HERRAMIENTAS DE PRUEBA DE SEGURIDAD DE APLICACIONES

Gina Alexandra Gomez Zafra
Máster interuniversitario en seguridad de las tecnologías de la información y de las comunicaciones (MISTIC)

Pau del Canto
Victor Garcia Font
Marzo de 2017

TABLA DE CONTENIDO

RESUMEN DEL PROYECTO

TITULO: Herramientas de prueba de seguridad de aplicaciones
AUTORES: GINA ALEXANDRA GOMEZ ZAFRA
PALABRAS CLAVES: Ciclo de vida del desarrollo, MarshalSec, estático, dinámico
DESCRIPCION:
En la actualidad, existen cantidades de empresas desarrolladoras de soluciones web y un sin número de herramientas disponibles para llevar a cabo la construcción de las mismas, estas pueden desarrollarse en código abierto o software licenciados, Este proceso de desarrollo de software sigue un proceso establecido: el ciclo de vida de desarrollo de software (SDLC). Es un plan de mejores prácticas que se ha adaptado a lo largo de los años y recomienda cómo se debe desarrollar, mantener y actualizar el software. Históricamente, la seguridad fue una idea tardía durante todo el proceso hasta hace unos años cuando se agregó una "S" adicional para "seguro", y aquellos en DevOps se encontraron con una nueva palabra de moda: ciclo de vida de desarrollo de software seguro o SSDLC. procesos manuales de seguridad como parte del ciclo de vida.
Aunque SSDLC no es un concepto nuevo, debemos cambiar la forma de pensar sobre cómo se implementa. La mayoría de las empresas que desarrollan software creen que lo hacen de forma segura de hecho, como señala el artículo de MarshalSec: no todos los desarrolladores leen el documentación de las bibliotecas que están usando.

Por esta razón se requiere de la realización de pruebas de seguridad para las aplicaciones web antes de utilizarlas con el fin de minimizar los riesgos de exposición, Existen herramientas como Pruebas de seguridad de aplicaciones estáticas (SAST) y Pruebas de seguridad de aplicaciones dinámicas (DAST), que pueden ser útiles en la fase de implementación y prueba, respectivamente, y tienen el beneficio de garantizar que la seguridad se encuentre en el proceso.
Por consiguiente, se propone el análisis de herramientas para el testeo de aplicaciones web y desarrollo de las metodología sobre una aplicación con HP Fortify.

ABSTRACT

TITLE: Application Security Testing Tools
AUTHORS: GINA ALEXANDRA GOMEZ ZAFRA
KEYWORDS: Development lifecycle, MarshalSec, static, dinamic
DESCRIPTION:
Currently, there are lots of companies developing web solutions and a number of tools available to carry out the construction of the same, these can be developed in open source or licensed software. This process of software development follows an established process: the software development life cycle (SDLC). It is a best practice plan that has been adapted over the years and recommends how the software should be developed, maintained and updated. Historically, security was a belated idea throughout the process until a few years ago when an additional "S" was added for "secure," and those in DevOps came across a new buzzword: safe software development lifecycle or SSDLC. manual safety processes as part of the life cycle. Although SSDLC is not a new concept, we must change the way we think about how it is implemented. Most companies that develop software believe that they do it safely in fact, as the MarshalSec article points out: not all developers read the documentation of the libraries they are using.

For this reason it is necessary to carry out security tests for web applications before using them in order to minimize exposure risks. There are tools such as Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST), which can be useful in the implementation and testing phase, respectively, and have the benefit of ensuring that security is in the process.
Therefore, the analysis of tools for the testing of web applications and development of the methodology on an application with HP Fortify is proposed.

INTRODUCCIÓN

Hoy en día existe una gran necesidad de incorporar seguridad en el proceso de desarrollo de software. A menudo existe la creencia de que la seguridad aumenta los tiempos en el proceso de desarrollo, que en última instancia afecta el tiempo de entrega(deadline). Pero existe un gran riesgo de que se liberen productos vulnerables si la seguridad no es implementada desde el inicio del desarrollo. Claramente, ninguna opción es ideal.

Es aquí donde entra en juego la automatización. Idealmente, necesita una integración transparente y una automatización completa de la solución de seguridad en todas las etapas del proceso de desarrollo. A diferencia de llevar a cabo el proceso manualmente, la automatización del proceso proporcionará hallazgos y retroalimentación continuamente.

La automatización resuelve varios de los viejos problemas asociados con los procesos SSDLC tradicionales: significa que la seguridad es un elemento central en todo momento, no ralentiza DevOps, incorporar la seguridad en el ciclo de vida del desarrollo desde el principio asegurará un producto final mucho más sólido. Puede llevar un poco más de tiempo entregar el software, pero, a la larga, dará sus frutos.

OBJETIVOS

OBJETIVO GENERAL

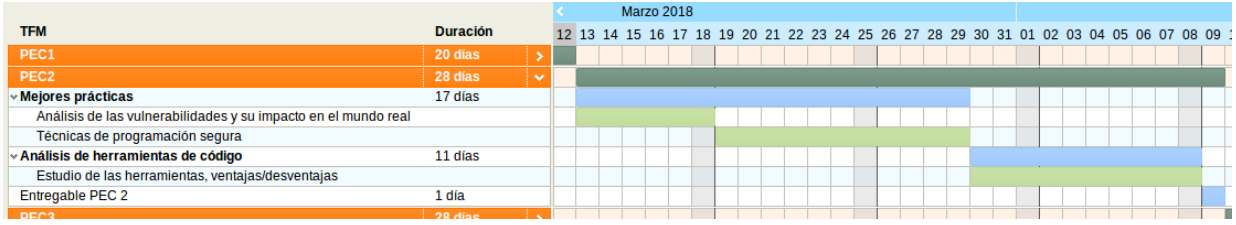
Cuantificar el nivel de riesgo para la seguridad y realizar un análisis de la seguridad de código en todo el ciclo de vida del desarrollo de software desde el desarrollo hasta la implementación utilizando análisis de código dinámico y estático.

OBJETIVOS ESPECÍFICOS

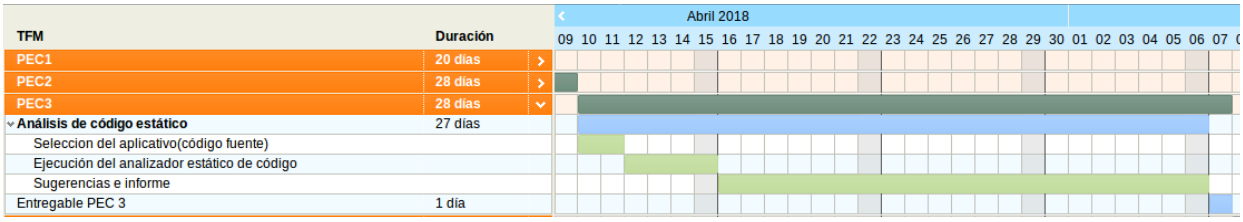
- Estudio los diez riesgos de seguridad más importantes en aplicaciones web según la organización OWASP
- Realizar un análisis sobre las características que ofrecen las herramientas que apoyan el proceso de análisis de seguridad del código.
- Resaltar la importancia de la ejecución de pruebas de seguridad en todo el ciclo de vida del desarrollo de software.
- Realizar un análisis de código estático y dinámico automatizado haciendo uso de las herramientas de HP Fortify.

PLANIFICACIÓN TEMPORAL

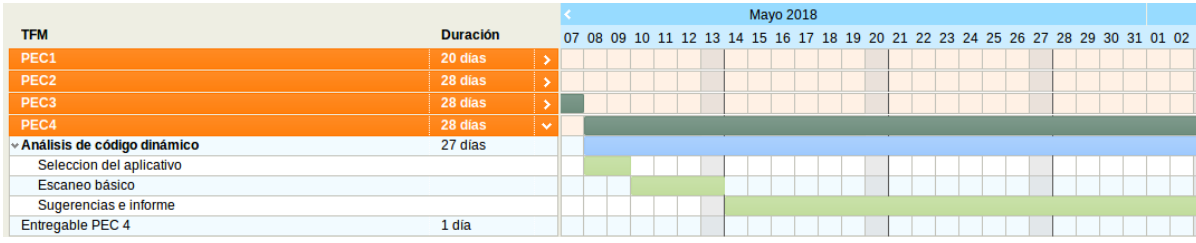
TFM	Duración
PEC1	20 días
Introduccion	4 días
Enunciar problema y justificación del proyecto	3 días
Formulacióno de los objetivos	2 días
Planificación temporal	3 días
Marco de referencia	7 días
Definición del marco conceptual	
Definición del marco teórico	
Entregable PEC 1	1 día



TFM	Duración
PEC1	20 días
PEC2	28 días
Mejores prácticas	17 días
Análisis de las vulnerabilidades y su impacto en el mundo real	
Técnicas de programación segura	
Análisis de herramientas de código	11 días
Estudio de las herramientas, ventajas/desventajas	
Entregable PEC 2	1 día



TFM	Duración
PEC1	20 días
PEC2	28 días
PEC3	28 días
Análisis de código estático	27 días
Seleccion del aplicativo(código fuente)	
Ejecución del analizador estático de código	
Sugerencias e informe	
Entregable PEC 3	1 día



TFM	Duración
PEC1	20 días
PEC2	28 días
PEC3	28 días
PEC4	28 días
Análisis de código dinámico	27 días
Seleccion del aplicativo	
Escaneo básico	
Sugerencias e informe	
Entregable PEC 4	1 día

MARCO DE REFERENCIA

MARCO CONCEPTUAL

- OWASP: Proyecto de código abierto dedicado a determinar y minimizar las causas que hacen que el software sea inseguro.
- HP Fortify: Soluciones completas de seguridad de aplicaciones.
- Análisis estático: Tipo de análisis que se le realiza al software bajo prueba, sin ejecutar el programa, se revisa estándares, reglas de diseño, inspección de código. Detectan código malicioso, faltas a reglas, estándares y mal uso de recursos. Son para garantizar la calidad antes de la ejecución evaluando el diseño.
- Análisis dinámico: Este análisis se realiza en tiempo de ejecución su finalidad es identificar punteros no asignados, comprobar la aritmética de punteros y para controlar la asignación, el uso y desafectación de la memoria e indicar las fugas de memoria.
- Vulnerabilidad: Es una debilidad en un mecanismo que compromete directamente la confidencialidad, integridad o disponibilidad del activo al que se le atribuye. Una vulnerabilidad puede ser identificada dentro de un programa, un equipo o incluso en procedimientos que pueden ofrecer al atacante una puerta abierta. Las vulnerabilidades se caracterizan por la ausencia o debilidad de una medida que puede ser explotada.

MARCO TEÓRICO

- Seguridad Informática

La información es uno de los activos más valiosos para las empresas y es uno de sus objetivos garantizar la existencia de ella. Se puede definir la seguridad informática como el conjunto de medidas que se deben tomar para prevenir, detectar y corregir los posibles problemas asociados con la integridad, confidencialidad y disponibilidad de los recursos informáticos.

Integridad: La finalidad de este principio es asegurar que los datos no han sido alterados ni destruidos de modo no autorizado. Para evitar este tipo de riesgos se debe dotar al sistema de mecanismos que prevengan y detecten cuándo se produce un fallo de integridad y que puedan tratar y resolver los errores que se han descubierto.

Confidencialidad: Se define la confidencialidad como el hecho de que los datos o información estén únicamente al alcance del conocimiento de las personas, entidades o mecanismos autorizados, en los momentos autorizados y de una manera autorizada.

Disponibilidad: La disponibilidad está asociada a la fiabilidad técnica de los componentes del sistema de información y se define como el grado en el que un dato está en el lugar, momento y forma en que es requerido por el usuario autorizado.

Responsabilidad. Todas las acciones relevantes de seguridad del software o de los usuarios se deben almacenar y rastrear, con atribución de responsabilidad. Este rastreo debe ser posible en ambos casos, es decir, mientras y después de que la acción registrada ocurra. Según la política de seguridad para el sistema se debería indicar cuáles acciones se consideran como seguridad relevante, lo que podría hacer parte de los requisitos de auditoría.

No repudio. Esta propiedad le permite al software y a los usuarios refutar o denegar responsabilidades de acciones que ha ejecutado. Esto asegura que la responsabilidad no puede ser derribada o evadida.

- OWASP Top 10

El Proyecto de seguridad de aplicaciones web abiertas (OWASP) es un comunidad abierta dedicada a permitir a las organizaciones desarrollar, comprar y mantener aplicaciones seguras.

El desarrollo de software se ha vuelto cada vez más complejo y la dificultad de lograr la seguridad de las aplicaciones aumenta exponencialmente. El rápido ritmo de los modernos procesos de desarrollo de software hace que los riesgos más comunes sean esenciales para descubrir y resolver de forma rápida y precisa.
El proyecto OWASP presenta los 10 errores de seguridad más comunes en el desarrollo del software:

A1:2017 - Inyección
Estas fallas ocurren cuando se envían datos no confiables a un intérprete, como parte de un comando o consulta y permiten la ejecución de comandos involuntarios y/o acceso a los datos sin la debida autorización.

A2:2017 - Pérdida de Autenticación y Gestión de Sesiones
Funciones implementadas incorrectamente relacionadas a autenticación y gestión de sesiones en la aplicación, permiten a los atacantes comprometer usuarios, contraseñas, token de sesiones o explotar otras fallas de implementación para asumir la identidad de otros usuarios.

A3:2017 - Exposición de Datos Sensibles
El mal manejo a los datos sensibles permite a los atacantes robar o modificar dichos datos con el fin de cometer delitos. Los datos sensibles requieren métodos de protección adicionales, como el cifrado en almacenamiento y tránsito.

A4:2017 - Entidad Externa de XML (XXE)
Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML. Las entidades externas pueden utilizarse para revelar archivos, escanear puertos de la LAN, ejecutar código de forma remota y realizar ataques de denegación de servicio (DoS).

A5:2017 - Pérdida de Control de Acceso
Esta falla se debe a la falta de restricción o validaciones sobre los permisos que tienen los usuarios autenticados en la aplicación.

A6:2017 - Configuración de Seguridad Incorrecta
Hace referencia a la falta o mala configuración de seguridad de todo el conjunto de elementos que comprende el despliegue de una aplicación web, desde la misma aplicación hasta la configuración del sistema operativo o el servidor web.

A7:2017 - 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 los cuales permiten ejecutar comandos en el navegador de la víctima.

A8:2017 - Deserialización Insegura
Estos defectos ocurren cuando una aplicación recibe objetos serializados dañinos y estos objetos pueden ser manipulados o borrados por el atacante para realizar ataques de repetición, inyecciones o elevar sus privilegios de ejecución.

A9:2017 - Uso de Componentes con Vulnerabilidades Conocidas
La implementación de un componente externo se debe analizar a fondo debido a que estos pueden contener vulnerabilidades debilitando las defensas de las aplicaciones y permitir diversos ataques e impactos.

A10:2017 - Registro y Monitoreo Insuficientes
El registro 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.

ANÁLISIS DE VULNERABILIDADES Y SU IMPACTO EN EL MUNDO REAL

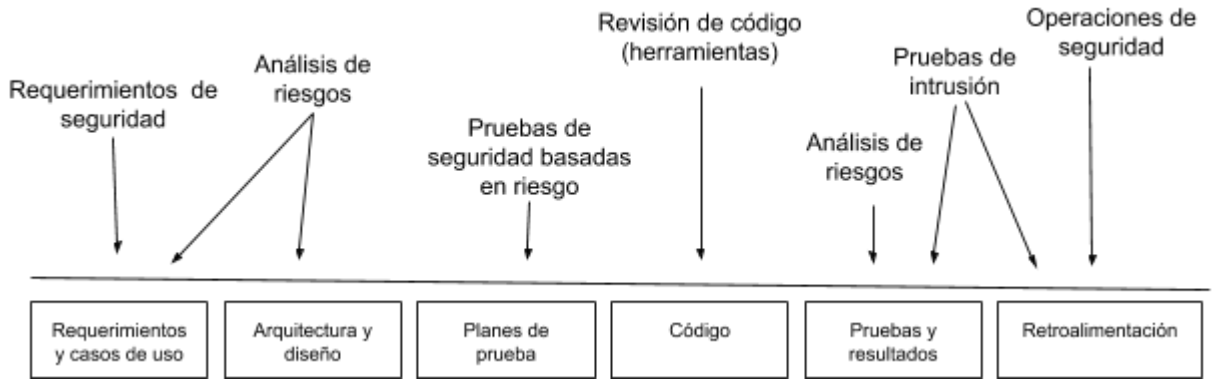
El primer día de clase, los ingenieros mecánicos aprenden una lección crítica: preste atención y aprenda esto, o el puente que construya podría caerse (como lo sucedido en el puente Tacoma Narrows en 1940). Por el contrario, en el primer día de la clase de ingeniería de software, a los desarrolladores se les enseña que pueden construir cualquier cosa que puedan soñar. Por lo general, comienzan con "hola mundo".
Un enfoque demasiado optimista para el desarrollo de software ciertamente ha llevado
a la creación de algunas cosas alucinantes, pero también nos hemos olvidado de pensar qué sucedería si el software fuera atacado intencional o maliciosamente.
El mayor problema en la seguridad informática hoy en día es que la mayoría de los sistemas no están construidos teniendo en cuenta la seguridad. Las tecnologías de red reactiva, como los cortafuegos, pueden ayudar a aliviar los ataques infantiles de guiones obvios en los servidores, pero no hacen nada para solucionar el verdadero problema de seguridad: un mal software. Si queremos resolver el problema de seguridad informática, tenemos que hacer más para desarrollar software seguro.

La seguridad del software es la práctica de construir software seguro y que funcione correctamente bajo un ataque malicioso. A medida que los profesionales toman conciencia de la importancia de la seguridad del software, adoptan y desarrollan cada vez más un conjunto de mejores prácticas.

PRUEBAS DE SEGURIDAD EN EL SDLC

Regularmente se han conocido casos identificados al robo de identidad o datos, la mayoría de organizaciones son vulnerables en mayor medida.
Las organizaciones asumen que un Firewall/IDS o cualquier otro dispositivo de seguridad de red será suficientes para la infraestructura, los datos de información procesados por las aplicaciones se vuelven seguros por sus dispositivos pero el hecho de que la mayoría de los ataques dirigidos contra aplicaciones desarrolladas inseguramente evadirán todos estos dispositivos de seguridad y actualmente la mayoría de los ataques se dirigen al nivel de aplicación.
Por esta razón, los desarrolladores y los arquitectos de aplicaciones necesitan conocer la importancia de la seguridad de la aplicación y cómo puede incorporada como un proceso en el ciclo de vida del desarrollo.
La existencia de vulnerabilidades de una aplicación puesta en producción es responsabilidad del dueño de la empresa y tendrá grave impacto de ser explotadas.
La seguridad puede ser considerada como una solución holística en lugar de considerar aisladamente, de modo que la validación de seguridad es un proceso ejecutado por QA o por los test de penetración y aún así podría llegar a ser insuficiente.
Las pruebas de seguridad del software aseguran que el producto final resulte seguro para los clientes, normalmente se finaliza el desarrollo del software y antes de poner en producción se verifican si hay vulnerabilidades, los problemas generalmente se solucionan con parches, pero estoy resulta ser mucho más costoso que abordar el problema real.
Si los problemas se solucionan durante el proceso de desarrollo de software gran parte de los costos se reducen al evitar múltiples ciclos de pruebas y parches

A continuación se especifica los puntos de contacto de seguridad del software y muestra cómo los profesionales del software pueden aplicarlos a los diversos artefactos de software producidos durante el desarrollo del software. Esto significa entender cómo algunos puntos de contacto son, por su propia naturaleza, más poderosos que otros.



TÉCNICAS DE PROGRAMACIÓN SEGURA

Las buenas prácticas de programación segura es un tema que hace parte parte integral del proceso de desarrollo. Por lo tanto, para que el software cumpla sus objetivos de seguridad, estas técnicas deben ser integradas en el ciclo de vida de desarrollo del software(SDLC) mediante la inclusión de determinadas actividades.
Esta integración se conoce como security development lifecycle.

A continuación se presentan las fases del ciclo de vida del desarrollo del software:

Formación y Actualización: En seguridad y programación de código, Se debe contar con la formación adecuada para mantener al equipo de desarrollo al día sobre las novedades relacionadas a la seguridad.
Requerimientos: Asignación de expertos objetivos, criterios, umbrales y evaluación de riesgos. Los objetivos y requisitos de seguridad y privacidad deben ser definidos al iniciar el proceso de desarrollo de la aplicación.
Diseño: Requisitos de diseño, analizar las vías de entrada, modelado de amenazas. Se incluyen las especificaciones funcionales y de diseño en términos de seguridad. Además, también se lleva a cabo el análisis de riesgos y el modelado de amenazas para identificar las amenazas y vulnerabilidades en el software.
Desarrollo: El objetivo es escribir código con la mejor calidad posible.
La calidad está relacionada con la seguridad, en este punto se deben tener en cuenta los siguientes ítems:

- Herramientas de desarrollo.
- Directrices de codificación segura.
- Análisis estático.

Pruebas: Se debe verificar que los diseños del sistema y del código pueden resistir a un ataque. Por lo tanto, en esta fase se llevarán a cabo las tareas siguientes:

Análisis dinámico.

Pruebas de penetración.

Revisión del modelo de amenazas.

Validación: Antes de poner el software en producción se debe probar el plan de respuestas a incidentes y realizar una revisión final de seguridad.

Mantenimiento: Todas las modificaciones deben ser ser gestionadas con el denominado gestión del cambio, unido a los requerimientos de seguridad y privacidad.

Herramientas de Análisis de código

Análisis Estático (SAST)		
Nombre	**Licencia**	**Descripción**
FindBung	open source	Detector defectos (escáner de seguridad básico) para Java.
Bugscam	open source	Analizador sobre IDA. busca llamadas peligrosas en código binario ejecutable.
Compuware DevParther SecurityChecker	comercial	Escáner de seguridad de código para .NET (C# y VB.NET)
CodeScan	comercial	análisis estático de código para ASP y PHP
Coverity Prevent	comercial	Detector de defectos y escáner de seguridad para Android, C/C++, Fortran, Java, JS, .NET, Python, PHP, Rubi
Cqual	académico	Analizador de flujo de datos para C, basado en análisis de tipo
Flawfinder	open source	Escáner de seguridad para lenguaje C
Fortify SCA	comercial	Escáner de seguridad para diversos lenguajes
CodeSonar	comercial	Vulnerabilidades y otros defectos en C/C++y ADA
ITS4	freeware	Busca llamadas a funciones potencialmente peligrosas en código C
Klockwork Insight	comercial	Escáner de seguridad para lenguajes C/C++ y Java

McCabe IQ	comercial	Analizador de flujo de control general y cobertra. Soporta C, C++, C#, Java, Fortran, VB, y otros
Microsfoft FxCop, prefast, CAt.NET/XSSDetect	freeware	Analizadores de código para lenguajes .NET, C/C++
MOPS	académico	Verifica vulnerabilidades en secuencias de llamadas a funciones C
OWASP LAPSE	open source	Escaner simple de seguridad para Java (plugin Eclipse)
Parasoft	comercial	Análisis estático general, con algunas reglas de seguridad, para Java, C/C++, .NET y HTML
Pixy	open source	inyección SQL y XSS para PHP
Pscan, RATS	open source	Busca llamadas a funciones potencialmente peligrosas en código C
PUMA SCAN	freeware	Extensión para Visual Studio, análisis de vulnerabilidades
QARK	freeware	Análisis de vulnerabilidades para aplicaciones android
Análisis Dinámico (DAST)		
Nombre	**Licencia**	**Descripción**
Zed Attack Proxy	open source	Escáner de seguridad para aplicaciones y servicios web
Fortify WebInspect	comercial	Escáner de seguridad para aplicaciones y servicios web
Vega	open source	Escáner de seguridad para aplicaciones y servicios web
Grabber	open source	Escáner de seguridad para aplicaciones y servicios web
IBM Security AppScan	comercial	Escáner de seguridad para C/C++, JAVA/JSP, .NET(C#/VB.NET) y VB6/ASP

FindBugs
un programa que usa análisis estáticos para buscar errores en el código de Java. Es software libre, FindBugs puede ser ejecutado por medio de Línea de comando, ant, GUI Eclipse, Maven, Netbeans, Jenkins, Hudson, IntelliJ.
Tiene 3 categorías de clasificación de errores:
Correctness bug (exactitud): Un aparente error de codificación que dio como resultado un código que probablemente no era lo que el desarrollador pretendía

Bad Practice (Mala práctica): Violaciones de la práctica de codificación recomendada y esencial.
Dodgy (poco fiable): Código que es confuso, anómalo o escrito de una manera que conduce a errores.
http://findbugs.sourceforge.net/

Compuware DevParther SecurityChecker
Diseñado para integrar todas las soluciones de .NET, funciona a través del código en tiempo de ejecución en tres fases operativas diferentes:
Discovery (Descubrimiento): Rutinas de descubrimiento para abrirse camino a través de una aplicación completa por sí mismo.
Analysis (Análisis): Informes de detalles de vulnerabilidad de la aplicación.
Advice *(*Asesoramiento): Consejos detallados de reparación de vulnerabilidades.
http://searchwindevelopment.techtarget.com/tip/DevPartner-SecurityChecker-25-does-just-that-for-ASPNET-sites

CodeScan
Ofrece informes visuales, métricas, líneas de tiempo y reglas de codificación facilitando el seguimiento de las mejoras, Cuenta con integración para Eclipse, IntelliJ, Visual Studio, Jenkins.
https://www.code-scan.com/

Coverity Prevent
Permite analizar rápidamente grandes proyectos que superan las 100 millones de líneas de código, Al ofrecer integraciones con herramientas de desarrollo clave y sistemas de CI/CD, Permite e informa fácilmente sobre la calidad del código y el estado de seguridad, los riesgos, las tendencias y el cumplimiento normativo de los requisitos de seguridad y mercados verticales.
https://scan.coverity.com/

Cqual
Proporciona un mecanismo liviano y práctico para especificar y verificar las propiedades de los programas C, Amplía el sistema de tipos de C con calificadores de tipo definidos por el usuario adicionales, El programador agrega anotaciones de calificador de tipo a su programa en algunos lugares clave, y Cqual realiza la inferencia de calificador para comprobar si las anotaciones son correctas. Los resultados del análisis se presentan con una interfaz de usuario que permite al examinar los calificadores inferidos y sus rutas de flujo.
http://www.cs.umd.edu/~jfoster/cqual/

Flawfinder
Examina el código fuente C/C++ e informa posibles vulnerabilidades de seguridad ordenados por nivel de riesgo, Es muy útil para encontrar y eliminar rápidamente al menos algunos posibles problemas de seguridad.
Los reportes se presentan por medio de salidas de texto o HTML o valores separados por comas (CSV).
https://www.dwheeler.com/flawfinder/

CodeSonar
Permite analizar y validar rápidamente el código, fuente y/o binario, identificando vulnerabilidades graves o errores que causan fallas del sistema, poca confiabilidad, fallas del sistema o condiciones insegura por medio de un flujo de datos unificado y un análisis de ejecución simbólica que examina el cálculo de todo el programa.
https://www.grammatech.com/products/codesonar

ITS4
Busca potenciales desbordamientos de búfer, creado para reemplazar el uso de grep en la auditoría y desarrollo de código. Su análisis es bastante rudimentario; no construye un árbol de análisis sintáctico de un archivo de entrada, sino que lo escanea en busca de llamadas que se sabe que son peligrosas, como strcpy y popen.
http://seclab.cs.ucdavis.edu/projects/testing/tools/its4.html

Klockwork Insight
Amplía la cobertura del análisis más allá de la sintaxis y la semántica, identificando cuestiones críticas de estándares de seguridad, seguridad y codificación incluye integración con Eclipse, Visual Studio y IntelliJ IDEA.
https://www.klocwork.com/

McCabe IQ
Entorno interactivo para administrar la calidad del software a través de análisis estáticos avanzados, basado en la medición de la calidad del software ofrece índices de error más bajos, ciclos de prueba más cortos y menos esfuerzos de mantenimiento del software.
Incluye las siguientes métricas:
Complejidad ciclomática, esencial, del diseño del módulo, de integración, Líneas de código, y halstead.
http://www.mccabe.com/iq_developers.htm

Microsoft FxCop, prefast, CAt.NET/XSSDetect
Analiza conjuntos de códigos administrados e informa sobre los ensamblados, como posibles mejoras de diseño, localización, rendimiento y seguridad. Muchos de los problemas se refieren a violaciones de las reglas de programación y diseño establecidas en las pautas de diseño.

Cuenta con una interfaz gráfica de usuario y una herramienta de línea de comandos adecuada para su uso como parte de procesos de compilación automatizados o integrado con Microsoft Visual Studio .NET como una herramienta externa.
https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx

MOPS
Encuentra errores de seguridad en los programas C y para verificar el cumplimiento de las reglas de programación defensiva, proporciona una infraestructura que permite codificar y verificar las propiedades de seguridad de interés
versión actual: prototipo de investigación.
http://web.cs.ucdavis.edu/~hchen/mops/

OWASP LAPSE
Las vulnerabilidades detectadas por LAPSE+ están relacionadas con la inyección de datos no confiables para manipular el comportamiento de la aplicación.
Se necesitan tres pasos en LAPSE+ para la detección de este tipo de vulnerabilidades:
Vulnerability Source (Fuente de vulnerabilidad): Detección de los puntos de código que pueden ser fuente de un ataque de inyección de datos no confiable.
Vulnerability Sink (Profundización de la vulnerabilidad): Identifica los puntos que pueden propagar el ataque y manipular el comportamiento de la aplicación.
Provenance Tracker (Rastreador de procedencias): Verifica si es posible llegar a una Fuente de vulnerabilidad desde un Vulnerability Sink que realiza la propagación hacia atrás a través de las diferentes asignaciones.
https://www.owasp.org/index.php/OWASP_LAPSE_Project

Parasoft
Ayuda encontrar y corregir rápidamente los defectos del código con un análisis de ruta completa para una detección de violación precisa, admite técnicas de análisis estático preventivo (Patrón) y Detección (Flujo), junto con un conjunto completo de Métricas para la estructura del código, admite la creación de reglas personalizadas con RuleWizard.
Fácil de configurar, fácil de automatizar, no intrusivo y escalable en múltiples equipos.
https://www.parasoft.com/products/ctest#staticanalysis

Pixy
Es una herramienta que se ejecuta en Java y puede escanear código PHP para identificar vulnerabilidades de inyección XSS y SQL. Se considera una de las mejores herramientas para descubrir vulnerabilidades de inyección SQL en codigo PHP.
https://pentestlab.blog/tag/pixy/

Puma Scan
Proporciona análisis de código fuente continuo en tiempo real a medida que los equipos de desarrollo escriben el código. Las vulnerabilidades se muestran de inmediato en el entorno

de desarrollo como corrector ortográfico y advertencias del compilador, evitando que los errores de seguridad entren en sus aplicaciones.
Contiene una base de reglas que permiten configurar la solución de forma segura.
https://www.pumascan.com/

QARK
Diseñado para buscar vulnerabilidades de seguridad de aplicaciones de Android, ya sea en el código fuente o en los APK empaquetados. La herramienta también es capaz de crear APKs y/o comandos ADB desplegables "Proof-of-Concept", capaces de explotar muchas de las vulnerabilidades que encuentra. No es necesario rootear el dispositivo de prueba, ya que esta herramienta se centra en las vulnerabilidades que se pueden explotar en condiciones seguras.
https://github.com/linkedin/qark

Zed Attack Proxy
Ayuda a encontrar automáticamente vulnerabilidades de seguridad en las aplicaciones web. También es una gran herramienta para pentesters con experiencia para usar en pruebas de seguridad manuales.
Es multiplataforma, Traducido a más de 20 idiomas, está en desarrollo activo por un equipo internacional de voluntarios.
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project#tab=Main

Vega
Ayuda a encontrar y validar vulnerabilidades como Inyección SQL, Cross-Site Scripting (XSS), información confidencial divulgada inadvertidamente y otras. Está escrito en Java, basado en GUI, y se ejecuta en Linux, OS X y Windows.
Los módulos de detección de Vega están escritos en Javascript. Es fácil crear nuevos módulos de ataque utilizando la rica API expuesta por Vega.
https://subgraph.com/vega/

Grabber
Detecta vulnerabilidades como Cross-Site Scripting, Inyección de SQL, Comprobación de archivos de respaldo, realiza un análisis híbrido/Crystal ball para la aplicación PHP usando PHP-SAT, tiene un analizador de código fuente de JavaScript que permite evaluación de la calidad o corrección del JavaScript con JavaScript Lint, se ejecuta en sistema operativo Linux.
https://tools.kali.org/web-applications/grabber

IBM Security AppScan
Mejora la seguridad de las aplicaciones web y móviles, la administración del programa de seguridad de las aplicaciones y fortalece el cumplimiento normativo.

Ayuda a identificar riesgos de seguridad recomienda soluciones a los riesgos y permite generar informes, clasifica y prioriza los activos de la aplicación según el impacto del negocio e identifique áreas de alto riesgo disminuyendo la probabilidad de ataques.
https://www.ibm.com/security/application-security/appscan

En el presente trabajo se profundizará en las herramientas de análisis de código que ofrece HP Fortify para el análisis estático y dinámico.

HP Fortify

Fortify ayuda en la construcción de procesos para asegurar que el software que ejecuta el negocio es seguro. Incorpora un programa de aseguramiento de seguridad del software(SSA) el cual ayuda a proteger proactivamente y reducir los riesgos y costos de encontrar y remediar vulnerabilidades, aumenta la productividad de equipos de desarrollo y establece bases para las mejores prácticas de codificación segura.

Ofrece rápidamente un software seguro permitiendo la búsqueda de problemas de seguridad al principio del ciclo de desarrollo. Cuenta con integraciones para los IDE más conocidos, herramientas de compilación, repositorios de código, seguimiento de errores detectando 763 categorías únicas de vulnerabilidades en 25 lenguajes de programación y más de 911,000 API únicas.

La principal desventaja es que se encuentra atado a la versión del IDE lo cual dificulta su integración con el equipo de desarrollo.

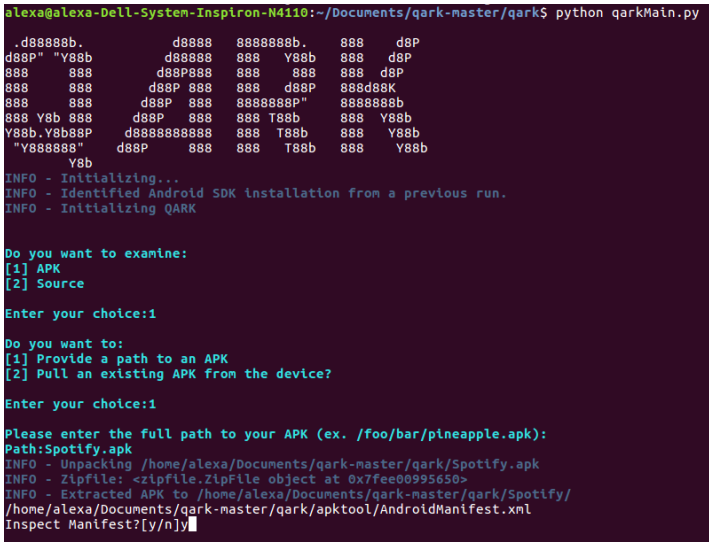
Características clave:

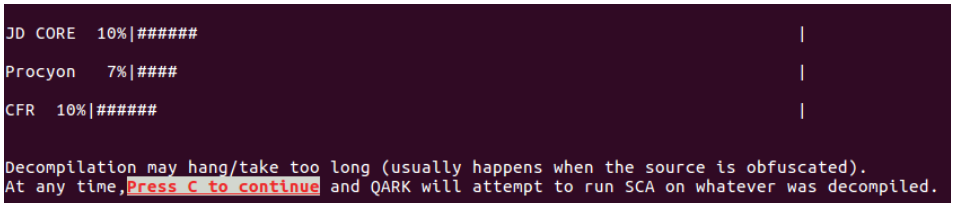
- Reduce el riesgo comercial mediante la identificación de vulnerabilidades que representan la mayor amenaza
- Identifica y elimina vulnerabilidades explotables rápidamente con un proceso repetible.
- Reduce los costos de desarrollo mediante la identificación de vulnerabilidades al principio del SDLC.
- Educa a los desarrolladores en prácticas seguras de codificación.
- Reúne a los equipos de desarrollo y seguridad para encontrar y solucionar problemas de seguridad.

PRÁCTICA: ANÁLISIS DE CÓDIGO ESTÁTICO

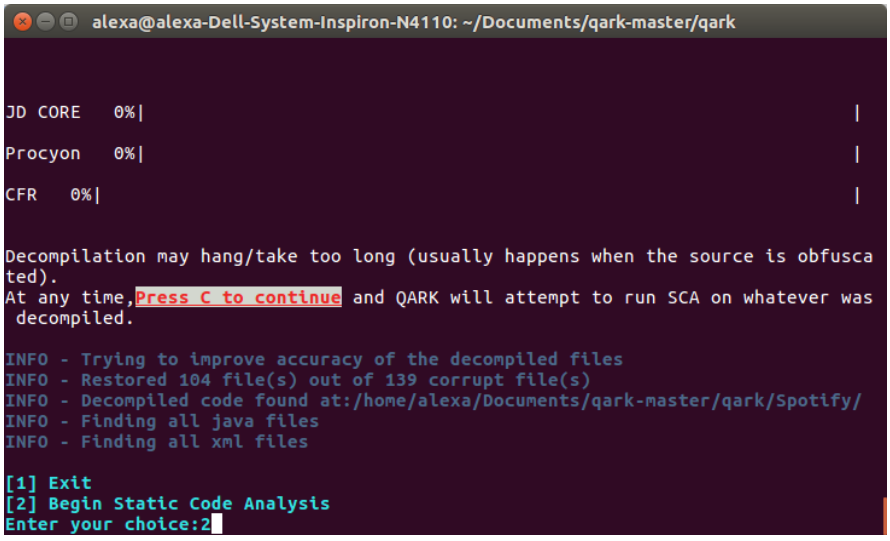
QARK

Para este análisis se utiliza el apk de Spotify, primero Qark intenta decompilar los archivos utilizando herramientas como JD CORE, Procyon, CFR, este proceso puede tardar bastante tiempo y depende si el código está ofuscado

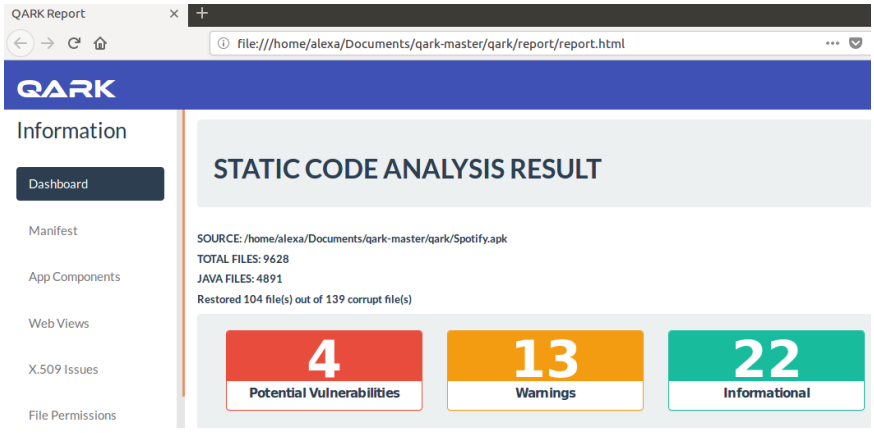




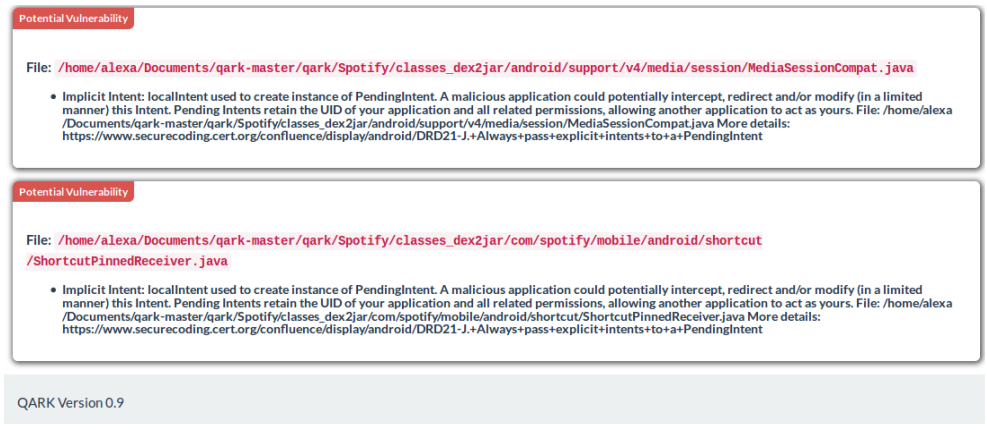
Una vez decompilado el código se inicia el análisis estático

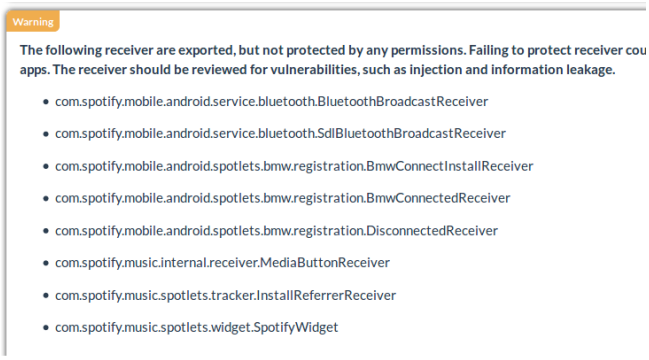


Al finalizar se genera un reporte HTML donde se muestran las vulnerabilidades potenciales, advertencias, mensaje de información.



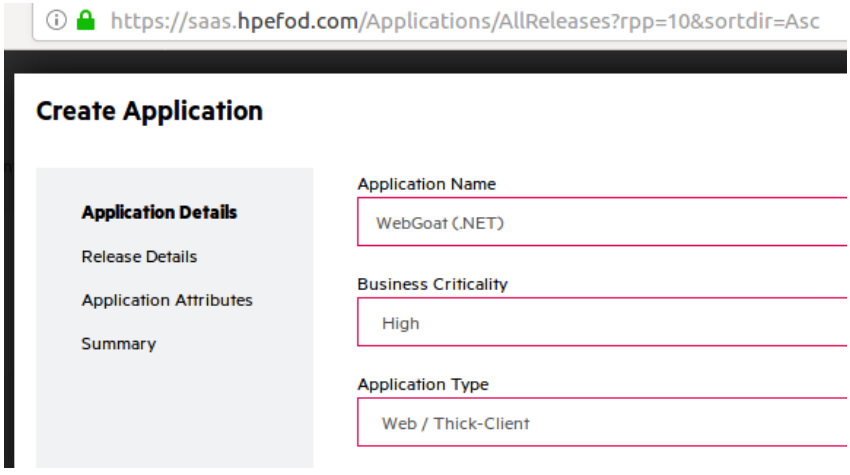
QARK muestra la descripción de la vulnerabilidad, en que archivo se presenta y una página de referencia donde se puede consultar en detalle y visualizar una sugerencia de mitigación del error

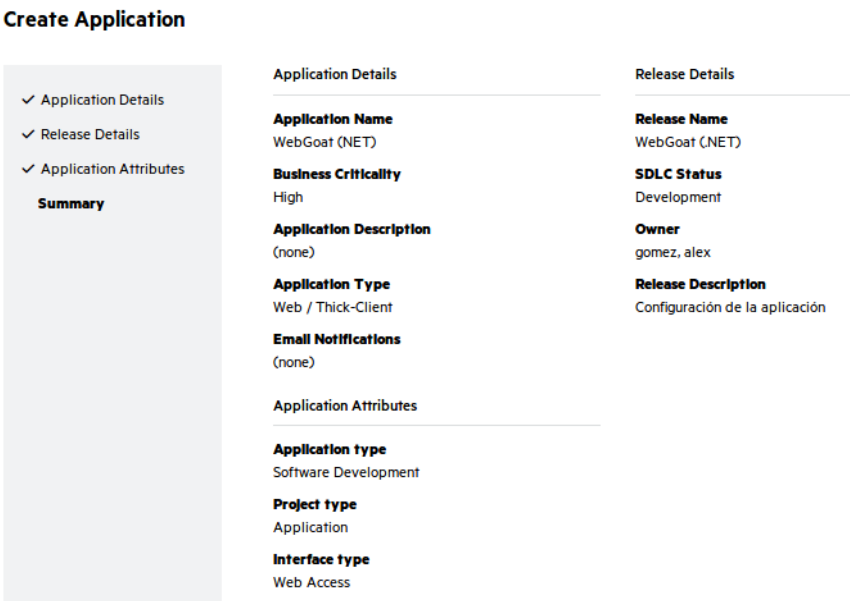




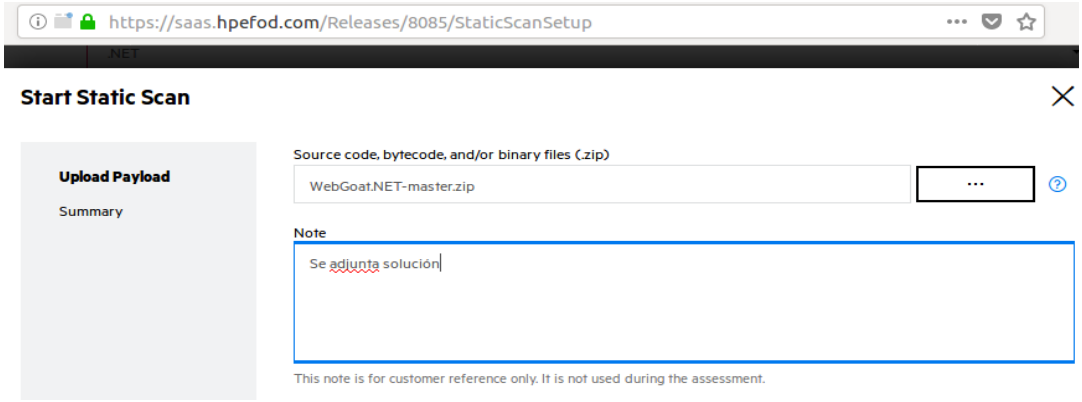
Fortify OnDemand

Esta herramienta permite encontrar problemas de configuración, identifica y prioriza las vulnerabilidades de seguridad en las aplicaciones en ejecución. Simula las técnicas de piratería del mundo real y proporciona un análisis completo de aplicaciones y servicios web. Permite observar la reacción de la aplicación a los ataques en el nivel de código durante las exploraciones dinámicas, facilita la configuración de políticas e informes preconfigurados para todas las principales normativas de cumplimiento relacionadas con la seguridad de las aplicaciones web, incluidas PCI DSS, DISA STIG, NIST 800-53, ISO 27K, OWASP e HIPPAA.
Para este análisis fué realizado con la aplicación [WebGoat.net](), se inicia con la creación y configuración de la aplicación:



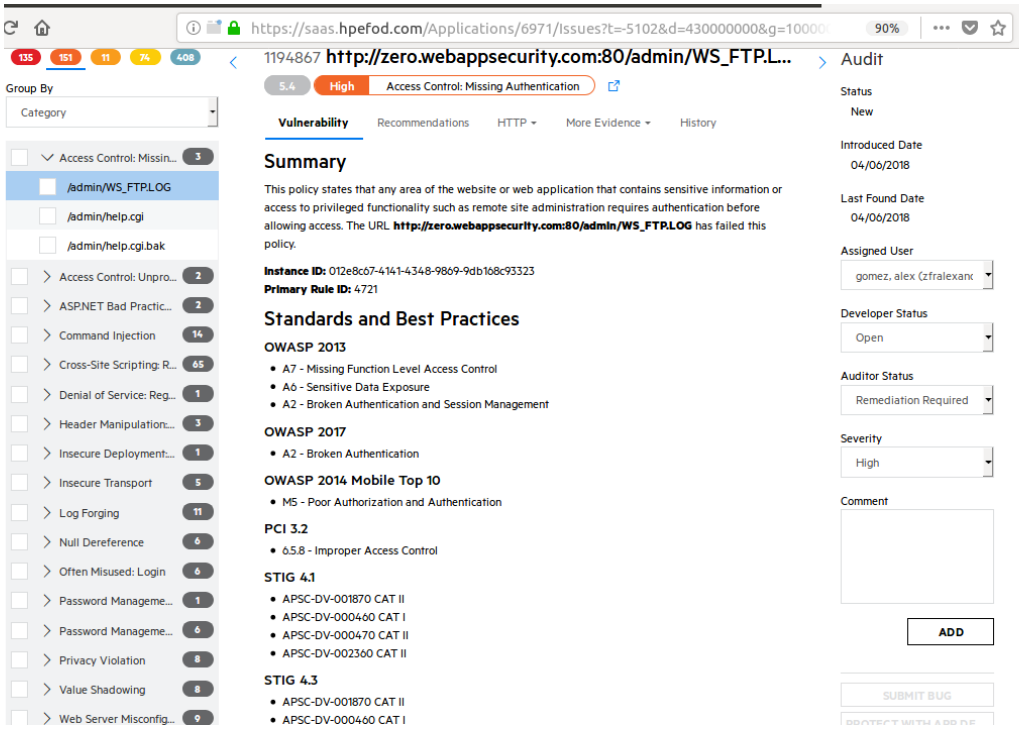


Al ser un análisis estático se debe proporcionar los archivos del código fuente(bin/zip)

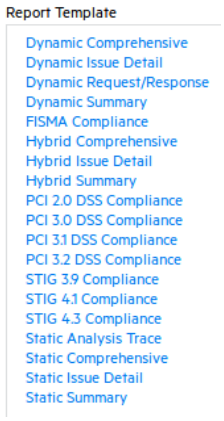


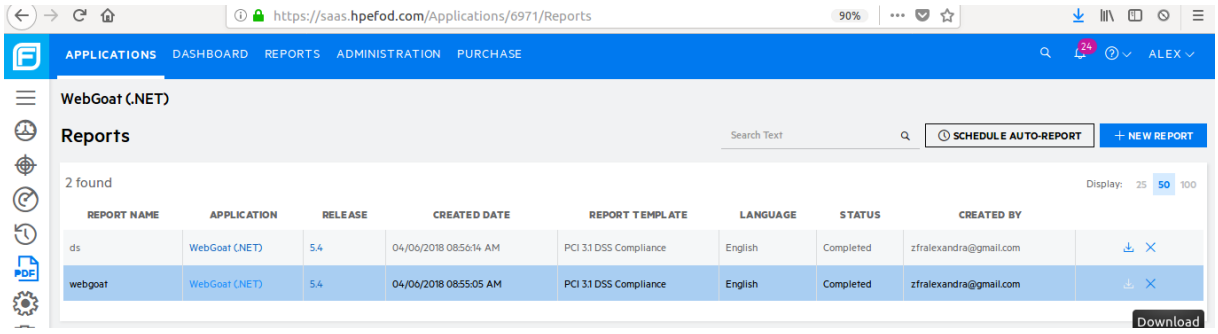
Una vez finalizado el análisis se visualiza el resultado de vulnerabilidades encontradas ordenadas por su criticidad, en total se muestran 371 vulnerabilidades entre críticas, altas, medias y bajas

Vulnerabilidad alta, Control de acceso: Perdida de autenticación
En esta pantalla se da una descripción de la vulnerabilidad o error, se referencia el estándar y se dan algunas recomendaciones para mitigarla.



La pantalla de los reportes estos formatos pueden ser configurados de acuerdo a las necesidades del analista:





El reporte permite visualizar el resumen de las vulnerabilidades encontradas ordenadas por nivel de criticidad, y una gráfica de la probabilidad de impacto.

Executive Summary

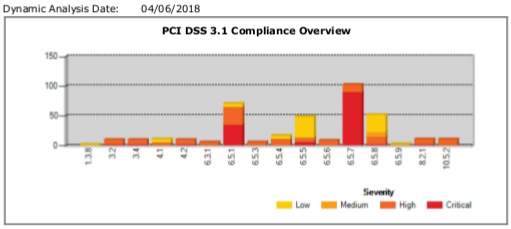
Tenant:	dddd_39_FMA_332817681
Application:	WebGoat (.NET)
Release:	5.4
Business Criticality:	Medium
SDLC Status:	Production
Static Analysis Date:	04/06/2018
Dynamic Analysis Date:	04/06/2018

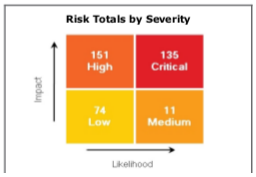
Fortify on Demand Security Rating

★☆☆☆☆

408 issues — Status: Fail

Static: ✔ Dynamic: ✔





Issue Status

New	Existing	Reopened
408	0	0

Se lista cada vulnerabilidad/error encontrado

PCI 3.1 Issue Breakdown

Reported issues in the table may violate more than one compliance requirement. As such, the same issue may appear in more than one row. The total number of unique vulnerabilities is reported in the executive summary table.

Compliance Requirement	Severity				
	Critical	High	Medium	Low	Total
1.3.8 Do not disclose private IP addresses and ...				2	2
2.1 Always change vendor-supplied default acco...					0
2.2.4 Do not use vendor-supplied defaults for s...					0
3.2 Do not store sensitive authentication data a...	2	8			10
3.3 Protect stored cardholder data					0
3.4 Render PAN unreadable anywhere it is stored	2	8			10
3.6.1 Generate strong cryptographic keys					0
4.1 Use strong cryptography and security proto...		5		6	11
4.2 Never send unprotected PANs by end-user ...	2	8			10
5.1 Use and regularly update anti-virus softwar...					0
6.2 Ensure all system components and softwar...					0
6.3.1 Hardcoded Sensitive Information		6			6
6.5.1 Injection Flaws	35	30		6	71
6.5.2 Buffer Overflows					0
6.5.3 Insecure Cryptographic Storage		6			6
6.5.4 Insecure Communications		11		6	17
6.5.5 Improper Error Handling	6	6	2	35	49
6.5.6 High Risk Vulnerabilities		9			9
6.5.7 Cross-Site Scripting (XSS)	90	13			103
6.5.8 Improper Access Control	2	13	7	30	52
6.5.9 Develop and maintain secure systems and...			2	1	3
7.1.2 Restrict access to privileged user IDs					0
7.2 Restrict access by business need-to-know					0
8.1.8 Terminate Idle Sessions					0
8.2.1 Render authentication credentials unreada...	2	9			11
10.2.1 Audit user access to cardholder data					0
10.2.4 Audit Invalid access attempts					0
10.3.4 Verify success or failure indication in audi...					0
10.5.2 Protect audit trail files from unauthorized...		11			11

Para cada error se muestra la descripción del error, página o línea de código afectada, exploit y recomendaciones para mitigar la vulnerabilidad

Issue Detail

Below is an enumeration of all issues found in the project. The issues are organized by priority and category and then broken down by the package, namespace, or location in which they occur.

The priority of an issue can be Critical, High, Medium, or Low.

Issues from static analysis reported on at same line number with the same category originate from different taint sources.

4.1.1 Cross-Site Scripting: Reflected — Critical

CWE-811, CWE-116, CWE-80, CWE-79
OWASP Top 10: A7
PCI 3.2: 6.5.7 Cross-Site Scripting (XSS)

Summary

A Unicode conversion Cross-Site Scripting (XSS) vulnerability was found. This vulnerability is due to an input validation error in the filtration of special HTML characters supplied as Unicode characters. If exploited, an attacker could craft a malicious link containing arbitrary HTML or script code to be executed in a user's browser. Recommendations include modifying the web.config file to use only Unicode code page for output or filtering full-width ASCII characters from all non-trusted data sources.

Explanation

The application fails to properly validate Unicode characters in the "Request Validation" and "HttpServerUtility.HtmlEncode" security mechanisms. If exploited, an attacker could control the Web browser of other Web users who view the page by embedding malicious HTML tags and JavaScript. An attacker could use this technique to steal sensitive information such as credit card numbers, usernames, passwords, files, and session identifiers from the Web users.

Execution

http://[TARGET]/attack.aspx?test=%uff1cscript%uff1ealert('vulnerability')%uff1c/script%uff1e

Vulnerable Systems:

- Microsoft ASP.NET 1.1 SP1
- Microsoft ASP.NET 1.1
- Microsoft ASP.NET 1.0 SP2
- Microsoft ASP.NET 1.0 SP1
- Microsoft ASP.NET 1.0

Recommendation

For Security Operations:

No patch is currently available.

Modify the web.config file to use only Unicode code page for output. To do this, add the following lines to your web.config file:

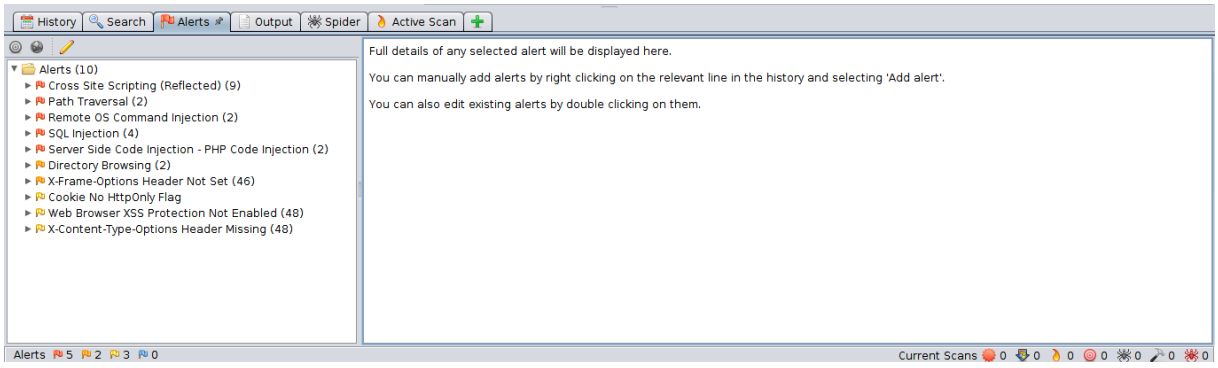
```
<configuration>
  <system.web>
    <globalization responseEncoding="utf-8" />
  </system.web>
```

PRÁCTICA: ANÁLISIS DE CÓDIGO DINÁMICO

OWASP ZEP Attack Proxy(ZAP)

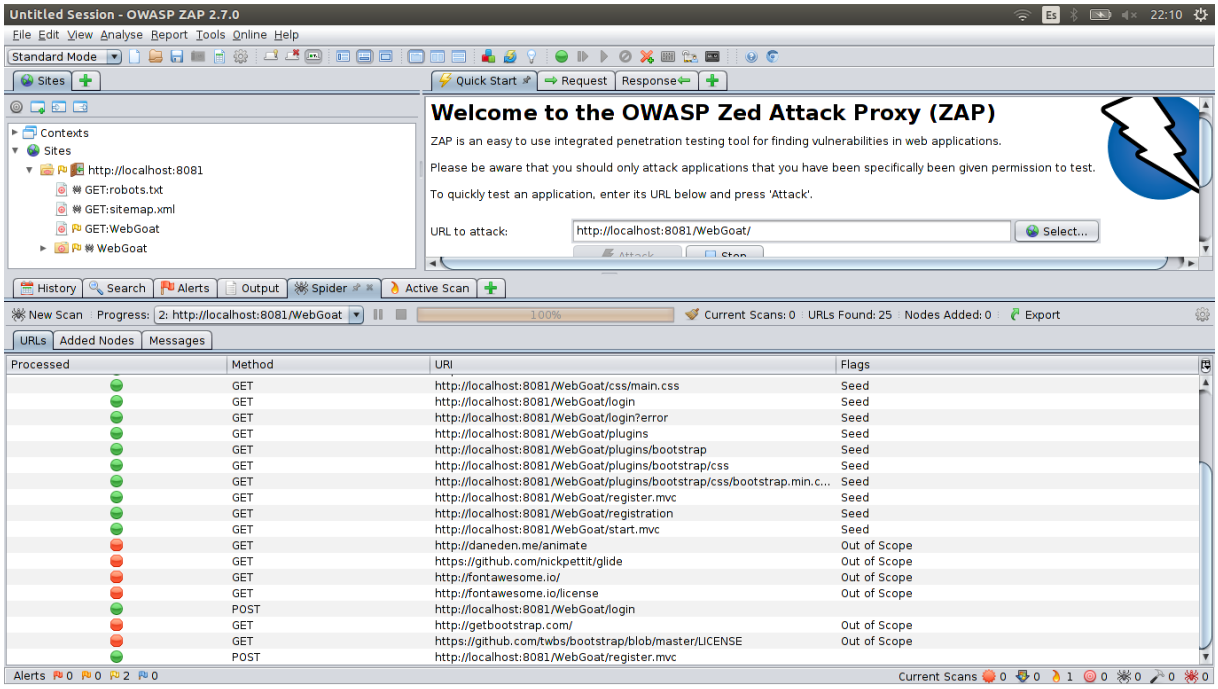
Para este ejemplo se publicó localmente la aplicación [WebGoat.net](), se ejecuta la aplicación y se agrega la dirección URL

En la pestaña Alert se muestran las vulnerabilidades que encuentra a medida que se ejecuta el análisis, en este caso se muestran 163 vulnerabilidades agrupadas por tipo de alerta y nivel de criticidad

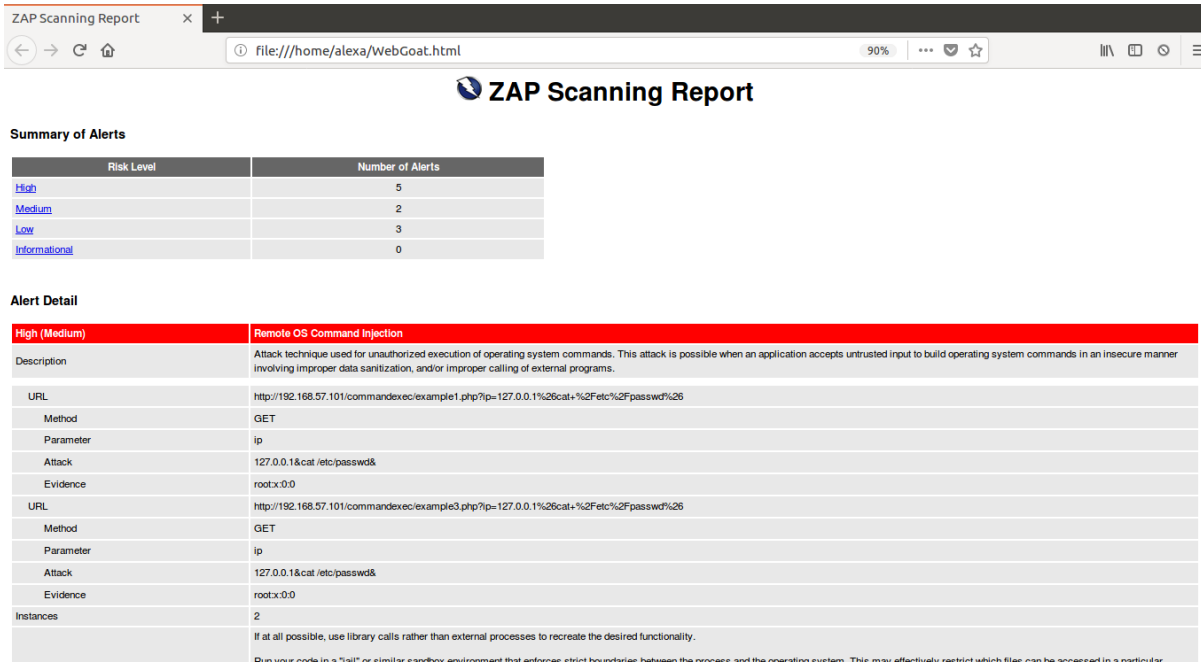


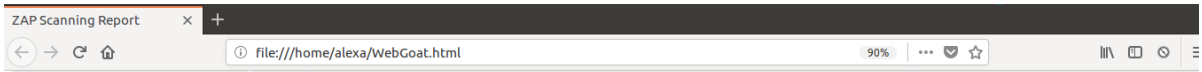
Para cada vulnerabilidad se detalla: URL afectada, tipo de riesgo, nivel de confidencialidad, parámetro y ataque realizado, CWE ID, WESC ID y Descripción

En la pestaña “*Spider*” o Araña se hace el visualiza los recursos (URLs) del sitio



Para ver un mayor detalle de las vulnerabilidades se puede generar un reporte HTML, XML, MarkDown o Json





Solution	If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." When constructing OS command strings, use stringent whitelists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping. Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components. Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address.
Reference	http://cwe.mitre.org/data/definitions/78.html https://www.owasp.org/index.php/Command_Injection
CWE Id	78
WASC Id	31
Source ID	1