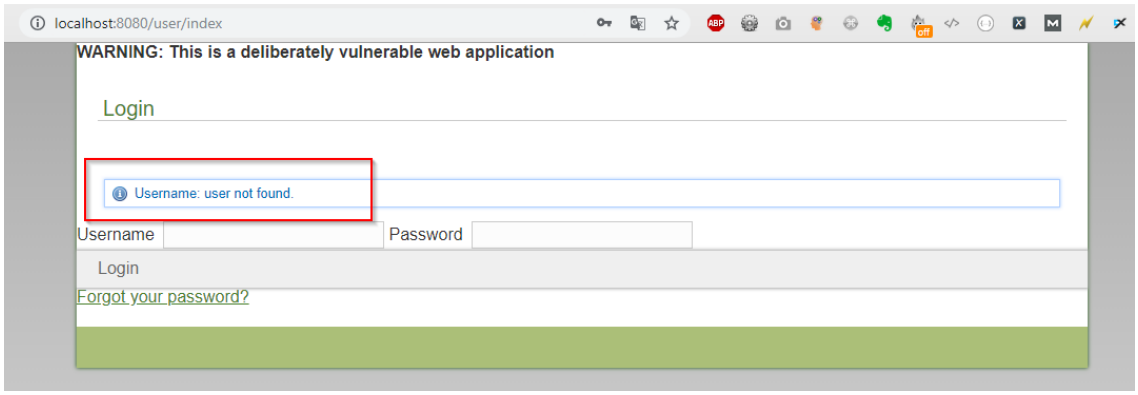


Figura 74 Validación de Usuario Inválido

Datos de Prueba 2: Se ha ingresado un usuario válido con password incorrecto:

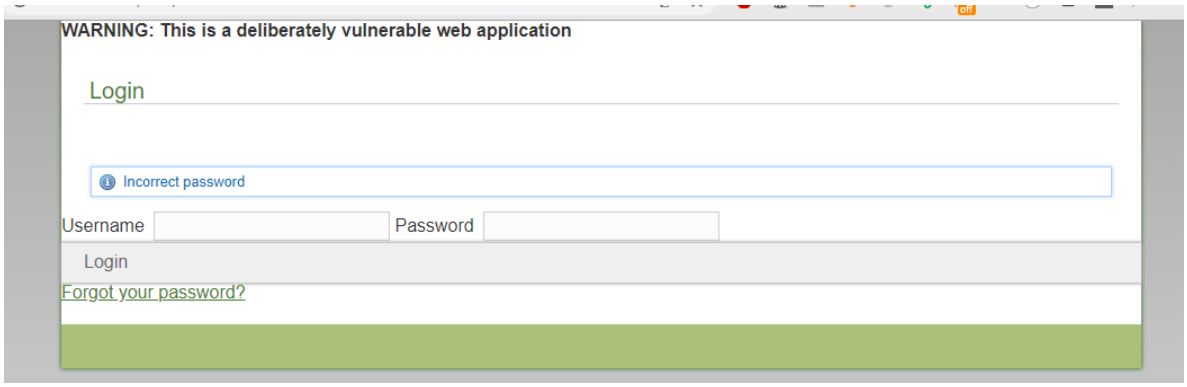


Figura 75 Validación Con Password Incorrecto

CP_ID: CA01 **Incidencia Encontrada**
La aplicación está dando a conocer, tanto para el usuario y password cuando uno de los dos es incorrecto, esto es un error de la aplicación a nivel de seguridad dado que un intruso puede llegar adivinar tanto el usuario y password aprovechando que el sistema indica cuando una de las dos credenciales es incorrecta.

Tabla 6 Incidencia Encontrada Caso de Prueba

10.16 Pruebas Automatizadas Para CP_ID:CA01

BDD Security Framework se basa en las buenas prácticas de seguridad para la fase de gestión de sesiones, una vez se hayan realizado pruebas manuales poderlas automatizar para garantizar la seguridad en las pruebas de regresión.

Alguno de los escenarios se puede detallar y están alojado en archivo “session_managment.feature”

authetication.feature
Codigo Fuente:
@iriusrisk-cwe-664-fixation **Scenario:** Issue a new session ID after authentication Given a new browser or client instance And the login page And the value of the session ID is noted When the default user logs in And the user is logged in Then the value of the session cookie issued after authentication should be different from that of the previously noted session ID

Tabla 7 Código Fuente Para El Feauture Para La Autenticación

authetication.feature
Codigo Fuente:
@session_management **Feature:** Session Management Verify that there are no weaknesses in the session management implementation @iriusrisk-cwe-613 @skip **Scenario:** Invalidate the session after a period of inactivity Given a new browser or client instance When the default user logs in Then the user is logged in When the session is inactive for 15 minutes Then the user is not logged in

Tabla 8 Código fuente para el feature Administración de Sesión

En este código se puede apreciar que corresponde al Feature verificar que no existan debilidades en la implementación de la gestión de sesión. Para ello da a conocer Escenario, entre ellos está el de verificar si la aplicación Invalida la Sesión después de un periodo de inactividad.

authetication.feature
Codigo Fuente:
@iriusrisk-cwe-613-logout **Scenario:** Invalidate the session when the user logs out Given a new browser or client instance When the default user logs in Then the user is logged in When the user logs out Then the user is not logged in

Tabla 9 Código Fuente Para La Sesiones Invalidas

10.17 Control de Acceso

Se refiere a todas aquellos tipos de pruebas de seguridad que comprueban qué roles pueden acceder o no a un sistema.

10.18 Pruebas Manuales

A continuación, se darán a conocer algunos casos de pruebas de seguridad manuales, con su respectiva ejecución.

CP_ID: CA02 **Caso de Prueba:** Evitar credenciales por parámetro en texto plano.
Pasos - Ingresar a la aplicación Ropeytasks - Ingresar al Login - Ingresar un usuario y/o password validos
Verificar que: Las credenciales suministradas no viajen en texto plano.

Tabla 10 Caso de Prueba Evitar Credenciales por Parámetros en Texto Plano

10.19 Ejecución de prueba

Se procede con la ejecución de la prueba donde se suministra como un usuario válido y password válido:

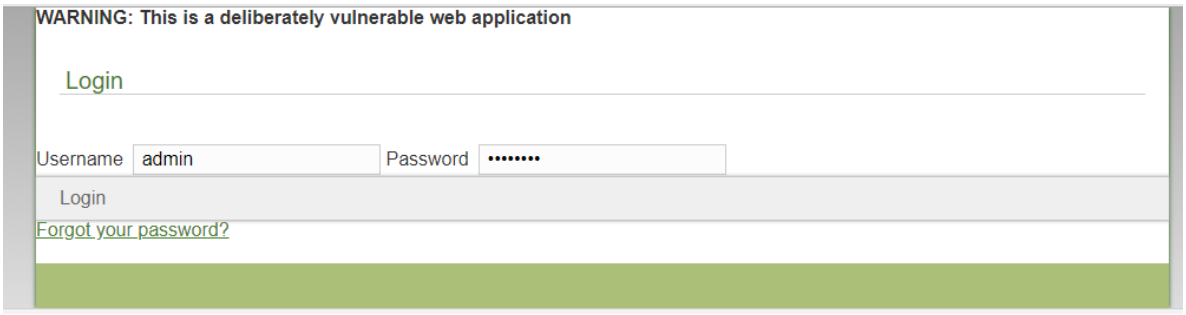


Figura 76 Ingreso Usuario y Password Valido

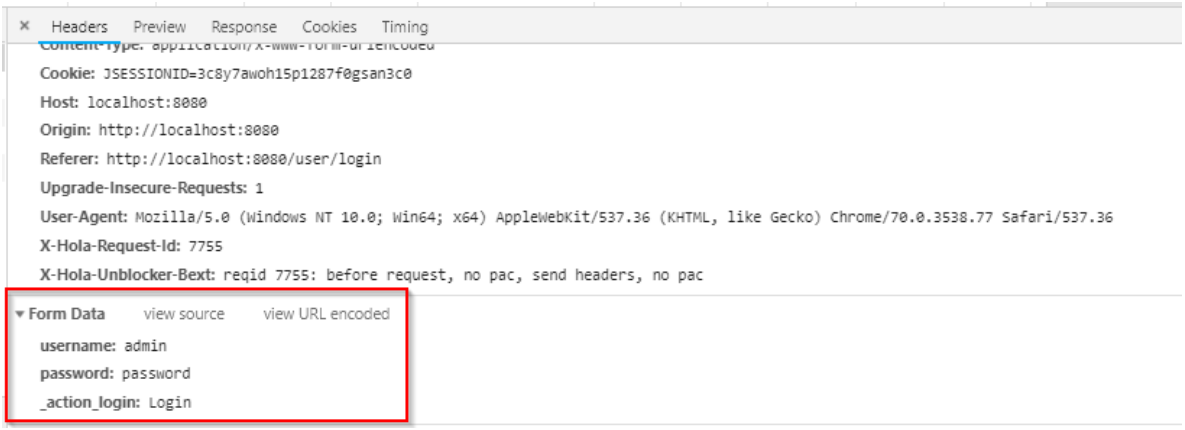


Figura 77 Usuario y Password Viaja en Texto Plano

CP_ID: CA02

Incidencia

Se evidencia cómo la información sensible viaja a través de la API de manera insegura dando a conocer el usuario y password utilizado en el momento de la autenticación.

Tabla 11 Evidencia del Caso de Prueba CA02

10.20 Pruebas Automatizadas Para CP_ID: CA02

Cabe recordar que el lenguaje Gherking se define por lo general respetando sus sentencias básicas:

Feature: nombre de la funcionalidad que vamos a probar, el título de la prueba.
Scenario: habrá uno por cada prueba que se quiera especificar para esta funcionalidad.
Given: acá se marca el contexto, las precondiciones.
When: se especifican las acciones que se van a ejecutar.
Then: y acá se especifica el resultado esperado, las validaciones a realizar.

Para dar a conocer cómo está definido alguno de los escenarios, es necesario ingresar al “authetication.feature”.

authetication.feature
Codigo Fuente:

```
@authentication
Feature: Authentication
  Verify that the authentication system is robust

  @iriusrisk-cwe-178-auth
  Scenario: Passwords should be case sensitive
    Given a new browser or client instance
    When the default user logs in
    Then the user is logged in
    When the case of the password is changed
    And the authentication tokens on the client are deleted
    And the login page is displayed
    And the user logs in
    Then the user is not logged in
```

Tabla 12 Feature Que Verifica la Robustes del Sistema

Este Feature consiste en verificar que la Autenticación del sistema sea robusta. Uno de los primeros escenarios es probar que el campo donde se ingresa el password debe ser case sensitive, es decir si un password fue escrito con solo letras minúsculas y posteriormente el mismo usuario ingresa ese password con mayúsculas la prueba pueda detectar anomalías de manera automatizada y continua.

Otro de los escenarios importantes para mencionar se encuentra el asegurar que el formulario que presenta la aplicación para el inicio de sesión este a través de una conexión segura como lo es HTTPS:

authetication.feature
Codigo Fuente:

```
@iriusrisk-cwe-295-auth
Scenario: Present the login form itself over an HTTPS connection
  Given a new browser instance
  And the client/browser is configured to use an intercepting proxy
  And the login page is displayed
  And the HTTP request-response containing the login form
  Then the protocol should be HTTPS
```

Tabla 13 Escenario que Verifica el Transporte por HTTPS

Este es un escenario bastante importante, el cual de manera continua y automatizada verifica siempre que el formulario web en que aparece el login esté a través del protocolo HTTPS con el fin de que las credenciales viajen de forma cifrada.

authetication.feature

Codigo Fuente:

```
@iriusrisk-cwe-319-auth
Scenario: Transmit authentication credentials over HTTPS
  Given a new browser or client instance
  And the client/browser is configured to use an intercepting proxy
  And the proxy logs are cleared
  When the default user logs in
  And the HTTP request-response containing the default credentials is selected
  Then the protocol should be HTTPS
```

Tabla 14 Verifica que la Autenticación viaje por HTTPS

10.21 Despliegue

Se refiere a la fase de pruebas donde se verifica que la configuración de la seguridad es correcta teniendo en cuenta los siguientes puntos:

- Configuración correcta de los usuarios, roles y permisos que el sistema tendrá en producción.

- A nivel de infraestructura revisar que la configuración del host y la red sean la esperadas para ello debe validar los puertos que únicamente deben estar abiertos.

10.22 Pruebas Automatizadas Para Probar el Despliegue

Este Feature tiene como finalidad verificar la configuración de los host con respecto a lo mínimo necesario en seguridad que un host debe tener. Por ejemplo, se pude observar que el Scenario consiste en probar que únicamente los puertos 80, 443 se encuentren abiertos que son los puertos por defecto para los protocolos Http y Https.

Host_config.feature

Codigo Fuente:

```
@host_config
Feature: Host Configuration
  Verify that the configuration of the host and network are as expected

  @iriusrisk-open_ports
  Scenario Outline: Only the required ports should be open
    Given the target host name <host>
    When TCP ports from <startPort> to <endPort> are scanned using <threads> threads and a timeout of <timeout> milliseconds
    And the <state> ports are selected
    Then the ports should be <ports>
    Examples:
```

host	startPort	endPort	threads	timeout	state	ports
localhost	1	65535	100	500	open	80,443

Tabla 15 Scenarios Outline Que Verifican los Puertos

10.23 Identificando componentes en el DOM a través de Selenium Web Driver

En el anterior capitulo se dio a conocer algunas configuraciones necesarias para poder poner en funcionamiento el framework. En este paso es necesario hacer uso de este, para lo cual se crea un paquete "net.continuumsecurity.examples.ropeytasks" y dentro una clase con nombre "RopeyTasksApplication". En esta clase se hará uso de webdriver que identificará los elementos del login, con el fin de poderlos automatizar y posteriormente usar el archivo con extensión .feature,

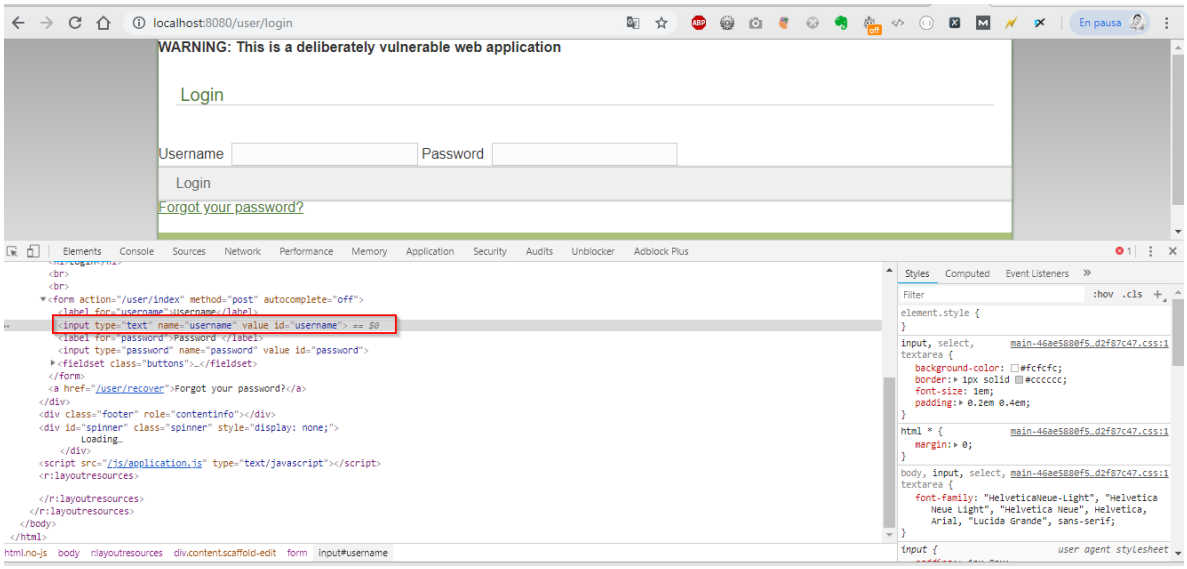


Figura 78 Identificando Nombre de Usuario a Través del DOM

La librería de Selenium Web Driver tiene una API que permite escribir pruebas en diferentes lenguajes de programación. Es posible comunicarse con selenium mediante las llamadas a diferentes funciones que permiten

identificar los elementos en el DOM de una aplicación. La identiicación es posible de diferentes formas, bien sea por id, nombre, o XPath en caso de ser requerido.

RopeyTasksApplication.java

Código Fuente:

```
    /**
     *
     * Metodo que permite identificar los elementos en el DOM de la pantalla Login
     * @param query
     */
    @Override
    public void login(Credentials credentials) {
        UserPassCredentials creds = new UserPassCredentials(credentials);
        driver.findElement(By.id("username")).clear();
        driver.findElement(By.id("username")).sendKeys(creds.getUsername());
        driver.findElement(By.id("password")).clear();
        driver.findElement(By.id("password")).sendKeys(creds.getPassword());
        driver.findElement(By.name("_action_login")).click();
    }
```

Tabla 16 Identificación de Locators A Través del DOM

Gracias a Selenium Web Driver es posible llevar a cabo la automatización de pruebas funcionales y de seguridad en cualquier aplicación web. Es así que las pruebas funcionales automatizadas de software empiezan a apoyar las pruebas de seguridad automatizadas mediante un proceso de revisión continuo y automático.

10.24 Ejecución con BDD Security

Proporcionar una monitorización continua por medio de reportes que indiquen el estado actual de la aplicación a nivel de seguridad, es uno de los trabajos que hace posible este framework cada vez que se ejecuta un archivo ".feature" por medio de gradle que permite correr cada uno de los escenarios que se encuentran tagueados de la siguiente manera:

Tag	Function
@appscan	Runs automated application level tests against the application using OWASP ZAP.
@authentication	Verifies that the authentication is robust
@authorisation	Verify that the access control model is enforced so that only the authorised users have access to their own data.
@data_security	Verify that the application does not allow the browser to cache sensitive data.
@host_config	Verify that the configuration of the host and network are as expected by carrying out a port scan.
@http_headers	Verify that HTTP headers adequately protect data from attackers.
@nessus_scan	Scan the hosts for known security vulnerabilities using Nessus.
@passive_scan	Navigate and spider the application and identify vulnerabilities passively using OWASP ZAP.
@session_management	Verify that there are no weaknesses in the session management implementation.
@ssl	Ensure that the SSL configuration of the service is robust.

Figura 79 Tagueo de todo los Features de BDD Security

Comando Gradle para ejecutar los Escenarios BDD authentication.feature
Código Fuente:
./gradlew -Dcucumber.options="--tags @authentication"

Tabla 17 Comando Que Permite Ejecutar Las Pruebas En BDD Security Framework

En esta oportunidad se han corrido todos los escenarios para el feature que tiene como objetivo comprobar que la autenticación de la aplicación sea robusta. Para hacer la demostración se lleva a cabo la ejecución de estos escenarios a través de la aplicación prueba RopeyStas.

Cabe mencionar que tradicionalmente este tipo de prueba se llevaría a cabo manualmente por el equipo de seguridad. Sin embargo, al automatizar la prueba desde del equipo de QA, se puede realizar en menos de 4 segundos y lo más importante es que estaría realizándose de manera continua. De esta manera el equipo de desarrollo pueda estar informado y actualizado del estado de seguridad de la aplicación gracias al reporte generado.

10.25 Generación de reportes con BDD Security

Cuando se corren las pruebas con gradle, al finalizar se produce un informe para mostrar el resultado de cada prueba individual junto con su resumen de resultados. Cabe mencionar que estos informes están disponibles en numerosos formatos: JSON, XML y HTML.

A continuación, se dará a conocer el reporte que genera este framework, con respecto a la ejecución de pruebas del feature @authentication.

10.26 Reporte por Features

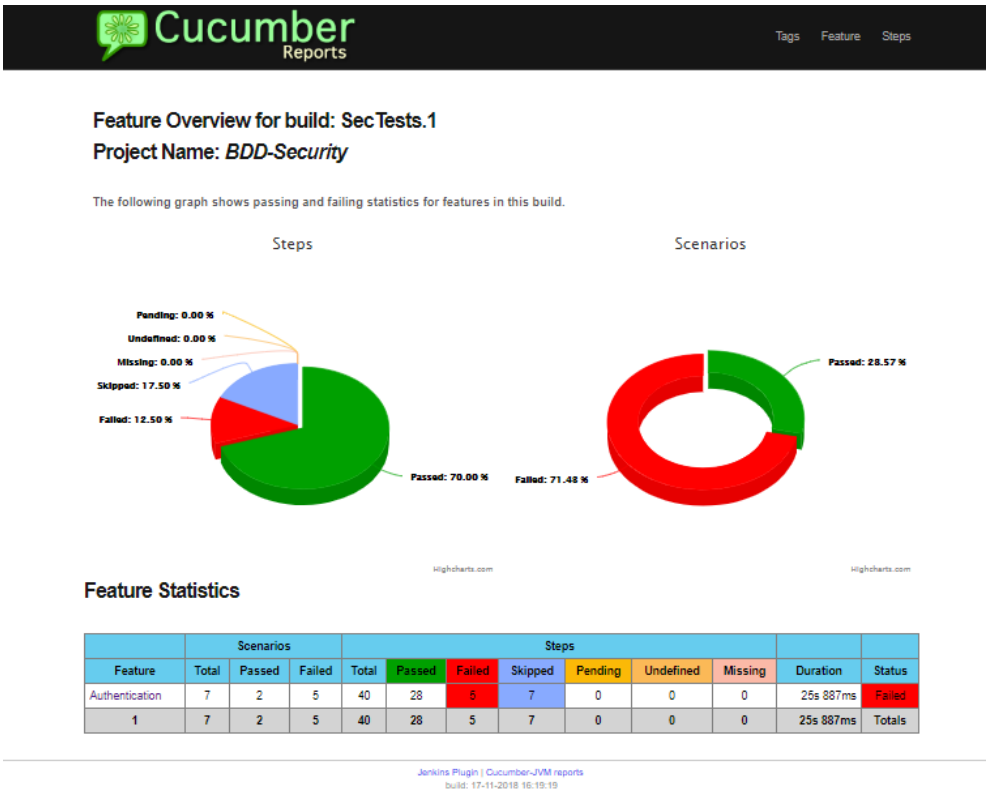


Figura 80 Reporte por Features

El reporte por feature da a conocer un resultado a nivel general de los escenarios ejecutados: el total, cuáles pasaron y cuáles fallaron, y al final la duración que tuvo toda la ejecución y un estatus.

Además de este resumen, se puede ver información más detallada en pruebas individuales dentro de cada sección. Estas indican cualquier falla junto con las razones de la misma.

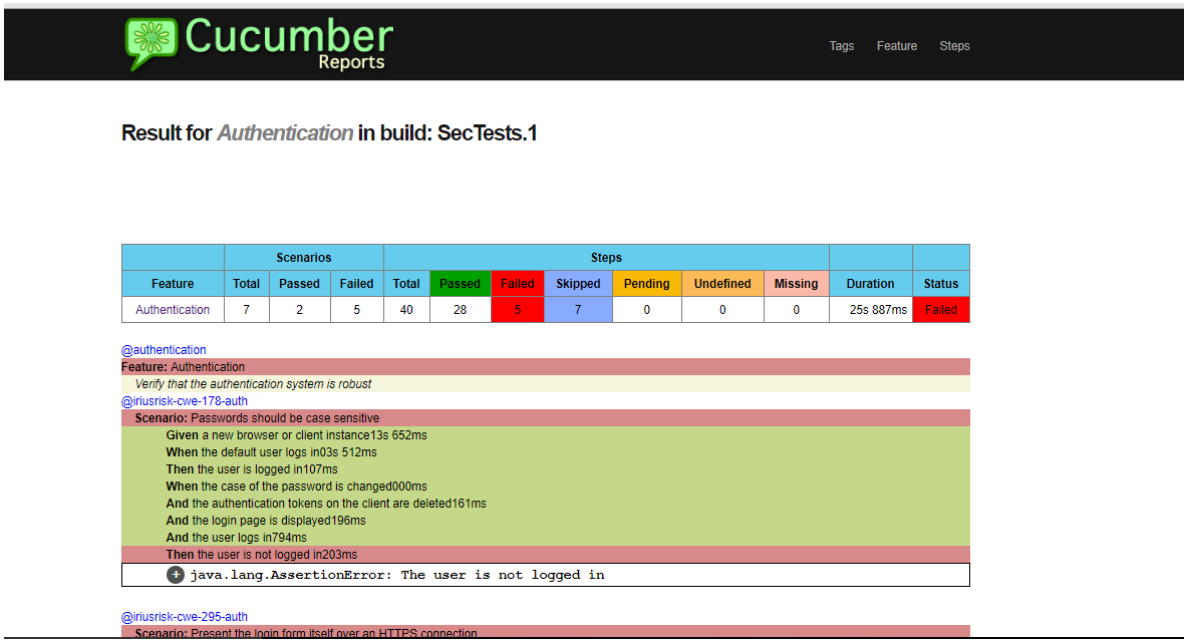


Figura 81 Resumen detallado por Features

10.27 Reporte por Steps

Este reporte da un resultado estadístico por cada uno de los steps ejecutados:

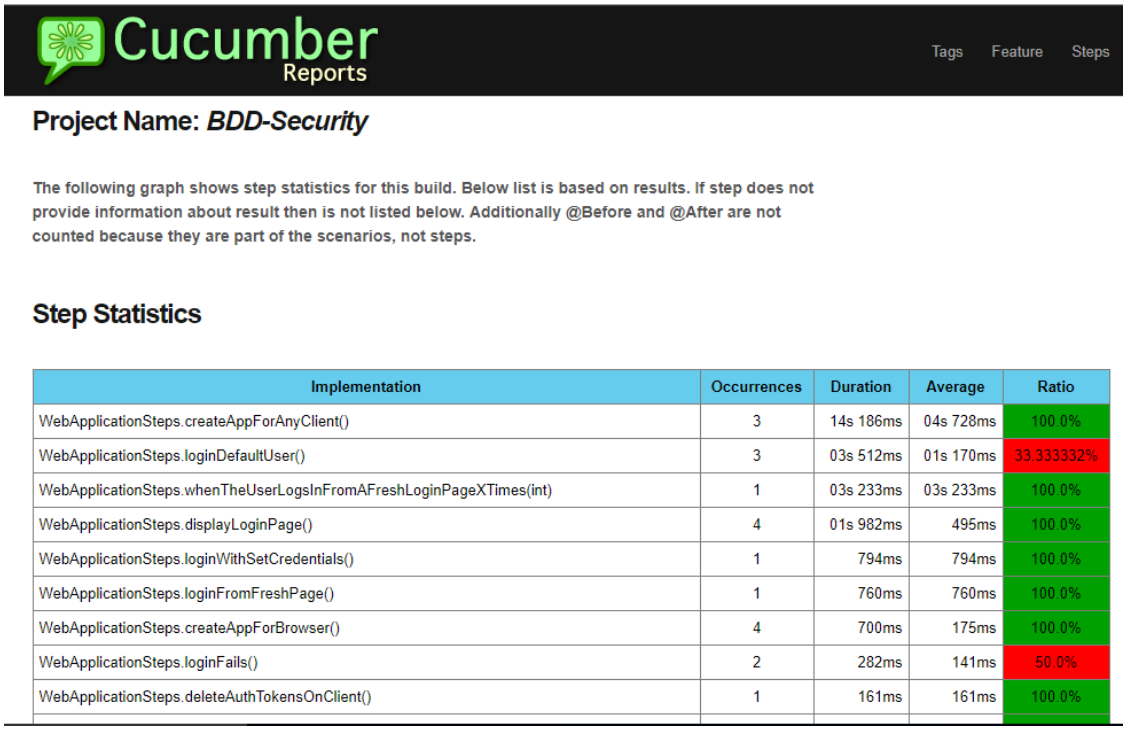
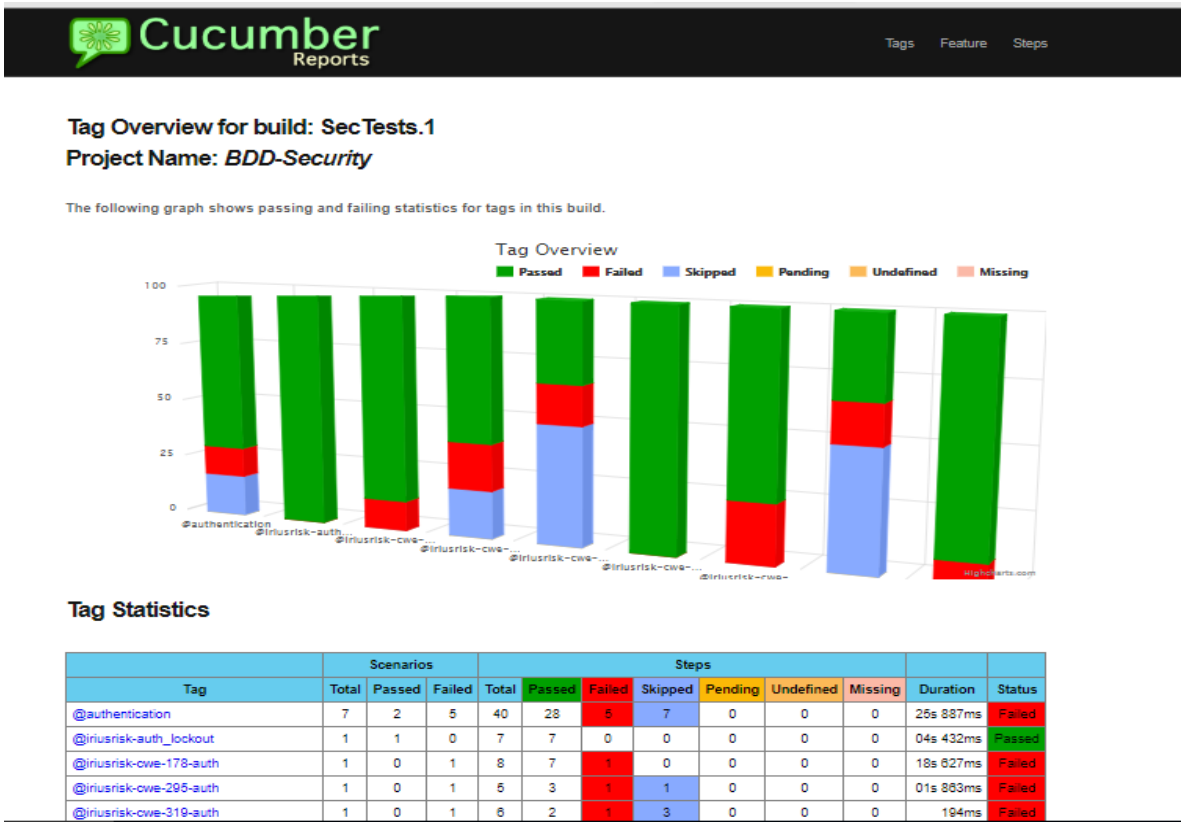


Figura 82 Reporte por Steps

10.28 Reporte de Tags



	Scenarios			Steps								
Tag	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing	Duration	Status
@authentication	7	2	5	40	28	5	7	0	0	0	25s 887ms	Failed
@iriusrisk-auth_lockout	1	1	0	7	7	0	0	0	0	0	04s 432ms	Passed
@iriusrisk-cwe-178-auth	1	0	1	8	7	1	0	0	0	0	18s 627ms	Failed
@iriusrisk-cwe-295-auth	1	0	1	5	3	1	1	0	0	0	01s 863ms	Failed
@iriusrisk-cwe-319-auth	1	0	1	6	2	1	3	0	0	0	194ms	Failed

Figura 83 Reporte por Tags

10.29 Pruebas de Seguridad Automatizadas Tradicionales

Anteriormente se ha detallado la forma de llevar a cabo las pruebas de seguridad automatizadas, basadas en BDD. Estas pruebas son bastante útiles para los equipos de QA que están llevando las pruebas automatizadas utilizando esta tecnología. Sin embargo, donde se dejan aquellos equipos de QA que no están haciendo uso de este tipo de tecnologías y que están desarrollando sus scripts automatizados de la forma tradicional con selenium

web driver sin hacer uso de Cucumber. Como anteriormente se ha mencionado en esta tesis, existe un momento donde el equipo de QA decide cómo automatizar sus pruebas y qué tecnologías usar. De acuerdo a la necesidad de cada proyecto y la forma de trabajo, algunos equipos eligen la automatización con base a BDD y otros la tradicional. Entonces se plantea el siguiente interrogante: ¿cómo se puede involucrar seguridad con las pruebas de automatización tradicionales? la respuesta se ha explicado por partes durante el desarrollo de esta tesis. Adicionalmente, en esta sección se hará hincapié en explicar cómo se puede llevar a cabo esta implementación por medio de un análisis práctico.

En anteriores capítulos se ha mencionado el uso de los Jenkins pipeline como una forma de automatizar los diferentes stages, con el fin de tener un flujo ordenado dentro un ambiente de CI/CD. Estas acciones tienen la finalidad de llevar un control de manera automatizada sobre una aplicación, siempre con la intención de que los equipos de Dev puedan hacer un mayor número de entregas del producto con mayor calidad y seguridad a un menor tiempo.

Figura 84 Stage de Pruebas de Seguridad Dentro un Pipeline

En la imagen anterior se observan los dos stages principales para esta tesis. Denominados Automatización de pruebas funcionales y Automatización de pruebas de seguridad. Si existe una correcta calidad a nivel funcional va servir para que exista una correcta seguridad, dado que estos dos procesos están fuertemente correlacionados. Es por ello que el stage “Automatización de Pruebas Funcionales” debe ir como antecesor del stage “Automatización de Pruebas de Seguridad”. Una vez que pasen todas las pruebas automatizadas como Ok, automáticamente pasa al siguiente stage para efectuar las pruebas de seguridad.

En esta ocasión las pruebas de automatización funcional serán las que alimentarán a las pruebas de seguridad, dado que estas necesitarán hacer un mapeo de toda la aplicación. Para ello es necesario que las pruebas funcionales puedan proveer de esta información, es decir que a medida que la aplicación crece a nivel funcional, los scripts de pruebas automatizadas funcionales van subiendo en número y esto ayuda a alimentar a las pruebas automatizadas de seguridad, teniendo en cuenta que entre más cobertura de una aplicación tenga las pruebas de seguridad mayor será el número de probabilidades de encontrar incidentes de seguridad y funcionales.

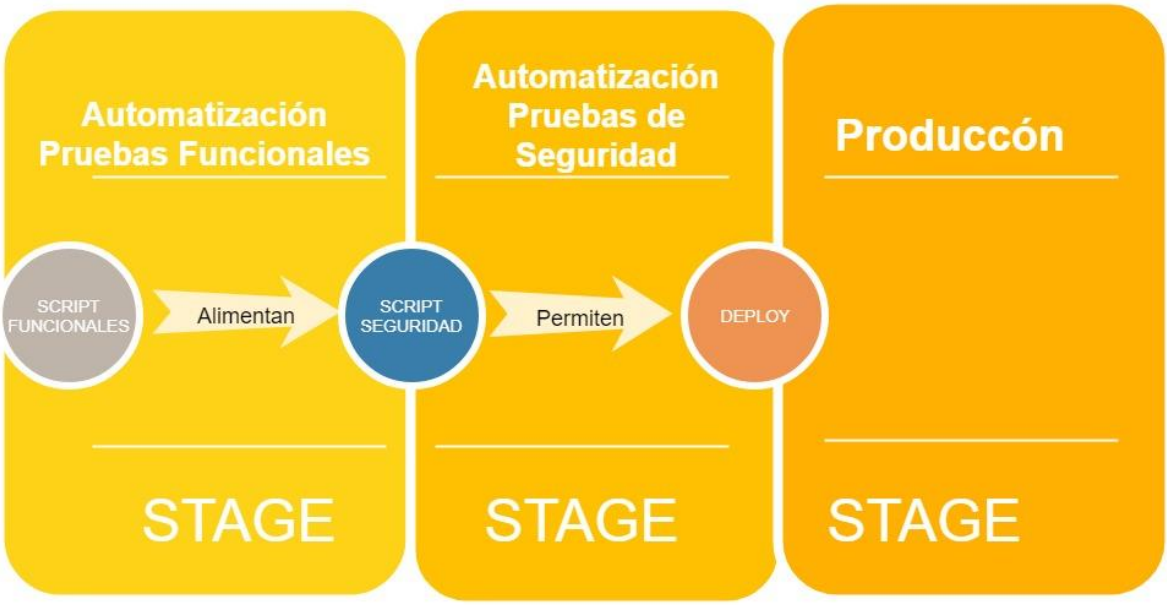


Figura 85 Automatización

Es así que una aplicación puede llegar a producción con un menor número de riesgos a nivel funcional y de seguridad.

10.30 Pruebas Automatizadas con BDD Vs Pruebas Automatizadas Tradicionales

Antes de seguir adelante es necesario explicar las diferencias de las pruebas Automatizadas con BDD y las pruebas Automatizadas Tradicionales.

En primer lugar, las pruebas Automatizadas con BDD como se explicó en la anterior sección, se refieren a aquellas pruebas que usan como tecnología Cucumber y Selenium. Las pruebas Automatizadas Tradicionales se refieren a aquellas pruebas que solo hacen uso de Selenium Web Driver y en su código no existe el lenguaje Gerking.

A continuación, se detalla una tabla de las principales diferencias:

Automatización Scripts con BDD - Cucumber	**Automatización Scripts sin BDD – Selenium Web Driver**
S conoce como una herramienta de prueba basada en un marco de comportamiento impulsado.	Se conoce generalmente como herramienta de prueba de automatización del navegador.
Solo acepta el lenguaje Gerking.	Los scripts de prueba se pueden escribir en Java, C # y otros idiomas
Se necesita escribir escenarios y verificarlo a través de analistas de negocios y otros equipos de administración.	Se necesita escribir scripts basados en casos de prueba y requisitos.
Los scripts de Cucumber son como documentar el procedimiento o la funcionalidad en el orden correcto	Los scripts de Selenium se parecen más a desarrollar una aplicación

Tabla 18 Cucumber vs Web Driver

10.31 Análisis Práctico: llevando a cabo pruebas de seguridad automatizadas sin BDD

La intención de estas pruebas de concepto es dar a conocer a nivel técnico cómo llevar a cabo las pruebas de seguridad automatizadas haciendo uso de la automatización tradicional.

10.32 Descripción del software utilizado

Bodgeit, aplicación vulnerable open source.
Eclipse IDE 2018
API ZAP V 6
Owasp Zap v 2.6

Descripción de la Arquitectura utilizada, donde solo hacen parte las capas Java Core y Java + Selenium Steps, sin incluir la capa de BDD Security Stories que hacen parte de Cucumber, en esta arquitectura solo existe una comunicación entre el Java Core y Selenium Steps.

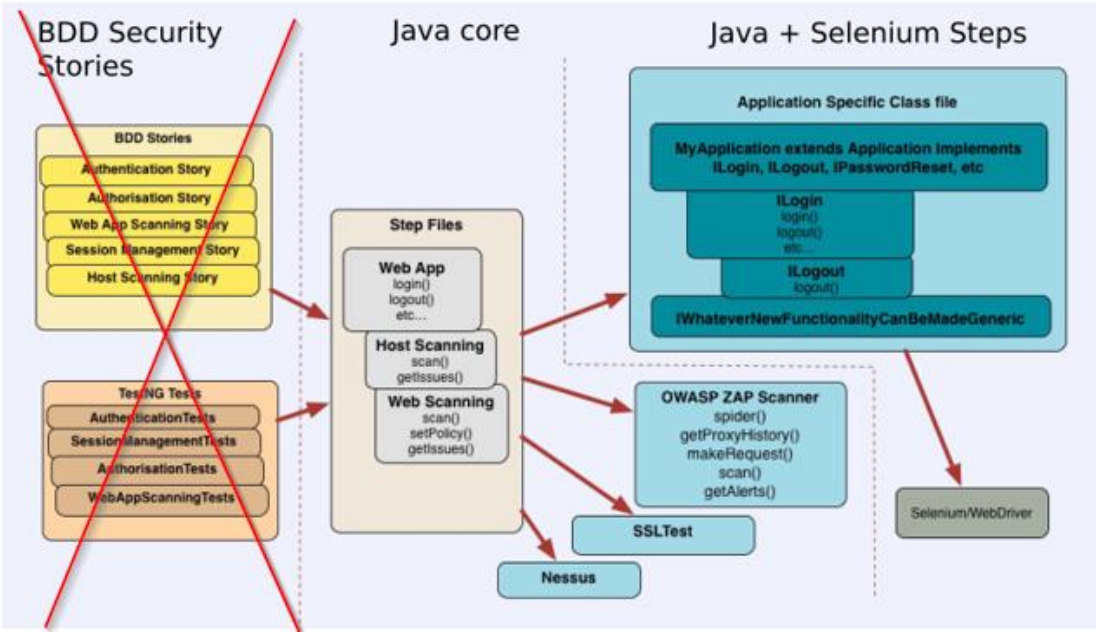


Figura 86 Arquitectura Java + Web Driver y ZAP

Es necesario importar el proyecto desde Eclipse IDE como un proyecto Maven. La siguiente imagen describe cómo es la estructura de un proyecto para pruebas automatizadas funcionales y de seguridad usando la API de Zap.

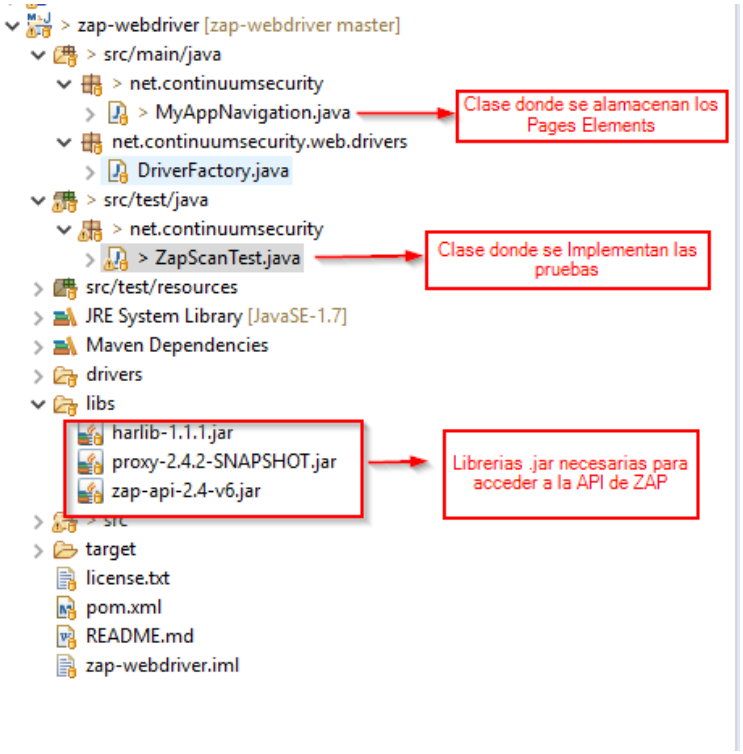


Figura 87 Librerias de ZAP API

Al ingresar a la clase MyAppNavigation.class, en esta clase la idea es empezar a encontrar los elementos en el DOM de la aplicación BodGeit y empezar a mapear toda la pantalla haciendo uso de Selenium Web Driver. A continuación se muestran algunas imágenes que explican la forma de identificar los elementos en pantalla.

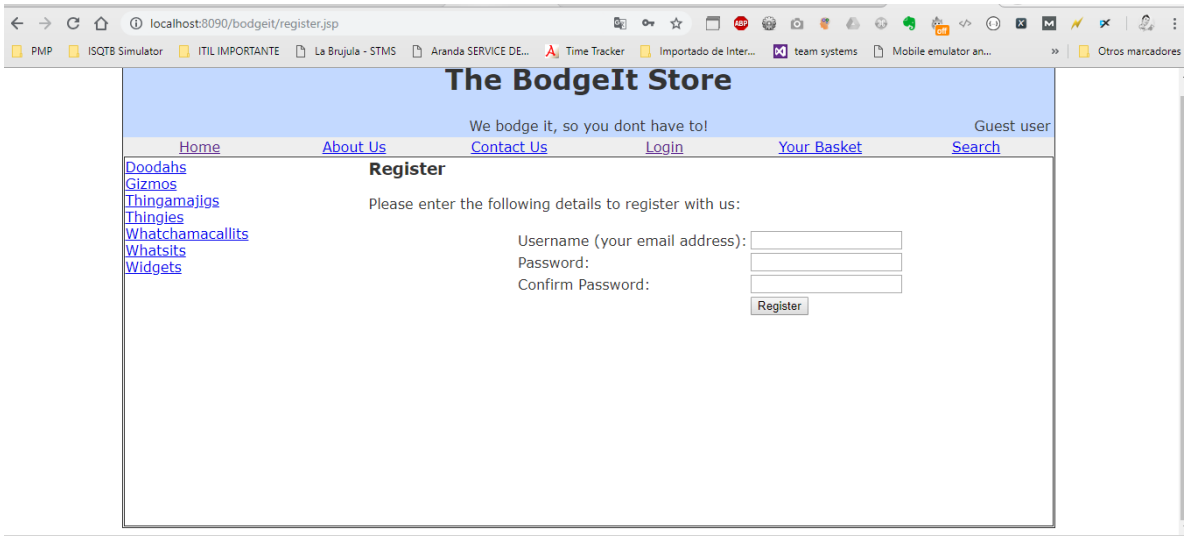


Figura 88 Interfaz gráfica de BodgeIt

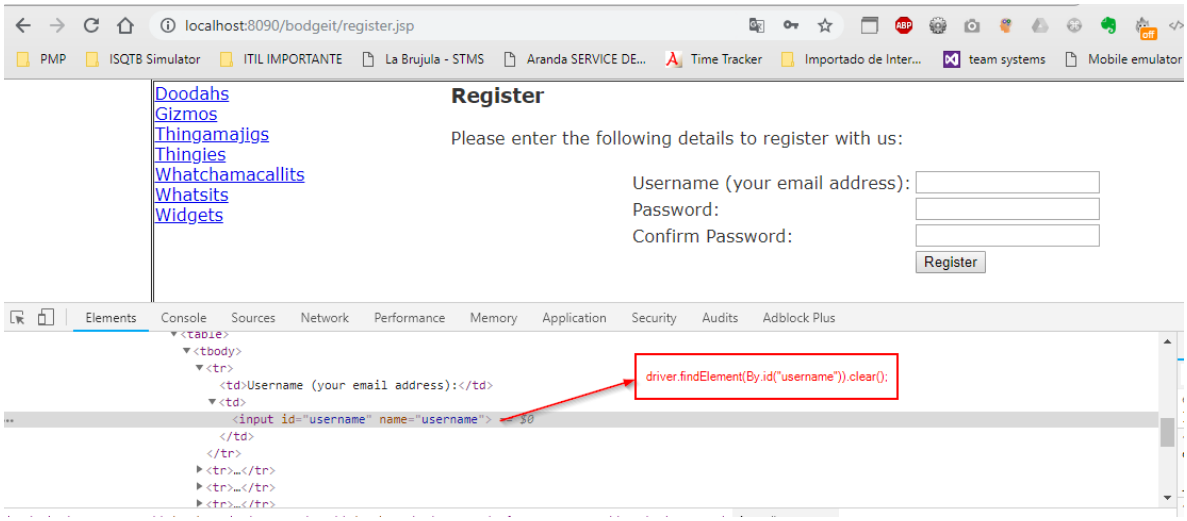


Figura 89 Identificación de nombre de Usuario con Selenium Web Driver

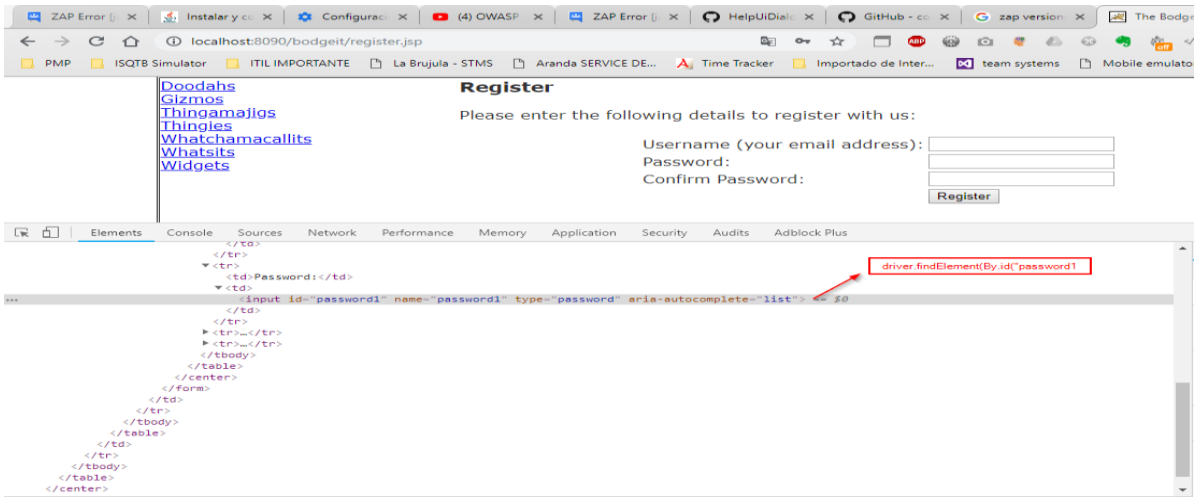


Figura 90 Identificación del Password Con Selenium Web Driver

Cuando se ha mapeado la pantalla con selenium, por acciones se crean diferentes métodos como por ejemplo el método que permite el registro de usuarios registerUser(), o por ejemplo el método que permite realizar el login de un usuario login().

net.continuumsecurity

MyAppNavigation.class

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

import java.util.concurrent.TimeUnit;


public class MyAppNavigation {

  //Instancia de WebDriver
    WebDriver driver;

    //Se crea una variable que permita almacenar la URL de la pagina
que esta
```

```
    //Usando
    final static String BASE_URL = "http://localhost:8090/bodgeit/";
    final static String LOGOUT_URL =
"http://localhost:8090/bodgeit/logout.jsp";

//Se crean variables para la autenticación del usuario de prueba
    final static String USERNAME =
"JohanJavierPizarro@MaestriaSecurity.net";
    final static String PASSWORD = "tables";

// Se crea un Constructor para la Clase que permita incializar las
variables
 public MyAppNavigation(WebDriver driver) {
        this.driver = driver;
        this.driver.manage().timeouts().pageLoadTimeout(5,
TimeUnit.SECONDS);
        this.driver.manage().timeouts().implicitlyWait(5,TimeUnit.SECONDS);
    }
//Método Login que tiene todos los Pages Elements de la Pagina de
Login de La // Aplicación BodGeit y permite loguear a un usuario de
manera automatizada
   public void login() {
        driver.get(BASE_URL);
        driver.findElement(By.linkText("Login")).click();
        driver.findElement(By.id("username")).clear();
        driver.findElement(By.id("username")).sendKeys(USERNAME);
        driver.findElement(By.id("password")).clear();
        driver.findElement(By.id("password")).sendKeys(PASSWORD);
        driver.findElement(By.id("submit")).click();
        verifyTextPresent("successfully");
   }
```

```
//Método registro de usuario que permite registrar un usuario  de manera //automatizada
public void registerUser() {
        driver.get(BASE_URL+"register.jsp");
        driver.findElement(By.id("username")).clear();
        driver.findElement(By.id("username")).sendKeys(USERNAME);
        driver.findElement(By.id("password1")).clear();
        driver.findElement(By.id("password1")).sendKeys(PASSWORD);
        driver.findElement(By.id("password2")).clear();
        driver.findElement(By.id("password2")).sendKeys(PASSWORD);
        driver.findElement(By.id("submit")).click();
    }

//Método que Automatiza todas las acciones que se encuentran después de que un // usuario ha iniciado sesión
public void navigateBeforeLogin() {
        driver.get(BASE_URL);
        driver.findElement(By.linkText("Home")).click();
        driver.findElement(By.linkText("Doodahs")).click();
        driver.findElement(By.linkText("Zip a dee doo dah")).click();
        driver.findElement(By.linkText("About Us")).click();
        driver.findElement(By.linkText("Scoring page")).click();
        driver.findElement(By.linkText("Your Basket")).click();
        driver.findElement(By.linkText("Search")).click();
        driver.findElement(By.name("q")).clear();
        driver.findElement(By.name("q")).sendKeys("test");
        driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();

        //Make sure we're on the page we're supposed to be on
        verifyTextPresent("Results Found");
        driver.findElement(By.linkText("Search")).click();
        driver.findElement(By.linkText("Advanced Search")).click();
        driver.findElement(By.id("product")).clear();
```

```
        driver.findElement(By.id("product")).sendKeys("test");
        driver.findElement(By.id("desc")).clear();
        driver.findElement(By.id("desc")).sendKeys("test");
        driver.findElement(By.id("type")).clear();
        driver.findElement(By.id("type")).sendKeys("test");
        driver.findElement(By.id("price")).clear();
        driver.findElement(By.id("price")).sendKeys("test");
        driver.findElement(By.cssSelector("input[type=\"submit\"]")).click();

        //Make sure we're on the page we're supposed to be on
        verifyTextPresent("Results Found");
    }
```

Tabla 19 Identificación de Componentes Con Selenium Web Driver

Por otro lado, se encuentra la clase más importante de todas ZapScanTest.class. En esta clase se definen todas las pruebas funcionales y se le agrega el plus de pruebas de seguridad usando la API de ZAP.

net.continuumsecurity

ZapScanTest.java

```
import static org.hamcrest.core.IsEqual.equalTo;



public class ZapScanTest {
    static Logger log = Logger.getLogger(ZapScanTest.class.getName());

// Se Agregan variables que permiten configurar las pruebas a través del mismo // puerto de escucha de Owasp Zap
    private final static String ZAP_PROXYHOST = "localhost";
```

```
    private final static int ZAP_PROXYPORT = 8888;
    private final static String ZAP_APIKEY = null;

    // Change this to the appropriate driver for the OS, alternatives in the drivers directory
    private final static String CHROME_DRIVER_PATH = "C:\\Driver\\chromedriver.exe";
    private final static String MEDIUM = "MEDIUM";
    private final static String HIGH = "HIGH";
    private ScanningProxy zapScanner;
    private Spider zapSpider;
    private WebDriver driver;
    private MyAppNavigation myApp;

//Esta una de la variables claves dado que aquí se definen las políticas o //tipos de pruebas de seguridad que necesitamos que la API de ZAP tenga en //cuenta para las pruebas.
    private final static String[] policyNames = {"directory-browsing","cross-site-scripting","sql-injection","path-traversal","remote-file-inclusion","server-side-include",
            "script-active-scan-rules","server-side-code-injection","external-redirect","crlf-injection"};
    int currentScanID;

    @Before
    public void setup() {
        zapScanner = new ZAProxyScanner(ZAP_PROXYHOST,ZAP_PROXYPORT,ZAP_APIKEY);
        zapScanner.clear(); //Start a new session
        zapSpider = (Spider)zapScanner;
        log.info("Created client to ZAP API");
```

```
        driver =
DriverFactory.createProxyDriver("chrome",createZapProxyConfigurationFo
rWebDriver(), CHROME_DRIVER_PATH);
        myApp = new MyAppNavigation(driver);
        myApp.registerUser(); //Doesn't matter if user already exists, bodgeit
just throws an error
    }

    @After
    public void after() {
        driver.quit();
    }

//Aquí se codifican las pruebas funcionales en este caso es una
prueba que
// verifica la seguridad antes de hacer el login Automatizado
    @Test
    public void testSecurityVulnerabilitiesBeforeLogin() {
        myApp.navigateBeforeLogin();
        log.info("Spidering...");
        spiderWithZap();
        log.info("Spider done.");

        setAlertAndAttackStrength();
        zapScanner.setEnablePassiveScan(true);
        scanWithZap();

        List<Alert> alerts = filterAlerts(zapScanner.getAlerts());
        logAlerts(alerts);
        assertThat(alerts.size(), equalTo(0));
    }
```

```
// Metodo que permite realizar un log de todas las alertas que Zap API encuentra
  private void logAlerts(List<Alert> alerts) {
        for (Alert alert : alerts) {
           log.info("Alert: "+alert.getAlert()+" at URL: "+alert.getUrl()+" Parameter: "+alert.getParam()+" CWE ID: "+alert.getCweId());
        }
    }

//Metodo que remueve los falsos positivos
private List<Alert> filterAlerts(List<Alert> alerts) {
       List<Alert> filtered = new ArrayList<Alert>();
       for (Alert alert : alerts) {
          if (alert.getRisk().equals(Alert.Risk.High) && alert.getConfidence() != Alert.Confidence.Low) filtered.add(alert);
       }
       return filtered;
    }

    public void setAlertAndAttackStrength() {
        for (String policyName : policyNames) {
           String ids = enableZapPolicy(policyName);
           for (String id : ids.split(",")) {
              zapScanner.setScannerAlertThreshold(id,MEDIUM);
              zapScanner.setScannerAttackStrength(id,HIGH);
           }
        }
    }

//Este es el método que pone a escuchar a WebDriver en el mismo puerto que se //encuentra escuchando el proxy de ZAP.
private static Proxy createZapProxyConfigurationForWebDriver() {
        Proxy proxy = new Proxy();
```

```
        proxy.setHttpProxy(ZAP_PROXYHOST + ":" + ZAP_PROXYPORT);
        proxy.setSslProxy(ZAP_PROXYHOST + ":" + ZAP_PROXYPORT);
        return proxy;
    }

//Este es el método que hace un escaneo Spider sobre la aplicación.
private void spiderWithZap() {
        zapSpider.excludeFromSpider(myApp.LOGOUT_URL);
        zapSpider.setThreadCount(5);
        zapSpider.setMaxDepth(5);
        zapSpider.setPostForms(false);
        zapSpider.spider(myApp.BASE_URL);
        int spiderID = zapSpider.getLastSpiderScanId();
        int complete  = 0;
        while (complete < 100) {
            complete = zapSpider.getSpiderProgress(spiderID);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        for (String url : zapSpider.getSpiderResults(spiderID)) {
            log.info("Found URL: "+url);
        }
    }
```

Tabla 20 Configuración de Proxy y Construcción de Pruebas Con Web Driver

11.0 Conclusiones

La seguridad siempre debe ser un proceso continuo durante toda la etapa de desarrollo de un producto y debe estar en conjunto con las pruebas

funcionales. Por este motivo los equipos de QA deben incluir dentro de sus procesos de prueba, de forma indispensable, la seguridad.

En muchas ocasiones suele pasar que los integrantes de los equipos de QA no incluyen la seguridad dentro de sus procesos. Se fundamentan en que no son expertos en el tema e ignoran que este proceso fundamental. Al mismo tiempo, dejan la responsabilidad a los equipos de seguridad. Esta forma de actuar se debe cambiar, dado que la seguridad de un desarrollo de software no es de unos pocos sino de todos los equipos y si hablamos del equipo de QA es el equipo que más responsabilidad tiene sobre la calidad del producto, tanto a nivel funcional como de seguridad.

Por suerte, el enfoque DevOps ha llegado a los equipos para quedarse, convirtiéndose en un gran integrador de roles donde todos deben colaborar con todos. Sin embargo, esto genera un desafío general en las compañías y también a nivel personal de cada uno de los integrantes, dado que tanto los empresarios como los miembros del equipo deben estar en constante capacitación, para que la colaboración sea más efectiva.

Las empresas deben incentivar cada vez más los Hackatons de manera regular donde puedan participar equipos de Desarrollo y QA con el fin de que puedan pensar como atacantes, para que cada uno se pueda hacer a una idea de cómo podrían los intrusos atacar sus sistemas.

En muchas organizaciones no hay tantos ingenieros de seguridad como lo son los equipos de Dev y QA. Entonces las compañías que quieran optar por tener un enfoque SecDevOps deben aprovechar sus pocos ingenieros de seguridad para que estos puedan capacitar a los miembros de Dev y QA, ya que una vez que están debidamente capacitados, estarán mejor equipados para comprender las partes del código y a nivel funcional que puedan ser vulnerables desde una perspectiva de seguridad.

12.0 Bibliografía

[1] Wikipedia, «Desarrollo en cascada,» 29 Octubre 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Desarrollo_en_cascada.

[2] BSIMM, «BSIMM,» 2018. [En línea]. Available: https://www.bsimm.com/download.html.

[3] Ponemon Institute LLC , «Estudio del coste de una brecha de,» Ponemon Institute LLC , Estados Unidos, 2017.

[4] P. Santapau, «continuumsecurity.,» 2017. [En línea]. Available: https://continuumsecurity.net/.

[5] S. Wisseman, «TechBeacon,» 2015. [En línea]. Available: https://techbeacon.com/app-dev-testing/application-security-qa-why-they-are-better-together.

[6] J. R. Cristina, «ParadigmaDigital,» 26 Noviembre 2015. [En línea]. Available: https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops/.

[7] Wikipedia, «Wikipedia,» 9 Julio 2012. [En línea]. Available: https://es.wikipedia.org/wiki/DevOps#/media/File:Devops.svg.

[8] IBM, «IBM,» 2018. [En línea]. Available: https://www.ibm.com/es-es/cloud/devops/get-started.

[9] M. Setter, «blog.sqreen.io,» 19 Julio 2017. [En línea]. Available: https://blog.sqreen.io/secdevops/.

[10] P. Santapau, «YouTube,» 19 Mayo 2017. [En línea]. Available: https://www.youtube.com/watch?v=ORTnb_NiMxg.

[11] I. Eldridge, «New Relic,» 16 Julio 2018. [En línea]. Available: https://blog.newrelic.com/technology/what-is-secdevops/.

[12] M. Robles, «OWASP,» 2018. [En línea]. Available: https://www.owasp.org/index.php/Main_Page.

[13] «Reto Isaca 2018,» 2018. [En línea]. Available: http://www.isaca.org/chapters7/Madrid/AboutourChapter/Jovenes-Profesionales/Pages/Reto.aspx.

[14] softwaretestinghelp, «Software Testing Help,» 31 Agosto 2018. [En línea]. Available: https://www.softwaretestinghelp.com/continuous-integration-in-devops/.

[15] A. Phillips, «DEVOPS,» 29 Julio 2014. [En línea]. Available: https://devops.com/continuous-delivery-pipeline/.

[16] Tricentis, «Tricentis,» [En línea]. Available: https://www.tricentis.com/products/what-is-continuous-testing/.

[17] dzone, «dzone,» 09 Agosto 2018. [En línea]. Available: https://dzone.com/articles/building-a-continuous-delivery-pipeline-using-jenk.

[18] OWASP, «OWASP Top 10 - 2017,» 2017.

[19] «OWASP,» 2016. [En línea]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[20] Jenkins, «Jenkins,» 2018. [En línea]. Available: https://jenkins.io/doc/book/pipeline/.

[21] U. Farook, «we45,» 28 Febrero 2018. [En línea]. Available: https://www.we45.com/blog/how-to-integrate-zap-into-jenkins-ci-pipeline-we45-blog.

[22] Guru99, «Guru99,» [En línea]. Available: https://www.guru99.com/jenkins-pipeline-tutorial.html.

[23] continuumSecurity, «continuumsecurity,» 2019. [En línea]. Available: https://continuumsecurity.net/bdd-security/.

[24] continuumsecurity, «github,» 2019. [En línea]. Available: https://github.com/continuumsecurity/bdd-security/wiki/2-Getting-Started.

[25] S. d. Vries, «The OWASP™ Foundation,» 2017. [En línea]. Available: https://www.owasp.org/index.php/Main_Page.

[26] Oracle, «Java,» 2019. [En línea]. Available: https://www.java.com/es/download/.

[27] Eclipse Foundation, «Eclipse,» 2018. [En línea]. Available: https://www.eclipse.org/downloads/packages/release/neon/3/eclipse-ide-java-ee-developers.

[28] «GitHub,» 2019. [En línea]. Available: https://github.com/.

[29] gradle, «gradle,» 2019. [En línea]. Available: https://gradle.org/.

[30] Cucumber, «Cucumber,» 2019. [En línea]. Available: https://cucumber.io/.

[31] Selenium.org, «SeleniumHQ,» 2019. [En línea]. Available: https://www.seleniumhq.org/.

[32] IBM, «Altoro Mutual,» 2019. [En línea]. Available: https://demo.testfire.net/.

[33] RopeyTasks, «RopeyTasks,» 2019. [En línea]. Available: https://github.com/continuumsecurity/RopeyTasks.

[34] D. R. ALCAIDE, «DAVID ROMERO ALCAIDE,» 1 Abril 2017. [En línea]. Available: https://davidromeroalcaide.wordpress.com/2017/04/01/integracion-continua-con-jenkins-pipeline/.

[35] V. F. Membrillera, «slideshare,» 10 Julio 2018. [En línea]. Available: https://www.slideshare.net/VanesaFernandezMembr/pipeline-de-integracin-continua.