UNIVERSIDAD DE GRANADA

SEGURIDAD EN SISTEMAS OPERATIVOS

Evaluación de riesgos:

Escaneo de código fuente. Caja negra y caja blanca.

Autores:

- Juan Emilio García Martínez - juane619@correo.ugr.es
- Adrián Jesús Peña Martínez - adrianprodri@correo.ugr.es
- Julio A. Fresneda García - juliofresnedag@correo.ugr.es

Granada, 10 de Diciembre de 2018

1.- Introducción

Para la realización de este trabajo hemos elegido el dominio relacionado con la ciberseguridad de “Gestión de riesgos”.
En concreto, nos centramos en un tema dentro de la evaluación de riesgos que consideramos que es fundamental, el escaneo del código fuente, una de las bases para que nuestro software o cualquier software sea seguro y un punto que a menudo, a pesar de su enorme importancia, es descuidado debido a la necesidad de acabar el software de manera apresurada para poder cumplir con los períodos de entrega establecidos.

Esto anterior se ha mejorado en los últimos años, numerosas empresas han implementado una gran variedad de tecnologías dirigidas a proteger su infraestructura tecnológica. Hemos visto cómo estas tecnologías cada vez son más usadas: firewalls, IDSs(sistema de detección de intrusos), antiSPAM, AntiVirus, WAFs(Firewall para app web), VPNs, controles de acceso, etc.

Aún así, cada vez se producen más ataques y con un grado mayor de éxito. Por tanto, la pregunta es: si hemos invertido en asegurar nuestra tecnología y con ello hemos mejorado nuestra postura de seguridad, ¿por qué cada vez somos más vulnerables? ¿Por qué tenemos la sensación de que somos más frágiles?

Se han realizado algunos estudios donde se han analizado estos problemas, llegando a la siguiente conclusión: los atacantes están cambiando su patrón de comportamiento. La toma de medidas por parte de las empresas, organizaciones o particulares para asegurar sus activos ha sido fundamentalmente de protección perimetral. Los atacantes han visto limitadas sus posibilidades de intrusión y debido a ello han dirigido su atención al eslabón más débil de la cadena: las aplicaciones web expuestas a todos los usuarios.

Según la WASC (Web Application Security Consortium) más del 90% de vulnerabilidades de las aplicaciones web está en el código fuente.

Un defecto en una aplicación puede dejar a sus operaciones, empleados, clientes y socios expuestos a fraude, robo de identidad y desfiguración del sitio.

2.- Gestión de riesgos. General.

Definamos lo que puede significar “riesgo” en el contexto de la seguridad informática. Podemos decir que *el riesgo es una posible incertidumbre relativa a una amenaza que puede ocurrir, es decir, la posibilidad de que algo negativo ocurra.* Cuando nos referimos a que algo negativo puede ocurrir, lo haremos refiriéndonos a algo negativo sobre los datos de los que disponemos almacenados (información sensible sobre usuarios, sobre nuestra organización o sobre nosotros mismos).

Cuando hablamos de cómo gestionar el riesgo, estamos centrándonos en el procedimiento de realizar una lista con los posibles tipos de riesgos a los que estamos expuestos a la hora de manejar información o datos y asignarle un factor determinado (alto, medio, bajo) para abordar los riesgos según nuestras preferencias.
Dicha lista la podemos obtener identificando, analizando y valorando y clasificando cada riesgo. Una vez obtenida dicha lista debemos definir unas estrategias para el tratamiento de dichos riesgos.

Más en concreto, en la creación de dicha lista, se pueden definir estas cuatro fases:

- Análisis: Determina los componentes de un sistema que requiere protección, sus vulnerabilidades que lo debilitan y las amenazas que lo ponen en peligro, con el resultado de revelar su grado de riesgo.

- Clasificación: Determina si los riesgos encontrados y los riesgos restantes son aceptables.

- Reducción: Define e implementa las medidas de protección. Además sensibiliza y capacita los usuarios conforme a las medidas.

- Control: Analiza el funcionamiento, la efectividad y el cumplimiento de las medidas, para determinar y ajustar las medidas deficientes y sanciona el incumplimiento.

Todo el proceso está basado en las llamadas políticas de seguridad, normas y reglas institucionales, que forman el marco operativo del proceso, con el propósito de:

- Potenciar las capacidades institucionales, reduciendo la vulnerabilidad y limitando las amenazas con el resultado de reducir el riesgo.

- Orientar el funcionamiento organizativo y funcional.

- Garantizar comportamiento homogéneo.

- Garantizar corrección de conductas o prácticas que nos hacen vulnerables.

- Conducir a la coherencia entre lo que pensamos, decimos y hacemos.

En la gestión de riesgos debemos tener en cuenta la probabilidad de que sucedan cada uno de los posibles problemas, de esta forma, podemos priorizar los problemas y su coste potencial desarrollando un plan de acción adecuado.

La motivación para llevar a cabo una correcta gestión del riesgo suele ser económica, aunque también es muy importante la privacidad de los datos que puedan ser revelados o la importancia personal para uno mismo de datos que puedan ser perdidos para siempre.
Es por esto anterior que debemos poder obtener una evaluación económica del impacto de estos sucesos, como por ejemplo, contrastar el costo de la protección de la información en análisis contra el costo de volverla a producir (en caso de poder).

Es importante conocer qué se quiere proteger, dónde y cómo, asegurando que el coste invertido en esto anterior, obtengamos beneficios efectivos. Para esto, debemos identificar cada recurso a proteger como puede ser el hardware involucrado en guardar la información, software, accesorios, etc. y las amenazas a las que se encuentran. Por ejemplo, si tenemos información guardada en discos duros encerrados en un armario, deberíamos proteger el acceso a dicho armario, posibles destrozos sobre el armario, etc..
El principal problema que nos encontramos a la hora de gestionar el riesgo, es el coste de dicha acción. Normalmente, el presupuesto dedicado a la seguridad de la información es limitado, es por este motivo que debemos priorizar correctamente a la hora de mitigar.
Si bien es cierto que la evaluación de riesgos puede ser personalizada para cada organización, se pueden formular varias preguntas generales que ayudan a la identificación de lo anteriormente dicho:

- "¿Qué puede ir mal?"
- "¿Con qué frecuencia puede ocurrir?"
- "¿Cuáles serían sus consecuencias?"
- "¿Qué fiabilidad tienen las respuestas a las tres primeras preguntas?"
- "¿Se está preparado para abrir las puertas del negocio sin sistemas, por un día, una semana, cuanto tiempo?"
- "¿Cuál es el costo de una hora sin procesar, un día, una semana...?"
- "¿Cuánto, tiempo se puede estar off-line sin que los clientes se vayan a la competencia?"
- "¿Se tiene forma de detectar a un empleado deshonesto en el sistema?"
- "¿Se tiene control sobre las operaciones de los distintos sistemas?"
- "¿Cuantas personas dentro de la empresa, (sin considerar su honestidad), están en condiciones de inhibir el procesamiento de datos?"
- "¿A que se llama información confidencial y/o sensitiva?"
- "¿La información confidencial y sensitiva permanece así en los sistemas?"
- "¿La seguridad actual cubre los tipos de ataques existentes y está preparada para adecuarse a los avances tecnológicos esperados?"
- "¿A quien se le permite usar que recurso?"
- "¿Quién es el propietario del recurso? y ¿quién es el usuario con mayores privilegios sobre ese recurso?"
- "¿Cuáles serán los privilegios y responsabilidades del Administrador vs. la del usuario?"
- "¿Cómo se actuará si la seguridad es violada?"

Finalmente, una vez obtenida la lista de cada riesgo posible, se puede realizar un resumen de cada uno de los riesgos de este tipo:

Tipo de Riesgo	Factor
Robo de hardware	Alto
Robo de información	Alto
Vandalismo	Medio
Fallas en los equipos	Medio
Virus Informáticos	Medio
Equivocaciones	Medio
Accesos no autorizados	Medio
Fraude	Bajo
Fuego	Muy Bajo
Terremotos	Muy Bajo

3.- ANÁLISIS DEL CÓDIGO FUENTE

Cuando hablamos de análisis de código fuente nos referimos al análisis de este con el fin de encontrar errores en su funcionamiento, comportamientos no esperados y por supuesto fallos de seguridad que pueden ir desde vulnerabilidades en el código hasta fallos en el funcionamiento del mismo, a partir de este momento hablaremos de análisis estático de software.

Además de esto, permite conocer el grado de inseguridad de tus aplicaciones ante una intrusión, ayudando a mejorar la calidad de la configuración de las aplicaciones de las estructuras informáticas.

Un defecto en una aplicación puede dejar a sus operaciones, empleados, clientes y socios expuestos a fraude, robo de identidad, divulgación de información sensible y desfiguración del sitio.

En la actualidad, la mayoría de los problemas de seguridad se encuentran en las aplicaciones. El análisis del código fuente, provee detección precisa y recomendaciones claras, que permitirán aumentar el nivel de seguridad y evitar ser otra víctima más de las amenazas digitales.

Y sea código desarrollado por uno mismo o uno de terceros, antes de desplegar dicho software en producción es conveniente realizar análisis de vulnerabilidades de código fuente, tanto para aplicaciones móviles, como para aplicaciones de escritorio.

Estas son las principales razones y beneficios de ejecutar un análisis del código fuente:

- Reduce costos: el costo de arreglar una vulnerabilidad se dispara exponencialmente cuanto más avanzado esté en el ciclo de vida de desarrollo de software. Por ello es fundamental la identificación temprana de las deficiencias en seguridad.

No realizar un análisis al código fuente, puede derivar en problemas legales, pérdida de imagen corporativa, robos y fraudes. Invertir en este análisis tempranamente reduce considerablemente los costos y dolores de cabeza.

- Evita muchos problemas: además de los impactos propios de un ataque informático surgen diversas implicaciones, tanto legales ya que existen normativas internacionales que protegen la información de las personas, como también la desconfianza y pérdida de imagen corporativa de la organización.

Entre los problemas específicos que resuelve el análisis de código, están las inyecciones de código, que permiten reinterpretar el código del programa, modificando su comportamiento. La divulgación de información sensible por canales inseguros, por ejemplo sucede cuando no se utiliza cifrado de datos o el usado es débil.

- Es claro, específico y rápido: este tipo de proyectos son de rápida ejecución, y por ello el cliente obtiene resultados en corto tiempo, lo que le permite validar la seguridad del

software que está desarrollando o adquiriendo, y en función de estos resultados poder hacer un Plan de Remediación de las vulnerabilidades encontradas.

- Simula ataques reales: el análisis automático estático y dinámico proporciona una simulación de ataques muy real.

Lo recomendable es ejecutar tanto análisis estático, el cual analiza el código fuente y compilados en busca de riesgos de seguridad, como análisis dinámicos, el cual escanea aplicaciones web ejecutándose, trabaja enviando errores conocidos y fuzzing vía request http y analiza las respuestas.

Otros de los beneficios que nos aporta el análisis de código fuente tenemos los siguientes:

- La Detección de las vulnerabilidades de las aplicaciones.
- Conocer las acciones a seguir para corregir las vulnerabilidades encontradas.
- Mejorar la seguridad de los Sistemas de Información.
- Mejorar la eficiencia de la empresa u organización.
- Prevenir vulnerabilidades en futuros desarrollos

Ahora veamos las limitaciones o problemas que podemos encontrarnos a la hora de realizar un análisis de nuestro código fuente:

- Respecto a auditorías manuales:
  - Costo
  - Tiempos de Espera
  - Desarrollo Terminado

- Respecto al análisis de vulnerabilidades (Vulnerability Assessment):
  - No auditan todo el aplicativo
  - Son menos precisas
  - Pueden incurrir en degradaciones del servicio

- Limitaciones Comunes
  - No contemplan vulnerabilidades a futuro
  - No contemplan actualizaciones de software.

Corregir las vulnerabilidades en la fase de desarrollo reduce el costo de solucionarlas en las fases de pruebas o de producción.

Conclusión:

En el mundo que hoy nos toca vivir, es imprescindible evaluar correctamente los riesgos de seguridad que afectan sus inversiones, esto proporciona mayor valor y evita costos ocultos e indeterminados que podrían surgir si la aplicación/información es afectada. Además como vimos, no todo pasa por costos monetarios, sino por la desconfianza de clientes y litigios legales que le harían perder imagen y tiempo.

3.1. Tipos de análisis del código fuente

Disponemos de dos principales tipos respecto al análisis del código fuente: análisis dinámico de código y de análisis estático. ¿Cuál de los dos tipos de análisis es más conveniente?

A grandes rasgos, se puede afirmar que el análisis estático presenta frente al dinámico la ventaja de que se hace sin ejecutar el código. Como no necesita de esa ejecución, el análisis estático permite detectar errores en una fase muy temprana de la escritura. Así se ahorra mucho tiempo en fases posteriores del desarrollo. El problema más grave que ofrece en cambio, es que puede arrojar positivos que no lo son y cuya falsedad sólo se verá durante la ejecución del código.

El análisis dinámico de código se realiza mientras el código se está ejecutando. Es más lento y necesita un proceso completo de testeo. Sin embargo, permite ver muchos errores que quedan ocultos en un análisis estático.

En cada uno de los tipos de análisis hay distintas pruebas para lograr obtener información suficiente sobre el comportamiento real del software que puedan inducir a un fallo.

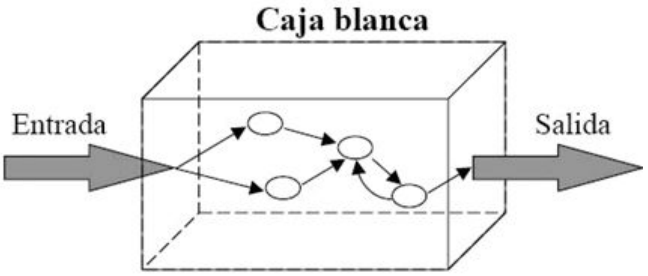
3.1.1. Analisis del codigo fuente dinámico

Como hemos comentado, este tipo de análisis se realiza mientras el código se está ejecutando. Para que el análisis dinámico resulte efectivo, el programa a ser analizado se debe ejecutar con los suficientes casos de prueba como para producir un comportamiento interesante.

Dentro del análisis dinámico de código fuente tenemos dos vertientes bien diferenciadas, estas son el análisis de caja negra y el análisis de caja blanca.

Estos nombres son muy descriptivos pues el análisis de caja negra se basa en analizar las entradas que recibe el software y las salidas que este produce para así poder detectar posibles anomalías, en este análisis no tenemos en cuenta el funcionamiento interno del software, es como si este fuese una caja opaca a través de la cual no podemos ver:

Por otro lado, tenemos el análisis de caja blanca, que es todo lo contrario, se basa en el análisis del funcionamiento interno del sistema, también llamado pruebas de caja de cristal o pruebas estructurales.
Se centran en los detalles procedimentales del software, por lo que que su diseño está fuertemente ligado al código fuente. Lo que se hace en estos análisis es escoger distintos valores de entrada para examinar cada uno de los flujos de ejecución del software que estamos analizando y de este modo asegurarse de que se devuelven los valores de salida adecuados:



Puesto que estas pruebas están basadas en una implementación concreta, si esta se modifica, las pruebas también deberán de ajustarse a esta nueva versión del software.

Sea cual sea el tipo de prueba que se realice, los análisis dinámicos de código necesitan de un equipo de control de calidad que los lleve a cabo. La independencia entre los desarrolladores y el equipo de pruebas debe ser total para evitar fallos durante el proceso.

3.1.1.1. Herramientas de análisis dinámico

Algunas herramientas para realizar este tipo de análisis son:

- Valgrind: conjunto de herramientas libres que ayuda en la depuración de problemas de memoria y rendimiento de programas. Dentro del conjunto de herramientas que ofrece Valgrind como Memcheck, Addrcheck, MAssif, Helgrind, Cachegrind, etc.. podemos destacar Memcheck, ya que permite detectar uso de memoria no inicializada, lectura/escritura de memoria que ha sido previamente liberada, lectura/escritura fuera de los límites de bloques de memoria dinámica, fugas de memoria, etc..

- Intel Inspector: herramienta parecida a Valgrind pero más centrada en aplicaciones desarrolladas en C y C++. Los chequeos que este programa realiza son orientados a detectar fugas de memoria, punteros desapuntados, variables no inicializadas, uso de inválidas referencias de memoria, chequeos de la pila, etc..

Hay muchísimas herramientas más que se comportan de manera parecida.
El precio a pagar por el uso de estas herramientas es una notable pérdida de rendimiento; los programas se ejecutan entre cinco y veinte veces más lento al usarlas, y su consumo de memoria es mucho mayor. Por ello normalmente no siempre se ejecuta un programa en desarrollo usando estas herramientas, sino que se usa en situaciones concretas cuando se está buscando un error determinado. Se trata de verificar que no haya errores ocultos.

3.1.2.- Analisis del codigo fuente estático

Como hemos comentado antes, el análisis estático del código es el proceso de evaluar el software sin ejecutarlo.
En la inmensa mayoría de casos se realiza sobre alguna versión de código fuente y en otros casos se realiza sobre el código objeto, en función del lenguaje haremos uno u otro o los dos.

Un analizador de código estático, recibe el código fuente de nuestro programa, lo procesa e intenta averiguar qué es lo que queremos que haga, dándonos sugerencias con las que poder mejorar dicho código. La pregunta es, ¿cómo hace esto?

Estas herramientas incluyen, por un lado, analizadores léxicos y sintácticos que procesan el código fuente y, por otro, un conjunto de reglas que aplicar sobre determinadas estructuras. Si nuestro código fuente posee una estructura concreta que el analizador considere como "mejorable" en base a sus reglas nos lo indicará y nos sugerirá una mejora.

Lo que nosotros obtenemos ejecutando dichos análisis es facilidad de mantenimiento en un futuro minimizando la parte técnica de nuestro software ya que dichos analizadores se centran en encontrar partes del código que puedan:

- reducir el rendimiento.
- provocar errores en el software
- complicar el flujo de datos
- tener una excesiva complejidad
- suponer un problema en la seguridad.

Como hemos comentado antes, existen varias pruebas que se realizan respecto a un tipo de análisis de código o a otro. En el caso del análisis estático tenemos los análisis automáticos que realizan los programas que hemos comentado y los análisis manuales que realizan personas, persiguiendo unos objetivos concretos:

- El análisis realizado por un programa de ordenador, o análisis automático, reduce la complejidad que supone detectar problemas en la base del código ya que los busca utilizando a unas reglas que tiene predefinidas.
- El análisis realizado por una persona, o análisis manual, se centra en apartados propios de nuestra aplicación en concreto como, por ejemplo, determinar si las librerías que está utilizando nuestro programa se están utilizando debidamente(versiones, funciones *deprecated*, etc..) o si la arquitectura de nuestro software es la correcta.

Ambos, por tanto, son complementarios. El análisis automático se centra únicamente en facetas de más bajo nivel como la sintaxis y la semántica del código, funcionando este análisis en cualquier tipo de aplicación, mientras que el análisis manual se ocupa de facetas

de más alto nivel como, por ejemplo, la estructura de nuestra aplicación o su manera de trabajar con otros elementos externos como las librerías.

Debemos, por tanto, unificar ambas para poder mejorar la base de nuestro código fuente, lo cual repercutirá en una mejora tanto del desarrollo como del mantenimiento de nuestro software antes incluso de llegar a ejecutarlo.

Cuándo debemos realizar este tipo de análisis? Cada vez que escriba una línea de código? Cuando termine el desarrollo completo? Cuando termine una nueva funcionalidad?

La respuesta a estas preguntas no están nada claras pero tenemos que tener en cuenta que este tipo de análisis es un medio y no un fin, para mejorar nuestro código fuente, por tanto, debemos usarlo solo únicamente como apoyo.

Esto anterior quiere decir que debemos periodizar el cuando realizamos un análisis automático o uno manual, dependiendo del grueso del código o de lo crítica que sea la parte a analizar. Es recomendable realizar un análisis manual cada vez que vayamos a crear una nueva funcionalidad en nuestro software, haciéndonos ver si la arquitectura de nuestro software nos permite implementarla con facilidad, si disponemos de los elementos necesarios para implementarla como librerías, etc..

Como recomendación nuestra y de manera natural, podemos realizar este tipo de análisis cuando veamos que nuestro software va mal, es decir, que nos esté costando mucho desarrollar nuevas funcionalidades o modificar las que ya tenemos, o vemos que las piezas de nuestro software no encajan bien entre ellas.

El análisis automático, en cambio, puede ser realizado con una mayor periodicidad ya que no requiere de intervención humana y, lo que es mejor, puede ser programado y repetido tantas veces como sea necesario. Además obra con objetividad, siempre nos va a devolver la misma respuesta ante el mismo código fuente de entrada.

3.1.2.1. Actividades complementarias al análisis estático del código

Además de todo lo visto anteriormente podemos aplicar más técnicas para conseguir mejorar el código fuente de nuestro programa y gracias a esto mejoraremos también la calidad del software final que usarán los usuarios.

Estas técnicas, al contrario que la vista anteriormente, se centran en analizar el código mientras este se encuentre en ejecución, parecido a como vimos con el análisis de caja negra. Estás se basan en:

- **Tests**: Son una serie de pruebas que permiten verificar y comprobar que el software cumple con los objetivos y con las exigencias para las que fue creado y además que no

se violan las políticas de seguridad establecidas en el software. Su misión es la de encontrar errores antes de que el software final sea utilizado por los usuarios y los hay de varios tipos, por ejemplo si queremos poner un software en producción primero deberemos de crear una batería de test y si los pasa entraría en producción, además podemos hacer que el proceso de testeo sea automático con herramientas como Jenkins y que si la nueva versión pasa estos tests automáticos entonces está entre como nueva release.

- **Profiling**: Con estas técnicas lo que intentaremos será medir el rendimiento del software mientras esté se ejecuta y es probado, determinaremos qué recursos se usan en cada momento y veremos si aparece alguna anomalía para así poder solucionarla.

3.1.2.2. Herramientas de análisis estático

En cuanto a estos programas tenemos varios y algunos de estos no revisan el código fuente como tal sino las vulnerabilidades de nuestro software y nos advierte sobre estas. Algunos de los softwares usados para el análisis de programas son:

- **Balbix** puede ser técnicamente un gestor de vulnerabilidades, pero lo hace mucho mejor y rompe los límites de su categoría. Es capaz de analizar cada tipo de activo vulnerable que se encuentra en una red, qué tipo de datos contiene, cuántos usuarios interactúan con él, si es o no un archivo público, y otros factores para determinar su importancia para una organización. Después compara cada vulnerabilidad con las fuentes de amenazas activas y predice la probabilidad de que se produzca una brecha en un futuro próximo, así como la pérdida o el daño a la empresa en caso de que se explote con éxito.

- **BluVector** ofrece detección y respuesta avanzadas, e incluso detección de amenazas, todo ello a velocidades de crucero. BluVector funciona casi de inmediato, pero también tiene capacidades de aprendizaje profundo de la máquina, por lo que se vuelve aún más inteligente con el tiempo. Aprenderá las complejidades de cada red que la implementa, ajustando sus algoritmos y motores de detección de la manera más adecuada para el entorno.

- En su núcleo, **Bricata** ofrece protección avanzada IPS/IDS con múltiples motores de detección y fuentes de amenazas para defender el tráfico de la red y los activos principales. Pero va un paso más allá, ya que añade la capacidad de cazar amenazas basadas en eventos o simplemente en anomalías.

- **Cloud Defender** es una herramienta fácil de usar que permite al personal local de TI inspeccionar sus desarrollos en la nube para buscar pruebas de amenazas o brechas ocultas. Pero también se puede utilizar en un modelo SaaS, ya que el equipo de ciberseguridad de Alert Logic se hace cargo de la mayoría de las funciones de ciberseguridad basadas en la nube.

- Implementado como un dispositivo virtual local, **Triage** se conecta con casi cualquier programa de correo electrónico corporativo y ayuda a administrar las respuestas a los informes de los usuarios sobre sospechas de phishing. El triaje sigue evolucionando, pero aún hoy representa una de las defensas más avanzadas contra el phishing.

- **Contrast Security** tiene una de las soluciones más elegantes que existen para la seguridad de las aplicaciones. El secreto está en el uso de la instrumentación del código de bytes, una característica de Java que se utiliza para ayudar a integrar programas y funcionalidades de las aplicaciones durante el desarrollo.

- **Digital Guardian Threat Aware Data Protection Platform** está a la vanguardia de los esfuerzos para contrarrestar las amenazas avanzadas, ya que ofrece seguridad de endpoints lista para implementar localmente en las instalaciones o en modalidad como servicio, y con el nivel de automatización adecuado para la organización que lo implementa.

- La plataforma **EnSilo** ofrece la tradicional protección de endpoints junto con la capacidad de ofrecer protección después de una infección. También puede atrapar amenazas, mantenerlas en su lugar y hacerlas inofensivas hasta que un cazador de amenazas pueda llegar para investigarlas.

Los análisis realizados por estas herramientas varía de aquellos análisis que solo tienen en cuenta el comportamiento de las instrucciones y declaraciones individuales a los que se incluye el código fuente del programa en su análisis.

Estos programas son tremendamente sofisticados y son una solución muy eficaz para el análisis de software, mucho más que el análisis de un humano observando el código fuente directamente.

Los usos de la información obtenida de un análisis varían desde indicar posibles errores de codificación hasta demostrar matemáticamente con métodos formales ciertas propiedades acerca de un programa dado (por ejemplo, que su comportamiento coincide con el de su especificación) dependiendo de qué programa se utilice para el análisis.

Muchos de estos softwares además no necesitan el código fuente pues se pone del lado de un posible atacante y aplica ingeniería inversa con el software a probar para ver si de este modo un atacante sería capaz de descubrir y explotar alguna posible vulnerabilidad.

El análisis de software de las formas descritas entra dentro de los objetivos del software de calidad.

4.- Bibliografía

Análisis estático software:
https://es.wikipedia.org/wiki/An%C3%A1lisis_est%C3%A1tico_de_software

Análisis dinámico software:
https://es.wikipedia.org/wiki/An%C3%A1lisis_din%C3%A1mico_de_software
Aitor, 2018
https://go4it.solutions/es/blog/analisis-dinamico-de-codigo-vs-analisis-estatico

Softwares de análisis:
John Breeden, 2018
https://cso.computerworld.es/tendencias/el-mejor-software-de-seguridad-de-2018

Análisis estático de código:
https://www.isecauditors.com/auditoria-de-codigo-fuente
https://software.intel.com/en-us/get-started-with-inspector

RFC 1244: Site Security Handbook. J. Reynolds - P. Holbrook. Julio 1991

Exposito, Raul, Análisis Estático del Código, Primera, Web, 2009
https://raulexposito.com/documentos/analisis-estatico-codigo/