UNIVERSIDAD DE GRANADA

Trabajo de Fin de Máster

Máster Profesional en Ingeniería Informática

Sistema de ciberdefensa dinámica basada en algoritmos evolutivos para la prevención de ataques informáticos

Autor
Ernesto Serrano Collado

Tutor
Juan Julián Merelo Guervós

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, 6 de diciembre de 2024

Sistema de ciberdefensa dinámica basada en algoritmos evolutivos para la prevención de ataques informáticos

Ernesto Serrano Collado

Resumen

Palabras clave: *seguridad informática, ciberdefensa, algoritmos genéticos, software libre*

Además de realizar una labor determinada de forma eficiente, los servicios informáticos deben ser capaces de evitar los ataques y de detectar los que haya. Una técnica de defensa consiste en convertirse en un objetivo móvil, que varíe el perfil de forma que los atacantes no lo reconozcan.

Mediante algoritmos evolutivos trataremos de configurar diferentes servicios de forma que se maximice la diversidad, a la vez que se optimice la seguridad y las prestaciones.

Evolutionary Based Moving Target Cyberdefense System

Ernesto Serrano Collado

Extended abstract

Keywords: *computer security, cyber defense, genetic algorithms, free software.*

Cyber attacks are one of the biggest problems for many organizations. That organizations are investing so many resources in detecting cyber-attacks, but they still have serious difficulties to prevent them.

The common way to perform an attack is using a list of known vulnerabilities. Many of these vulnerabilities can be caused by a bad configuration.

A simple protection against computer security threats can be implemented by properly tuning existing system software without the needed of expensive security solutions.

This project shows a low-cost technique to prevent computer attacks. This technique consists of constantly modifying the configuration of a server.

Using a genetic algorithm, many different possible configurations are mutated to find an optimal solution. This optimal configuration is regularly applied, so the systems information previously collected by a potential attacker is no longer effective, achieving a simple protection layer while optimizing the service configuration.

Yo, **Ernesto Serrano Collado**, alumno de la titulación **Máster Profesional en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, declaro que el presente Trabajo de Fin de Máster es original, no habiéndose utilizado fuentes sin ser citadas debidamente. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la Normativa de Evaluación y de Calificación de los estudiantes de la Universidad de Granada de 20 de mayo de 2013, *esto conllevará automáticamente la calificación numérica de cero [...] independientemente del resto de las calificaciones que el estudiante hubiera obtenido. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagien.*

Asimismo, autorizo la ubicación de la siguiente copia de mi Trabajo de Fin de Máster (*Sistema de ciberdefensa dinámica basada en algoritmos evolutivos para la prevención de ataques informáticos*) en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Además, este mismo trabajo está publicado bajo la licencia **Creative**

Y para que así conste firmo el presente documento.

Fdo: Ernesto Serrano Collado

Granada, a 6 de diciembre de 2024

D. **Juan Julián Merelo Guervós**, profesor del **Departamento de Arquitectura y Tecnología de los Computadores** de la **Universidad de Granada**.

Informa:

Que el presente trabajo, titulado ***Sistema de ciberdefensa dinámica basada en algoritmos evolutivos para la prevención de ataques informáticos***, ha sido realizado bajo su supervisión por **Ernesto Serrano Collado**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 6 de diciembre de 2024.

El tutor:

Juan Julián Merelo Guervós

Agradecimientos

A Georgia, que algún día será mejor ingeniera que su tío.

Al ZX Spectrum 128K de mis hermanos, porque sin él no habría llegado hasta aquí.

Índice general

Capítulo 1

Introducción

1.1. Motivación

En un mundo interconectado los ciberataques son uno de los mayores problemas para muchas organizaciones. Las organizaciones cada vez invierten más recursos en la detección de ataques informáticos, pero debido al amplio espectro de los mismos todavía tenemos serias dificultades para prevenirlos.

Las pérdidas en la economía mundial causadas por el cibercrimen y el ciberespionaje se estiman en cientos de miles de millones de dólares **herrero_cibercrimen_2015**. Según un estudio del Instituto de Investigación Interregional de Crimen y Justicia de las Naciones Unidas (UNICRI por sus siglas en inglés), el cibercrimen es una de las principales amenazas para la economía mundial pasando el coste asociado al cibercrimen de 400.000 millones de Dólares en 2014 **zappa_cybercrime:_2014** a alcanzar, según la Interpol, una estimacionón de 750.000 millones de Euros sólo en Europa **gil_cuanto_2018**.

Las pérdidas estimadas causadas por actividades cibernéticas maliciosas están dentro de los siguientes criterios **armin_2020_2015**:

- Robo de propiedad intelectual e información comercial confidencial.
- Robo de información sensible, incluida la posible manipulación del mercado.
- Coste de oportunidad que incluye interrupciones en el servicio, y una menor confianza en las actividades en línea.
- Pérdida causada por daños a la reputación de los negocios pirateados.

La forma mas común de realizar dichos ataques es haciendo uso de vulnerabilidades conocidas. Muchas de estas vulnerabilidades pueden ser causadas por una mala configuración o por una combinación inadecuada de parámetros. Además, un servicio determinado puede tener prácticamente infinitas configuraciones posibles, siendo unas menos funcionales y/o vulnerables que otras.

La protección contra multitud de amenazas de seguridad informática puede ser implementada mediante una correcta configuración del software existente sin necesidad de invertir en costosas soluciones de seguridad.

Asimismo, para mejorar el mecanismo de protección contra los ciberataques, el atacante puede ser engañado por un cambio continuo en la configuración de un determinado servicio y en el caso de que el atacante sea capaz de descubrir vulnerabilidades en un programa específico el algoritmo genético habrá cambiado la configuración antes de que el atacante pueda definir un ataque en base a las vulnerabilidades descubiertas.

Encontrar dichas configuraciones correctas, así como similares configuraciones totalmente funcionales, puede ser una tarea inabarcable si se realiza de forma manual por lo que lo ideal sería encontrar una forma de generad dichas configuraciones de forma automática.

Los algoritmos genéticos, que es una heurística de búsqueda, se podrían utilizan para descubrir nuevas configuraciones, seguras y diversas mediante el modelado de una determinada configuración como si fueran cromosomas y las distintas opciones de configuración individuales como si fueran genes **john__evolutionary__2014**. La idea principal de los algoritmos genéticos es que mediante mutación, cruce y selección de dichos cromosomas obtengamos mejores configuraciones. Dichas mutaciones se incorporan al azar, lo que proporciona diversidad.

En este caso concreto, los algoritmos genéticos nos pueden proporcionar una mejor seguridad a través de la diversidad **crouse__improving__2012**.

Todo el código utilizado para la realización de este proyecto así como las diferentes herramientas tienen licencias de código abierto, tanto por motivación ideológica como por seguridad, ya que diversos estudios indican que el Software Libre es mucho mas seguro que el propietario **walia__comparative__2006 mansfield-devine__open__2008 clark__is__2009**.

1.2. Definición del problema

Un atacante suele comenzar su ataque realizando un reconocimiento previo. Luego planea su ataque de acuerdo con la información que encuentra,

por lo tanto, un cambio periódico en la configuración es una ingeniosa forma de hacer que su esfuerzo de reconocimiento no sea efectivo y ayudaría a prevenir un ataque a un costo relativamente bajo.

La técnica del "objetivo móvil" o "Moving Target Defense" es aplicable incluso a elementos hardware como se ha visto en el diseño del procesador Morpheus que es capaz de cambiar su configuración interna cada 50 milisegundos **gallagher_morpheus:_2019**.

Considerando los algoritmos genéticos como una de las líneas de la inteligencia artificial, lo utilizaríamos **tribak_alisis_2012** para alterar la configuración de un servidor de forma regular saboteando así los esfuerzos de recopilación de información realizados por un posible atacante.

Las distintas configuraciones posibles van cambiando (mutando) mediante un algoritmo genético hasta alcanzar una solución óptima. Si dichas configuraciones óptimas son aplicadas de forma regular la información recopilada previamente por un posible atacante ya no es efectiva consiguiendo una capa de protección sencilla y económica a la vez que se optimiza la configuración del servicio **gensch_evolving_2016**.

Es importante asegurarse de que las configuraciones generadas funcionen de una manera correcta, y es importante que dicha configuración sea segura. Es difícil encontrar una configuración segura debido a la gran cantidad de configuraciones para probar, y además de eso, es difícil encontrar la combinación entre las configuraciones que hace que los componentes sean seguros. Para solucionar este problema deberemos contar con una herramienta que nos pueda indicar el nivel de seguridad de una configuración determinada.

1.3. Estructura del proyecto

Antes de pasar a detalles más técnicos, me gustaría detallar el contenido de este proyecto:

- En el *capítulo 1* (**Introducción**) se encuentra una breve introducción a nuestra idea, así como las motivaciones que nos han llevado a realizarla.
- El *capítulo 2* (**Objetivos**) define los objetivos que se quieren alcanzar con este proyecto.
- En el *capítulo 3* (**Antecedentes**) se analiza el estado de arte actual, así como algunas de las tecnologías y paradigmas que utilizaremos en nuestro proyecto.

- En el *capítulo 4* (**Metodología**) está la planificación y desarrollo de cada uno de los apartados del proyecto.
- En el *capítulo 5* (**Resultados**)se detallan todos los resultados obtenidos.
- En el *capítulo 6* (**Conclusiones**) se pueden encontrar las conclusiones finales así como las recomendaciones para futuros trabajos.

Para finalizar se incluye un anexo con el código fuente desarrollado y liberado bajo la licencia libre GPLv3 **free_software_foundation_gnu_2007**. Dicho código fuente también se puede encontrar en la url `https://github.com/erseco/moving_target_defense`.

Capítulo 2

Objetivos

Los objetivos de este proyecto son prevenir ataques informáticos utilizando la técnica del 'objetivo móvil', comprobar si utilizar una heurística de búsqueda, como pueden ser los algoritmos genéticos, puede servir para generar de forma automatizada las distintas configuraciones con la suficiente diversidad atendiendo también a la seguridad del sistema a la hora de generar dichas configuraciones.

Para poder acometer dichos objetivos se ha optado por el desarrollo una herramienta software que mediante el uso de algoritmos genéticos sea capaz de optimizar la configuración de un determinado servicio y que, aplicando dicha configuración de forma regular, la seguridad del servicio se vea incrementada al entorpecer la recopilación de información realizada por un posible atacante **john_evolutionary_2014**.

Para alcanzar estos objetivos realizaremos las siguientes tareas:

- **TAREA-1.** Analizar distintos servicios web candidatos a ser optimizados y securizados.
- **TAREA-2.** Analizar las posibilidades de configuración de los servicios anteriores.
- **TAREA-3.** Cuantificar la seguridad de una determinada configuración.
- **TAREA-4.** Desarrollar una herramienta para prevenir ataques informáticos.

2.1. Alcance de los objetivos

El fin inmediato de este proyecto es conseguir prevenir ataques informáticos ayudando a los administradores de sistemas securizar servidores de una forma sencilla.

Además todo el código así como la documentación resultante se liberará con una licencia libre para que cualquiera pueda hacer uso de las conclusiones y los datos extraídos del análisis.

2.2. Interdependencia de las tareas

Todos las tareas son interdependientes entre sí. En aspectos más relacionados con la realización del proyecto, la tercera tarea (**TAREA-3**) la el que nos brindará el sistema sobre la que trabajar, ya que sienta la base sobre la que aplicar la cuarta tarea (**TAREA-4**). El análisis realizado en las dos primeras tareas (**TAREA-1** y **TAREA-2**) nos indicarán que servidor y herramientas podremos utilizar para realizar el resto de tareas.

2.3. Conocimientos y herramientas utilizadas

Destacar en los aspectos formativos previos más utilizados para el desarrollo del proyecto los conocimientos adquiridos en las asignaturas "Cloud Computing" para el análisis y configuración de los diferentes servicios de red así como todo lo referente a virtualización de sistemas, "Inteligencia Computacional" para el desarrollo del algoritmo genético y "Planificación y Gestión de Proyectos Informáticos" para definir los requisitos y el planteamiento inicial del proyecto, siendo todas ellas del **Máster Profesional en Ingeniería Informática**. También destacar las asignaturas del **Grado en Ingeniería Informática** "Seguridad en Sistemas Operativos" para la parte de seguridad y "Servidores Web de Altas Prestaciones" para la realización de pruebas desde el punto de vista de disponibilidad y carga de trabajo.

Para la realización de cada una de las partes se han usado multitud de herramientas específicas tales como `LaTeX`, `Zotero`, `Docker` `Git` y `OWASP ZAP` entre otras.

Capítulo 3

Antecedentes

En este capítulo vamos analizar el estado de arte actual y las tecnologías candidatas a utilizarse en este proyecto en base a los objetivos que presentamos en el capítulo anterior.

Como ya vimos en la introducción, diversos estudios aseguran que es posible incrementar la seguridad de una configuración en base al uso de algoritmos genéticos, pero la mayoría son demostraciones teóricas sin ninguna implementación real cuantificable **john__evolutionary__2014 romero__sistema__2017 buji__genetic__2017**.

3.1. Herramientas existentes

A pesar de las multiples aproximaciones teóricas **schlenker__deceiving__2018 champagne__genetic__2018**, actualmente no existe ninguna herramienta, ni ningún ejemplo liberado de forma pública que permita comprobar y generar configuraciones e ir evolucionando las mismas para mejorar tanto su seguridad como su diversidad por lo que se ha optado por la realización de una herramienta capaz de realizar esto.

Para el análisis y cuantificación de vulnerabilidades si existe herramientas, además algunas de ellas liberadas con licencias abiertas. Hemos optado por OWASP ZAP ya que es uno de las herramientas de análisis de vulnerabilidades más utilizadas y además está liberada bajo la licencia GPLv3 **free__software__foundation__gnu__2007**.

3.2. Protocolos y/o servidores de red

En esta sección analizaremos algunos de los protocolos de red más conocidos así como algunas de sus implementaciones. Ateniéndonos a la filosofía abierta de este proyecto nos limitaremos a las implementaciones libres de los

mismos, además, diversas publicaciones demuestran que el software de código abierto es mas seguro que el software cerrado **walia__comparative__2006 mansfield-devine__open__2008 clark__is__2009**.

3.2.1. SMB - Server Message Block

Es un protocolo de red desarrollado por IBM a principios de la década de los 90 y adoptado por Microsoft a partir de 1992. Este protocolo permite compartir archivos e impresoras en red.

Debido a la cantidad de sistemas compatibles es uno de los protocolos más utilizados para compartir ficheros en redes empresariales, esto hace que sea uno de los objetivos principales de muchos ciberataques.

Por poner un ejemplo, en mayo de 2017 hubo un ciberataque masivo causado por el ransomware WannaCry **sarabia__mayor__2017**, dicho software hacía uso de un *exploit* conocido como EternalBlue que conseguía penetrar en sistemas que todavía hacían uso de los protocolos SMB1 y SMB2 obligando a MicroSoft a publicar parches incluso para sistemas operativos que habían terminado su ciclo de vida (Windows XP y Windows Vista).

Implementaciones libres

Nombre	Samba
Licencia	GPLv3
Año de lanzamiento	1992
Lenguaje de programación	C++, Python y C
Sitio Web	`https://www.samba.org`

Cuadro 3.1: Ficha técnica Samba

Samba es una implementación libre del protocolo usado para compartir archivos de Microsoft desarrollado originalmente para Unix por Andrew Tridgell utilizando técnicas de ingeniería inversa para averiguar el funcionamiento del protocolo. Aunque sigue siendo un protocolo propietario a partir de la versión 2.0 (2006) Microsoft comenzó a publicar las especificaciones del protocolo SMB para permitir la interoperatibilidad entre diferentes sistemas operativos.

3.2.2. NTP - Network Time Protocol

Network Time Protocol (NTP) es un protocolo utilizado para sincronizar los relojes de diversos sistemas informáticos.

Implementaciones libres

Nombre	OpenNTPD
Licencia	ISC
Año de lanzamiento	2004
Lenguaje de programación	C
Sitio Web	`http://www.openntpd.org`

Cuadro 3.2: Ficha técnica OpenNTPD

OpenNTPD es una implementación del protocolo NTP para sincronizar el reloj del sistema contra servidores NTP remotos. También puede actuar como un servidor NTP para clientes compatibles con NTP. OpenNTPD está desarrollado principalmente por Henning Brauer como parte del proyecto OpenBSD.

La motivación para desarrollar OpenNTPD fue una combinación de problemas con las implementación NTP existentes como pueden ser una configuración difícil, código complejo y difícil de auditar así como licencias incompatibles con la licencia BSD.

3.2.3. VPN - Virtual Private Network

Una red privada virtual (VPN) es una tecnología que permite crear una red segura de acceso local (LAN) sobre una red pública como puede ser Internet.

El protocolo más utilizado es IPSEC, pero también existen protocolos como pueden ser PPTP y L2TP. Cada uno con sus ventajas y desventajas en cuanto a seguridad, facilidad, mantenimiento y tipos de clientes soportados.

Implementaciones libres

Nombre	strongSwan
Licencia	GPLv2
Año de lanzamiento	2005
Lenguaje de programación	C
Sitio Web	`http://www.strongswan.org`

Cuadro 3.3: Ficha técnica strongSwan

Nombre	Libreswan
Licencia	GPL
Año de lanzamiento	2013
Lenguaje de programación	C
Sitio Web	`http://libreswan.org`

Cuadro 3.4: Ficha técnica Libreswan

3.2.4. SSH - Secure Shell

El protocolo SSH (Secure SHel) sirve para abrir un intérprete de comandos en un servidor remoto.

Además de dicho intérprete de comandos, SSH nos permite copiar datos de forma segura (scp), gestionar claves de tipo publica/privada para no requerir el uso de contraseñas y pasar los datos de cualquier otra aplicación por un canal seguro a través de túneles.

Implementaciones libres

Nombre	Dropbear
Licencia	MIT
Año de lanzamiento	2003
Lenguaje de programación	C
Sitio Web	`http://matt.ucc.asn.au/dropbear/dropbear.html`

Cuadro 3.5: Ficha técnica Dropbear

Dropbear es un servidor SSH desarrollado por Matt Johnston. Está diseñado para entornos con pocos recursos como pueden ser sistemas embebidos por lo que es ampliamente utilizado en routers basados en Linux como puede ser OpenWRT.

Nombre	OpenSSH
Licencia	BSD
Año de lanzamiento	1999
Lenguaje de programación	C
Sitio Web	`http://www.openssh.com`

Cuadro 3.6: Ficha técnica Libreswan

OpenSSH es una implementación libre del protocolo SSH desarrollado por Theo de Raadt, fundador también del proyecto OpenBSD.

3.2.5. FTP - File Transfer Protocol

FTP (File Transfer Protocol) es uno de los primero protocolos diseñado para la transferencia de archivos entre sistemas utilizando una arquitectura cliente/servidor. A pesar de sus problemas de seguridad **todd_why_2000** es uno de los protocolos más utilizados para transferir archivos. Dichos problemas se deben a que el todo el intercambio de información, desde la autenticación del usuario hasta la transferencia de, archivo, se realiza en texto plano sin ningún tipo de cifrado. Hoy en día se aconseja el uso de otros métodos mas eficaces como los programas rsync o scp basados a su vez en SSH. El propio repositorio de código de `vsftpd` fue comprometido en 2011 **hkcert_security_bulletin_sa11070501_2011**.

Implementaciones libres

Nombre	vsftpd
Licencia	GPLv2
Año de lanzamiento	2001
Lenguaje de programación	C
Sitio Web	`https://security.appspot.com/vsftpd.html`

Cuadro 3.7: Ficha técnica vsftp

vsftpd es un servidor FTP con licencia GPL para sistemas UNIX.

3.2.6. HTTP - Hypertext Transfer Protocol

El protocolo de transferencia de hipertexto, HTTP por sus siglas en inglés, es un protocolo de red. Se utiliza para enviar y recibir páginas web en Internet. Fue desarrollado por Tim Berners-Lee y está coordinado por el **World Wide Web Consortium** (W3C).

Implementaciones libres

Nombre	Apache
Licencia	Apache 2.0
Año de lanzamiento	1995
Lenguaje de programación	C
Sitio Web	`https://httpd.apache.org`

Cuadro 3.8: Ficha técnica Apache

El servidor HTTP Apache es un servidor web HTTP de código abierto desarrollado y mantenido por la Fundación Apache, siendo uno de los servidores HTTP más utilizados del mudo.

Nombre	NGINX
Licencia	BSD
Año de lanzamiento	2004
Lenguaje de programación	C
Sitio Web	`http://nginx.org`

Cuadro 3.9: Ficha técnica NGINX

NGINX es un servidor web de alto rendimiento inicialmente desarrollado por Igor Sysoev siendo uno de los mas utilizados en la actualidad.

3.3. Algoritmos genéticos

Un algoritmo genético es un algoritmo de optimización que imita el proceso de selección natural descrito por Charles Darwin en su teoría de la evolución y nos puede ayudar a resolver problemas de optimización y búsqueda imitando procesos biológicos naturales, como pueden ser la mutación, la selección y el cruzamiento **batista_algoritmos_2009**.

Los algoritmos genéticos son heurísticas de búsqueda que se utilizan a menudo para encontrar soluciones complejas y no obvias a la optimización algorítmica y a los problemas de búsqueda **orcero_inteligencia_2002**.

Los algoritmos genéticos se basan en un conjunto de individuos de una población natural, codificando la información de cada solución en una cadena llamada cromosoma. Los símbolos que forman la cadena son llamados genes.

Los cromosomas evolucionan a través de iteraciones, llamadas generaciones. En cada generación, los cromosomas son evaluados usando alguna medida de aptitud. Las siguientes generaciones (nuevos cromosomas), son generadas aplicando los operadores genéticos de selección, mutación y cruzamiento repetidamente.

Capítulo 4

Metodología

En este capítulo vamos a definir y a detallar las herramientas necesarias y los procesos que vamos a seguir para solucionar el problema planteado en base a los objetivos que presentamos en el segundo capítulo y a los antecedentes vistos en el tercero.

4.1. Herramientas utilizadas

4.1.1. OWASP ZAP

Analizar la seguridad es una tarea compleja que requiere tener en cuenta multitud de factores por lo que realizar dicho análisis de forma manual resulta poco menos que imposible. Por ello vamos a utilizar `ZAP` que es un analizador de vulnerabilidades de código abierto desarrollado por la organización `OWASP`. Dicho analizador fue una de las herramientas que obtuvo el premio *Bossie 2015* al mejor software de red y seguridad de código abierto **staff_bossie_2015**.

Aunque ZAP tiene una interfaz gráfica (ver 4.1) para nuestro proyecto vamos a automatizar su uso mediante la API disponible en lenguaje Python.

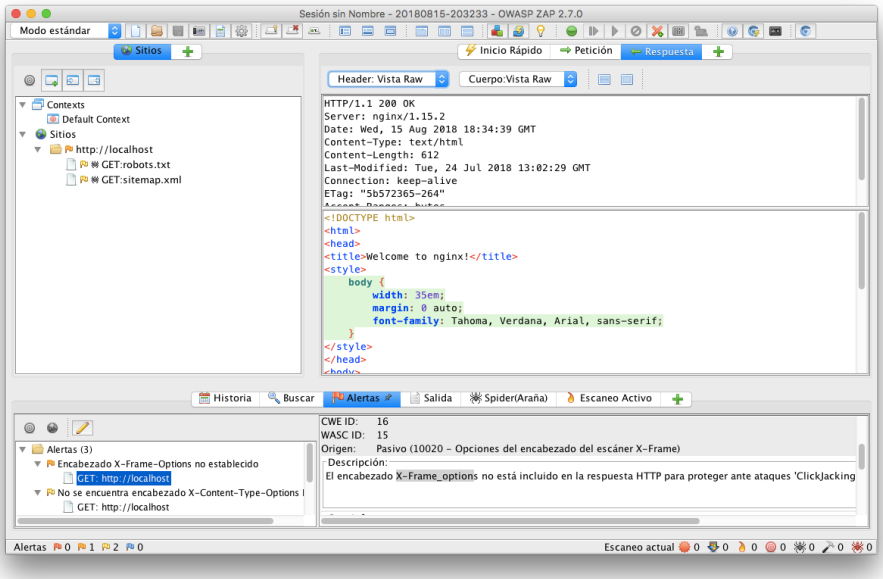

Figura 4.1: Ventana principal de OWASP ZAP

4.1.2. Docker

Para simular diferentes servidores de una forma rápida se ha optado por usar un sistema de virtualización ligera basada en contenedores. Para esto se ha usado **Docker** que es uno de los sistemas de virtualización ligera basada en contenedores más utilizada. Para la orquestación de los diferentes contenedores (Servidor y Analizador) se utilizará el estándar **docker-compose** que nos permite definir un conjunto de contenedores de una forma sencilla en un fichero de texto YML.

La orquestación estará compuesta de dos contenedores, uno basado en *Alpine Linux* en el que instalaremos el servidor que vayamos a analizar y un segundo contenedor basado en la imagen oficial de ZAP que podemos encontrar en la URL: `https://hub.docker.com/r/owasp/zap2docker-bare/`.

4.1.3. Python

Se ha decidido desarrollar este proyecto en el lenguaje de programación Python debido a estar familiarizado con este lenguaje y a la disponibilidad de una API de OWASP ZAP en dicho lenguaje. Además, aunque este proyecto no requiere un elevado rendimiento, diversas publicaciones indican un muy buen desempeño del lenguaje Python a la hora de trabajar con algoritmos genéticos **merelo-guervos_comparison_2016**.

Se han seguido buenas práctias de programación como pueden ser escribir pruebas unitarias de cada función desarollada. Para ejecutar dichas pruebas unitarias se ha utilizado la herramienta `pytest`. También se ha tenido en cuenta que la codificación siga una guía de estilo, concretamente `PEP8` que son es la guía de estilo considerada 'oficial' por parte de los desarrolladores de Python.

4.2. Análisis general del problema

Teniendo en cuenta la ingente cantidad de servicios de red con sus múltiples opciones de configuración se ha optado por limitar este proyecto a alterar y optimizar la configuración de un servidor HTTP, concretamente NGINX ya en los últimos años ha desbancado a Apache como el servidor HTTP más utilizado del mundo **w3techs__usage__2019**.

La elección del protocolo HTTP ha sido porque es el protocolo de capa de aplicación más extendido y utilizado en Internet, de hecho el estudio 'Global Internet Phenomena Report' de 2018 sitúa a HTTP muy por encima de otros servicios como la VoIP, la mensajería y los servicios de juegos en línea **sandvine__2018__2018**.

La configuración de NGINX, al igual que la de muchos otros servidores, se realiza mediante ficheros de texto plano con una sintaxis específica, la misma se puede encontrar definida en la siguiente URL: `http://nginx.org/en/docs/beginners_guide.html#conf_structure` y consta de múltiples directivas, podemos ver un ejemplo aquí: **??**.

Como podemos observar un fichero de configuración de NGINX tiene diferentes directivas que se pueden configurar de diferentes maneras, tanto para optimizar como para simplemente simular una configuración distinta y así hacer creer a un atacante que hemos cambiado de servidor HTTTP o incluso de servidor físico.

4.3. Banco de pruebas

Para empezar definimos un banco de pruebas para comprobar que la herramienta OWASP ZAP servía para nuestro propósito. Para ello se modificó el código de ejemplo que proporcionan para su API en Python para hacerla funcionar en un entorno de contenedores. El script (zap.py **??**) se conecta a una determinada URL y ejecuta una serie de comprobaciones para terminar devolviendo el número de vulnerabilidades encontradas. Dichas vulnerabilidades están basadas en la escala CVSS por lo que hay vulnerabilidades consideradas críticas y otras que son solo sugerencias.

A la hora de obtener una web para testear se estuvo barajando el uso de diferentes entornos web como pueden ser Galileo (Perl), WordPress (PHP), una simple web realizada en html7.1 e incluso una aplicación web premeditadamente vulnerable desarrollada por OWASP llamada `Juice-Shop` realizada con **Node.js** pero al final se optó por utilizar el mensaje de bienvenida de 'NGINX' (ver **??**). La configuración estándar de NGINX con dicho mensaje nos mostraba 5 alertas **??** siendo mucho mas fácil de gestionar que las 71 que obteníamos con "Juice-Shop".

Una parte muy importante para el correcto funcionamiento de OWASP ZAP fue simular un dominio, en este caso utilizamos el conocido **example.com** que es un nombre reservado por la IANA (Internet Assigned Numbers Authority) en el RFC2606 **eastlake_reserved_1999**. Para hacer uso del mismo se ha optado por darle ese nombre de **host** al contenedor que levanta NGINX.

Una vez hemos generado nuestro laboratorio para pruebas y viendo que se NGINX se ejecuta correctamente pasamos a probar como va cambiando la respuestas de OWASP ZAP dependiendo de la configuración de nuestro NGINX. Para ello agregamos la cabecera

```
add_header X-Frame-Options "SAMEORIGIN";
```

y ejecutamos el conjunto, dándonos 4 vulnerabilidades en lugar de las 5 anteriores. Esta prueba nos indica que efectivamente la modificación de los parámetros de NGINX puede darnos configuraciones más o menos seguras.

4.4. Eligiendo los parámetros

La última versión de NGINX (1.17.2) posee 739 directivas de configuración por lo que para simplificar nuestra tarea vamos a limitarnos a un conjunto mas pequeño de directivas (ver tabla 4.4).

En un primer momento pensamos en basarnos en las directivas **STIG**, pero estas directivas sólo están definidas para el servidor Apache. Por suerte al ser HTTP un estándar muchas de las directivas de Apache tienen su símil en NGINX por lo que hemos extraído un conjunto de las mas representativas y que además tuvieran su equivalente para el servidor NGINX. Las puntuaciones se basan en el sistema de puntuación CVSS de acuerdo con los resultados devueltos por ZAP con el código de prueba. También se han agregado algunas cabeceras identificativas como pueden ser la identificación del servidor (ver tabla 4.4) así como la directivas de compresión `gzip` para tener una mayor entropía.

Id	STIG ID	Configuración Apache	Equivalente NGINX
0	V-13730	MaxClients	worker_connections
1	V-13726	KeepAliveTimeout	keepalive_timeout
2	V-13732	FollowSymLinks	disable_symlinks
3	V-13735	Indexes	autoindex
4	V-13724	Timeout	send_timeout
5	V-13738	LimitRequestFieldsize	large_client_header_buffers
6	V-13736	LimitRequestBody	client_max_body_size
7	V-6724	ServerTokens	server_tokens
8			gzip

Cuadro 4.1: Lista de directivas NGINX utilizadas

Id	Cabecera
9	X-Frame-Options
10	X-Powered-By
11	X-Content-Type-Options
12	Server

Cuadro 4.2: Lista de cabeceras HTTP utilizadas

Pasamos a detallar que realiza cada directiva escogida y sus posibles valores:

4.4.1. worker_connections

Sintaxis	worker_connections number;
Por defecto	worker_connections 512;
Contexto	events

Establece el número máximo de conexiones simultáneas que pueden ser abiertas por un proceso de NGINX.

Debe tenerse en cuenta que este número incluye todas las conexiones (por ejemplo, conexiones con servidores proxy, entre otros), no sólo las conexiones con clientes. Otra consideración es que el número real de conexiones simultáneas no puede exceder el límite actual del número máximo de archivos abiertos.

4.4.2. keepalive_timeout

Sintaxis	keepalive_timeout timeout;
Por defecto	keepalive_timeout 75s;
Contexto	http, server, location

Establece un tiempo de espera durante el cual una conexión cliente permanecerá abierta en el lado del servidor. El valor cero deshabilita las conexiones de cliente Keep-alive. La cabecera "Keep-Alive: timeout=time" está soportada por Firefox y Chrome. Internet Explorer cierra las conexiones abiertas por sí mismo en unos 60 segundos.

4.4.3. disable_symlinks

Sintaxis	disable_symlinks on |off;
Por defecto	disable_symlinks off;
Contexto	http, server, location

Determina cómo se deben tratar los enlaces simbólicos al abrir archivos. Cuando está desactivado los enlaces simbólicos en la ruta de acceso están permitidos y no están marcados. Este es el comportamiento por defecto. Cuando está activado y algún componente de la ruta de acceso es un enlace simbólico, se deniega el acceso a ese archivo.

4.4.4. autoindex

Sintaxis	autoindex on |off;
Por defecto	autoindex off;
Contexto	http, server, location

Cuando está activado muestra el contenido de los directorios, en caso contrario no muestra nada.

4.4.5. send_timeout

Sintaxis	send_timeout time;
Por defecto	send_timeout 60s;
Contexto	http, server, location

Establece el tiempo de espera para transmitir una respuesta al cliente. El tiempo de espera se establece sólo entre dos operaciones de escritura

sucesivas, no para la transmisión de la respuesta completa. Si el cliente no recibe nada en este tiempo, la conexión se cierra.

4.4.6. large_client_header_buffers

Sintaxis	large_client_header_buffers number size;
Por defecto	large_client_header_buffers 4 8k;
Contexto	http, server

Establece el número máximo y el tamaño de los búferes utilizados para leer los encabezados de solicitudes de clientes grandes. Una línea de petición no puede exceder el tamaño de un búfer, o el error 414 (Request-URI Too Large) es devuelto al cliente. Un campo de encabezado de solicitud no puede exceder el tamaño de un búfer también, o el error 400 (Bad Request) es devuelto al cliente. Los búferes se asignan sólo bajo demanda. Por defecto, el tamaño del búfer es igual a 8K bytes. Si una vez finalizada la tramitación de la solicitud, la conexión pasa al estado de espera, se liberan estos búferes.

4.4.7. client_max_body_size

Sintaxis	client_max_body_size size;
Por defecto	client_max_body_size 1m;
Contexto	http, server, location

Establece el tamaño máximo permitido del cuerpo de la solicitud del cliente, especificado en el campo "Content-Length" del encabezado de la solicitud. Si el tamaño de una solicitud excede el valor configurado, el error 413 (Request Entity Too Large) se devuelve al cliente. Tenga en cuenta que los navegadores no pueden mostrar correctamente este error. Configurar el tamaño a 0 desactiva la comprobación del tamaño del cuerpo de la solicitud del cliente.

4.4.8. server_tokens

Sintaxis	server_tokens on |off;
Por defecto	server_tokens on;
Contexto	http, server, location

Habilita o deshabilita la emisión de la versión NGINX en las páginas de error y en el campo "Server" del encabezado de respuesta.

4.4.9. gzip

Sintaxis	gzip on |off;
Por defecto	gzip off;
Contexto	http, server, location

Habilita o deshabilita la compresión de las respuestas HTTP.

4.4.10. X-Frame-Options

Sintaxis	X-Frame-Options: DENY |SAMEORIGIN |ALLOW-FROM url;
Contexto	server, location

La cabecera "X-Frame-Options" puede ser usada para indicar si debería permitírsele a un navegador renderizar una página de forma embebida . Las páginas web pueden usarlo para evitar ataques de *clickjacking*, asegurándose que su contenido no es embebido en otros sitios.

4.4.11. X-Powered-By

Sintaxis	X-Powered-by: NGINX;
Contexto	server, location

La cabecera "X-Powered-By" se usa para especificar con que software se ha generado la respuesta por parte del servidor.

Se recomienda no dar información demasiado extensa en dicha cabecera ya que puede revelar detalles que pueden facilitar la tarea de encontrar y explotar fallos de seguridad.

4.4.12. X-Content-Type-Options

Sintaxis	X-Content-Type-Options: nosniff;
Contexto	server, location

El encabezado HTTP de respuesta "X-Content-Type-Options" es un marcador utilizado por el servidor para indicar que los tipos *MIME* anunciados en los encabezados "Content-Type" no se deben cambiar ni seguir. Esto permite desactivar el "MIME type sniffing".

Introducido por Microsoft en Internet Explorer 8 e implementado paulatinamente por el resto de navegadores, ayuda a los administradores puedan bloquear el rastreo de contenido, pudiendo transformar tipos MIME no ejecutables en tipos MIME ejecutables.

4.4.13. server

Sintaxis	Server: NGINX 1.12;
Contexto	server, location

La cabecera 'Server' contiene la información acerca del software usado por el servidor.

Al igual que con la cabecera 'X-Powered-By', se recomienda no dar información demasiado extensa en dicha cabecera ya que puede revelar detalles que pueden facilitar la tarea de encontrar y explotar fallos de seguridad.

4.5. Metodología de desarrollo

Utilizar una metodología como puede ser el desarrollo 'en cascada' no es viable en un proyecto de estas características, ya que se ha ido aprendiendo como funcionaban diversas herramientas a la vez que se estaba escribiendo el código, porque lo que han habido multitud de cambios.

Para el desarrollo de nuestro sistema se ha optado por una metodología ágil. Concretamente nos hemos basado en la metodología 'eXtreme Programming' o programación extrema. De hecho el carácter de investigación de este proyecto se amolda perfectamente a las directrices de esta metodología puesto que el software y los requisitos se han ido cambiando a medida que se iban obteniendo resultados con las distintas herramientas.

4.6. Diseño de la herramienta

Para diseñar la herramienta hemos usado, como hemos mencionado anteriormente, el lenguaje de programación Python. Se han hecho uso de las librerías de código abierto `nginx-config-builder`, `ZAP API` así como `mock` para simular el comportamiento de funciones internas para conseguir una ejecución determinística de las pruebas unitarias.

La herramientas se compone de un sencillo *script* 'genetic.py' que es el algoritmo genético en sí y que hace uso de algunas funciones auxiliares tambien definidas por nosotros para calcular el 'fitness' usando el valor numérico devuelto por ZAP así como para la generación de la configuración de NGINX

y la generación de los individuos, en la figura 4.2 se puede ver el diagrama de clases de nuestra herramienta con sus funciones definidas y en la figura 4.3 podemos ver un diagrama de paquetes en los que se puede apreciar como funciona el conjunto de la aplicación en conjunto con el servidor NGINX y la herramienta OWASP ZAP.

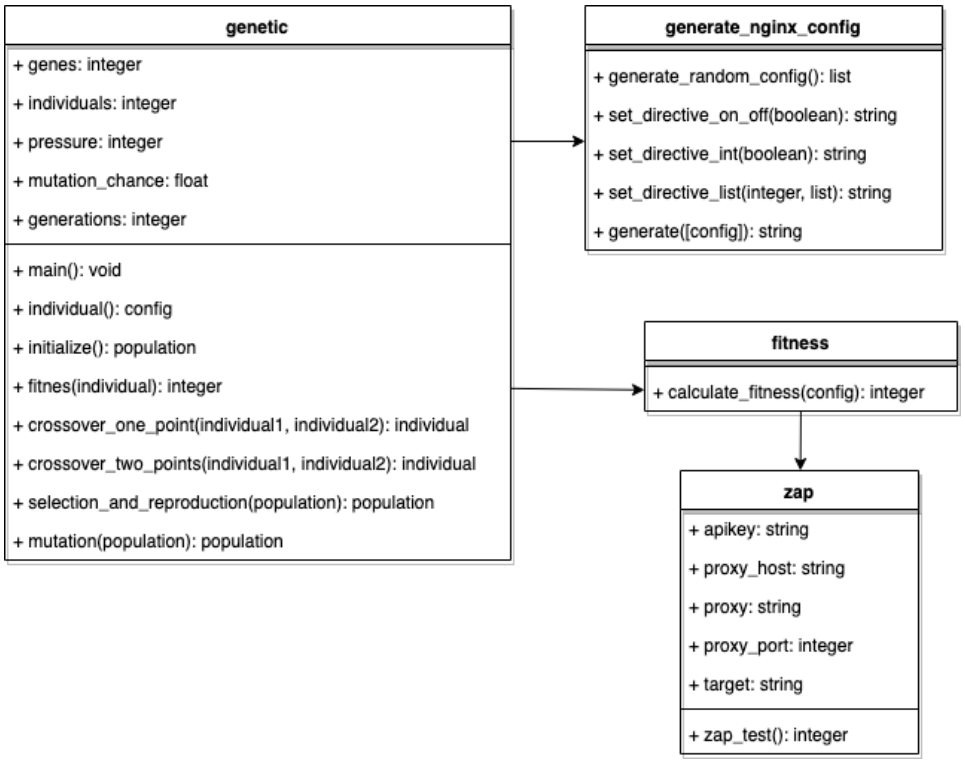

Figura 4.2: Diagrama de clases

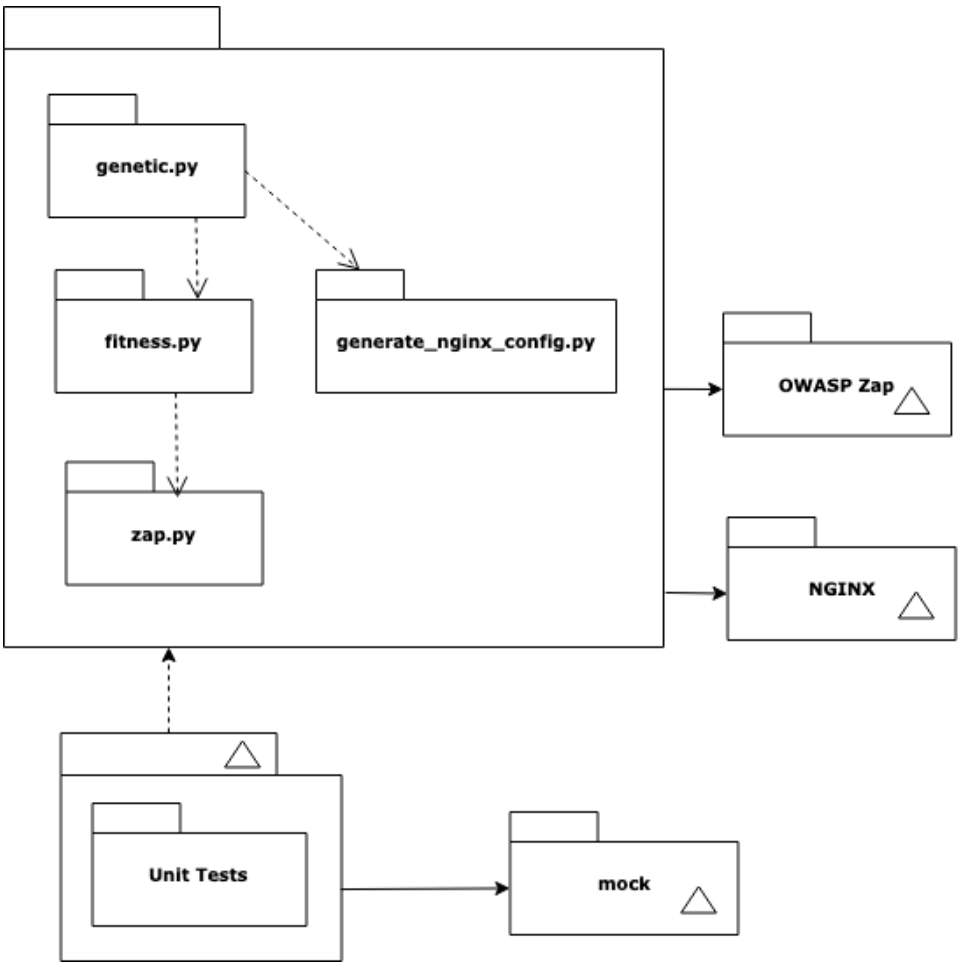

Figura 4.3: Diagrama de paquetes

En los siguientes puntos se detallará el desarrollo de cada una de las partes de la aplicación:

4.6.1. Generación de configuraciones

Para generar las distintas configuraciones de NGINX utilizamos la librería **nginx-config-builder** desarrollada por la empresa Linkedin y licenciada bajo la BSD. Dicha librería la podemos encontrar en `https://github.com/linkedin/nginx-config-builder`. Haciendo uso de esta librería desarrollamos un script **??** que en base a un cromosoma generaba una configuración, sin saber todavía si dicha configuración podía funcionar o no.

El siguiente paso es conseguir saber primeramente si la configuración

funciona y después saber si la misma configuración funcional es segura. Para lo primero el propio NGINX tiene una herramienta de comprobación de configuración, la podemos invocar con el comando **NGINX -t <archivo-configuracion.conf>**, dicha herramienta nos comprueba que la sintaxis del fichero de configuración sea correcto y NGINX pueda ejecutarse sin problemas. Para lo segundo configuraremos un servidor NGINX con dicha configuración y haremos uso de la herramienta ZAP, para poder hacerlo de forma dinámica y sencilla utilizaremos la potencia de Docker.

Nuestra función de evaluación utilizará una combinación de estas dos herramientas, ya que el primero nos servirá para descartar directamente las configuraciones erróneas y la segunda nos permitirá establecer la calidad de una configuración con un valor escalar.

4.6.2. Implementación del algoritmo genético

La implementación del algoritmo genético utiliza una combinación de procesos de selección, cruzamiento y mutación para mejorar la solución a través de varias generaciones.

El algoritmo genético responderá al segundo objetivo 'comprobar si utilizar una heurística de búsqueda, como pueden ser los algoritmos genéticos, puede servir para incrementar la seguridad de un sistema a la hora de generar configuraciones'

Aunque en un primero momento se planteó el uso de algún 'framework' de algoritmos genéticos como puede ser **DEAP**, desarrollado en Python, pero se optó en su lugar por utilizar un algoritmo sencillo que cubría de sobra nuestras necesidades.

Para la realización del algoritmo genético hemos utilizado como base uno de los múltiples ejemplos que hay en Internet. Concretamente el del estupendo tutorial que podemos encontrar en la web **robologs.net** concretamente en esta URL: `https://robologs.net/2015/09/01/`.

El sistema está diseñado para lograr el objetivo del planteamiento del problema. En este diseño, se han desarrollado dos algoritmos de cruzamiento (*crossover*), uno donde cruza a dos individuos mediante un sólo punto de cruce y otro donde se utilizan dos puntos de cruce.

El algoritmo genético se encarga de llamar a los anteriores `scripts` para generar soluciones de seguridad. En primer lugar, inicializa las soluciones aleatorias, que luego pasan por los procesos de selección, que se describen en detalle más adelante en este capítulo. El algoritmo genético depende de la puntuación de aptitud de cada solución para ir más allá en el proceso de

selección, lo que significa que es responsabilidad del algoritmo de puntuación de aptitud dar puntuaciones para las soluciones y enviarlas más allá del algoritmo genético.

El algoritmo de puntuación (fitness.py **??**) será el responsable de proporcionar al algoritmo genético las puntuaciones de fitness de las soluciones de seguridad. El sistema de puntuación se basa en las preferencias previas de STIG, que proporciona las vulnerabilidades, que será devuelto como un valor escalar dado por ZAP. Este valor escalar será un entero que puede ir de 0 a 999, siendo 0 una configuración sin ninguna vulnerabilidad y 999 una configuración incorrecta.

Básicamente el algoritmo crea una población de 'n' individuos aleatorios definidos en la variable 'individuals', y durante 'i' generaciones, definidas en la variable 'generations', los va seleccionado, cruzando y mutando. Para seleccionarlos lo que hace es otorgar una puntuación escalar con el algoritmo de fitness y ordenar la población de forma inversa, dejando al final de la lista los valores con menor fitness, pues son los que menor número de vulnerabilidades ha dado ZAP. Entonces se sustituyen los primeros elementos de la lista por el cruce de dos individuos aleatorios situados entre el final y el indicado por el valor 'pressure' que siempre deberá ser al menos mayor que 2 para darle mayor diversidad. Entonces se mutan los elementos con una probabilidad definida en la variable 'mutation_chance' y se vuelve a lanzar el algoritmo hasta que no queden generaciones. Al finalizar el algoritmo seleccionaremos el último elemento de la lista de la población evolucionada como el mejor de todos.

Capítulo 5

Resultados

En este capítulo pasamos a detallar los resultados de todos y cada uno de los test que realizamos en base a la metodología que definimos en el capítulo anterior.

5.1. Análisis inicial del sistema

Con nuestro generador de código podemos generar configuraciones aleatorias de NGINX como la que podemos ver en el listado ver **??**. Aunque como ya hemos comentado previamente esa configuración puede ser errónea o menos segura que otra configuración también generada aleatoriamente. En este caso concreto la configuración es claramente errónea porque ha seleccionado elementos que harían que NGINX no pudiera ni siquiera inicializarse.

Los cromosomas representan una combinación de configuraciones que tiene un tamaño construido de acuerdo a los parámetros de las configuraciones donde un alelo o más representan un parámetro. En esta investigación, el tamaño de los cromosomas es de 13 y el total de individuos en una generación es de 20, lo que es suficiente para mantener la diversidad de los parámetros de acuerdo con el número de parámetros. El número de soluciones en una generación ha sido determinado después de muchos experimentos del algoritmo. Es posible aumentar el número de cromosomas en una generación, lo que podría aumentar la diversidad, pero eso costaría más tiempo para las pruebas.

Para validar que nuestros datos son correctos se han introducido algunos errores en la configuración de forma intencional. Las cabeceras 'X-Frame-Options' y 'X-Powered-By' contienen unos valores incorrectos al final, por lo que el gen 10 nunca podrá tener un valor de 3, asimismo el gen 11 nunca podrá tener un valor de 5. Con esto nos aseguramos que efectivamente las configuraciones incorrectas nunca serán dadas por válidas.

A continuación pasamos a ejecutar nuestras pruebas utilizando diferentes valores y utilizando dos funciones de cruce distintas. Una funciona de cruzamiento lo hará en un único punto aleatorio y el otro lo hará en dos.

Se han generado los individuos de forma aleatoria y se ha ejecutado el algoritmo genético durante las generaciones indicadas, podemos ver que hay configuraciones incorrectas (en rojo).

Tras cada ejecución obtenemos una configuración de NGINX que podríamos aplicar en nuestro servidor.

Para facilitar la interpretación de los resultados en la siguiente tabla se adjunta una tabla con la correspondencia entre el identificador y su directiva NGINX asociada.

Directiva NGINX	**Identificador**
worker_connections	1
keepalive_timeout	2
disable_symlinks	3
autoindex	4
send_timeout	5
large_client_header_buffers	6
client_max_body_size	7
server_tokens	8
gzip	9
X-Frame-Options	10
X-Powered-By	11
X-Content-Type-Options	12
server	13

5.2. Cruzamiento en un único punto

5.2.1. Población de 10 individuos durante 2 generaciones

Resultados para una población de 10 individuos durante 2 generaciones:
Población inicial:

1	2	3	4	5	6	7	8	9	10	11	12	13
1884	75	1	1	0	805	1491	1	1	0	4	1	1
666	17	0	1	1	659	1251	0	1	0	5	1	1
1515	19	1	1	0	1104	1720	1	0	3	2	1	0
955	52	1	1	0	1867	525	1	1	0	5	0	2
1916	110	1	0	1	846	619	0	1	2	2	0	0
604	106	1	0	1	2004	878	1	1	2	4	1	0
925	65	1	1	0	884	2036	1	1	3	1	1	2
889	13	1	1	1	1599	1895	0	1	1	2	0	2
1324	92	1	1	1	2009	692	0	1	2	1	0	1
764	14	1	1	1	1076	1936	0	0	1	4	1	2

Población final:

1	2	3	4	5	6	7	8	9	10	11	12	13
1324	92	0	1	1	659	1251	0	1	0	5	1	1
604	106	1	0	1	2004	878	1	1	0	1	0	1
666	17	0	0	1	846	619	0	1	2	2	0	0
666	17	0	0	1	2009	692	0	1	0	2	0	0
666	17	0	0	1	846	619	0	1	2	1	0	1
666	17	0	1	1	2009	692	0	1	0	1	0	1
666	17	0	1	1	659	1251	0	1	0	5	1	1
666	17	0	0	1	846	619	0	1	2	2	0	0
604	106	1	0	1	2004	878	1	1	2	4	1	0
1324	92	1	1	1	2009	692	0	1	2	1	0	1

5.2.2. Población de 10 individuos durante 5 generaciones

Resultados para una población de 10 individuos durante 5 generaciones:
Población inicial:

1	2	3	4	5	6	7	8	9	10	11	12	13
808	90	1	0	1	1675	1228	0	0	0	3	1	0
1130	102	0	1	1	1347	1232	1	0	3	2	1	1
886	81	0	0	1	1860	1851	0	1	3	5	1	1
1845	92	1	1	1	1978	1661	1	0	3	0	0	1
1394	44	1	1	1	1986	975	0	1	0	3	0	1
1444	73	1	1	1	1364	1532	0	1	0	5	1	1
960	22	1	0	0	780	1749	1	0	3	3	0	0
1663	12	1	1	0	538	1004	0	1	3	2	0	0
877	18	1	0	1	972	1588	1	0	1	1	0	2
852	23	1	1	0	1881	1253	1	0	0	2	1	1

Población final:

1	2	3	4	5	6	7	8	9	10	11	12	13
808	90	1	1	0	1675	1228	0	0	0	3	1	1
808	90	1	1	0	1675	1228	0	0	0	3	1	0
808	23	1	0	1	1675	1228	0	0	0	3	0	1
808	23	1	0	0	1675	1228	0	0	0	3	0	1
808	90	1	0	0	1675	1228	0	0	0	3	1	0
808	23	1	0	0	1675	1228	0	0	0	3	1	0
852	23	1	0	1	1675	1228	0	0	0	3	1	0
852	23	1	0	1	1675	1228	0	0	0	3	0	1
808	90	1	1	0	1675	1228	0	0	0	3	1	1
808	90	1	1	0	1675	1228	0	0	0	3	1	1

5.2.3. Población de 20 individuos durante 20 generaciones

Resultados para una población de 20 individuos durante 20 generaciones:
Población inicial:

1	2	3	4	5	6	7	8	9	10	11	12	13
1243	104	0	0	1	1308	1083	1	1	0	1	0	1
821	90	0	1	0	1669	1685	1	0	3	4	0	1
729	31	0	1	1	1548	1278	1	0	1	1	0	2
1265	11	1	1	0	1906	1049	1	1	2	4	0	2
831	10	1	1	0	1095	917	0	1	0	2	1	0
1174	11	1	1	1	1049	1728	0	0	1	1	1	0
832	56	0	0	0	1942	1548	1	0	3	2	1	1
935	100	0	0	0	1562	1395	0	0	1	0	1	1
1886	64	1	1	1	1533	1351	0	0	0	0	0	2
724	70	0	0	1	831	1710	0	1	2	3	1	1
519	44	1	0	1	1340	1555	1	0	2	0	1	1
923	85	0	1	0	1910	1432	0	0	1	2	0	0
667	60	1	0	1	1692	1686	1	1	0	0	1	0
1959	15	0	1	1	1733	1242	1	1	3	0	0	2
765	32	0	0	0	1442	1625	1	1	1	1	0	2
1482	117	1	0	1	595	1477	1	0	0	5	1	1
1091	35	0	1	0	1389	2043	1	0	2	3	1	2
524	79	0	1	0	1019	1730	0	0	3	5	0	1
1499	10	0	0	1	1042	1140	1	1	3	1	1	2
966	12	1	0	0	1651	632	0	1	2	4	0	0

Población final:

1	2	3	4	5	6	7	8	9	10	11	12	13
524	18	0	0	0	1340	1686	0	1	0	0	1	0
865	18	0	0	0	1340	1686	1	1	0	0	1	0
524	39	0	0	0	1340	552	1	1	0	0	1	0
524	39	0	0	0	1340	1686	0	1	0	2	1	0
524	39	0	0	0	1340	1686	0	1	0	0	0	0
524	40	0	0	0	1340	1686	1	1	0	0	1	0
524	39	0	0	0	1340	1686	0	1	3	0	1	0
524	34	0	0	0	1340	1686	0	1	0	0	1	0
524	18	0	0	0	1340	1686	0	1	0	0	1	0
806	34	0	0	0	1340	1686	0	1	0	0	1	0
524	39	0	0	0	1340	1686	1	1	0	0	1	0
524	18	0	0	0	1340	1952	0	1	0	0	1	0
524	34	0	0	0	1340	1686	1	1	0	0	1	0
767	39	0	0	0	1340	1686	1	1	0	0	1	0
524	18	0	0	0	1340	1686	1	1	0	0	1	0
524	39	0	0	0	1340	1686	1	1	0	0	1	0
524	39	0	0	0	1340	1686	0	1	0	0	1	0
524	39	0	0	0	1340	552	1	1	0	0	1	0
524	34	0	0	0	1340	1686	1	1	0	0	1	0
524	18	0	0	0	1340	1686	0	1	0	0	1	0

5.3. Cruzamiento en dos puntos

5.3.1. Población de 10 individuos durante 2 generaciones

Resultados para una población de 10 individuos durante 2 generaciones:
Población inicial:

1	2	3	4	5	6	7	8	9	10	11	12	13
1997	107	0	1	0	1571	608	1	1	0	1	0	0
1879	23	1	1	0	1378	1842	0	1	0	2	0	0
1036	79	1	1	1	1687	763	1	0	3	4	1	2
1991	20	1	0	0	747	1662	1	1	1	0	0	2
1793	43	0	0	0	1080	1822	1	0	0	0	0	0
1471	81	0	1	0	1547	1390	1	0	2	5	0	2
1923	36	1	0	1	1269	717	0	0	1	4	0	1
961	47	1	0	1	634	1765	0	0	3	4	0	0
899	30	0	0	1	992	1202	0	1	1	2	0	2
1423	78	0	1	0	1615	2027	0	0	0	5	1	0

Población final:

1	2	3	4	5	6	7	8	9	10	11	12	13
899	34	0	0	1	634	1765	0	0	0	1	0	1
961	47	1	0	1	634	1765	0	0	3	4	0	0
961	47	1	0	1	634	1765	0	0	1	2	0	0
899	47	1	0	1	634	1765	0	1	0	1	0	1
961	47	1	0	1	634	1765	0	0	2	4	0	0
961	47	1	0	1	634	1765	0	0	3	4	0	0
961	47	1	0	1	634	1765	0	0	3	4	0	0
961	47	1	0	1	634	1765	0	0	3	4	0	0
899	30	0	0	1	992	1202	0	1	1	2	0	2
899	30	0	0	1	992	1202	0	1	0	1	0	1

5.3.2. Población de 10 individuos durante 5 generaciones

Resultados para una población de 10 individuos durante 5 generaciones:
Población inicial:

1	2	3	4	5	6	7	8	9	10	11	12	13
2008	57	1	1	0	1008	1736	1	0	2	3	0	0
2036	115	1	1	0	715	1646	0	0	0	3	0	2
806	10	0	0	1	763	1061	0	1	0	0	1	2
1184	19	0	0	0	687	1944	0	0	0	3	1	2
1244	100	0	0	1	1930	1088	0	0	0	1	0	2
1498	46	0	1	0	819	1391	0	0	1	1	0	2
2037	120	1	1	1	1137	1059	1	0	2	2	0	1
1171	118	0	0	1	1564	569	0	1	1	4	1	2
714	120	1	0	0	1282	1579	0	1	2	1	1	2
1710	87	0	0	0	654	529	0	0	2	1	1	2

Población final:

1	2	3	4	5	6	7	8	9	10	11	12	13
714	120	0	0	0	687	1944	1	1	2	1	1	2
714	43	0	0	0	1282	1944	0	0	2	1	1	0
714	43	0	0	0	1282	1944	1	0	2	1	1	2
714	120	0	0	0	1282	1579	0	1	1	1	1	0
714	120	0	0	0	687	1579	1	1	2	1	1	2
714	120	0	0	0	1282	1579	0	1	2	1	1	0
714	120	0	0	0	687	1944	1	1	2	1	1	2
714	43	0	0	0	1282	1579	0	0	2	1	1	2
714	120	1	0	1	687	1944	1	0	2	1	1	2
714	120	0	0	0	1282	1579	0	1	2	1	1	2

5.3.3. Población de 20 individuos durante 20 generaciones

Resultados para una población de 20 individuos durante 20 generaciones:
Población inicial:

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
1677	14	0	1	0	804	1001	1	0	0	0	0	0
1591	95	1	1	0	550	1731	0	0	1	0	0	2
776	110	1	0	1	1205	1182	1	0	3	3	1	0
1200	71	1	1	1	1903	1979	1	1	1	2	0	1
1961	28	0	0	1	1211	1485	1	1	0	4	0	2
942	10	0	1	1	1446	1525	1	0	0	1	0	1
537	48	1	0	1	1358	1930	1	0	3	1	1	1
1105	10	1	0	0	571	1297	0	1	0	3	1	2
1331	84	1	0	0	748	1126	0	0	2	5	1	2
2017	112	1	1	0	910	737	1	1	3	2	1	1
770	96	0	0	1	1257	1804	1	1	1	3	0	1
1569	104	1	1	1	713	671	0	0	0	3	0	1
1280	120	1	1	1	1461	1404	1	1	0	1	0	1
898	92	1	0	0	1943	1409	1	1	0	1	1	1
1632	70	1	0	0	1821	1836	1	0	1	5	0	1
982	36	0	1	1	1462	1446	1	0	1	1	1	2
1843	118	0	0	1	697	828	1	0	1	4	0	0
782	109	1	0	1	524	1928	1	1	1	4	1	2
1779	10	1	0	0	1627	1315	1	1	1	1	0	2
1927	21	1	0	0	771	576	1	1	2	2	1	0

Población final:

1	2	3	4	5	6	7	8	9	10	11	12	13
537	10	0	0	0	571	994	1	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	849	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	1	3	0	0
537	10	0	0	0	571	849	0	0	0	3	0	0
537	10	0	0	0	571	849	0	0	0	3	0	0
537	10	0	0	0	571	1285	0	0	0	3	0	0
537	10	0	0	0	571	849	0	0	0	3	0	0
537	10	0	0	0	1430	849	0	0	0	3	0	0
537	10	0	0	1	571	994	0	0	0	3	0	0
537	10	0	0	0	1028	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	994	0	0	0	3	0	0
537	10	0	0	0	571	849	0	0	0	3	0	0

5.4. Resultados para 30 ejecuciones del algoritmo

Se han realizado 30 ejecuciones del algoritmo utilizando el script `run.sh`, como los resultados en bruto son demasiado extensos se adjuntan junto al código fuente en la carpeta resultados. De estos datos lo que hemos extraído una muestra de la moda para ver cuales son los parámetros que el algoritmo considera que son más seguros.

1	2	3	4	5	6	7	8	9	10	11	12	13
519	33	0	0	0	1632	1557	0	0	0	1	0	2

5.5. Comentarios sobre los resultados

Como podemos observar, en las generaciones iniciales hay configuraciones correctas e incorrectas, las cuales podemos identificar porque tienen algún valor en rojo.

Con tan solo 2 generaciones y con una población de 10 individuos generados aleatoriamente la población final tiene configuraciones incorrectas pero aun así el algoritmo al ordenar la población en base al fitness nos posicionará

al final de la lista la que considera la configuración más segura de toda la población. Aun así se aprecia que con tan pocas generaciones el algoritmo no es capaz de evolucionar para descartar las configuraciones incorrectas, esta casuística se da por igual utilizando el cruzamiento por un punto y el de dos puntos.

Si ya pasamos a 5 generaciones manteniendo la población en 10 individuos generados aleatoriamente en ambos casos ya empieza a eliminar las configuraciones incorrectas, en ambos casos por igual aunque se aprecia que el cruce por dos puntos tiene la ventaja de proporcionar una mayor diversidad en la población.

Como prueba final hemos ejecutado el algoritmo durante 20 generaciones con una población de 20 individuos generados aleatoriamente vemos que, tanto con la función de cruce en un punto como con la que lo hace en dos puntos, el algoritmo descarta las configuraciones incorrectas pero en este caso, al ser tantas generaciones perdemos diversidad aunque ganamos en seguridad según el valor indicado por ZAP.

Tanto en 5 como en 20 generaciones el algoritmo genético ha evolucionado la población hasta llegar a un estado cercano al ideal en cuanto a seguridad.

Podemos afirmar que la funciona de cruzamiento en dos puntos proporciona un mejor resultado en menos generaciones, pero al aumentar el número de las mismas esta ventaja ya no se aprecia de forma tan evidente.

Por lo tanto, una vez decidido que la función de cruzamiento en dos puntos proporciona mejores resultados que la que lo hace ne un único punto sólo deberíamos variar el número de generaciones dependiendo del grado de seguridad y/o diversidad que quisiéramos obtener, siendo a mas generaciones un resultado mas seguro pero menos diverso y siendo a menos generaciones un resultado mas diverso pero menos seguro.

Una vez hemos deducido que 5 generaciones con una función de cruzamiento en dos puntos es la que mejor resultado nos daba hemos realizado 30 ejecuciones del algoritmo. De dichas ejecuciones hemos podido comprobar que hay directivas que, dentro de un rango, no afectan a la seguridad pero si nos proporcionan mas diversidad, en cambio hay otras directivas en las el valor siempre va a ser fijo para no generar una configuración posiblemente insegura.

Capítulo 6

Conclusiones

El objetivo de este proyecto era mejorar seguridad de un sistema mediante el uso de algoritmos genéticos modificando los parámetros de configuración de un servidor (en este caso NGINX). El algoritmo genético se aplicó con éxito, lo que permitió que las configuraciones evolucionaran de forma diversa y segura a lo largo del tiempo.

Algunas vulnerabilidades pueden ser causadas por una mala configuración o por una desafortunada combinación de configuraciones que es difícil que un administrador descubra manualmente debido a la gran cantidad de parámetros y a la gran cantidad de combinaciones posibles.

Por lo tanto, gracias a un algoritmo genético se consiguió encontrar configuraciones más seguras. Las configuraciones fueron representadas como cromosomas y el algoritmo tomó esos cromosomas a través de una serie de procesos de selección, cruzamiento y mutación que resultaron en configuraciones todavía más seguras que la generación anterior.

Gracias a esto podemos transformar nuestro servidor en un objetivo móvil cambiando la configuración de forma periódica.

Los resultados demuestran el rendimiento del enfoque evolutivo para la gestión de configuraciones que consiste en 13 parámetros del servidor NGINX. El ataque simulado de estas configuraciones se basó en la herramienta OWASP ZAP. El algoritmo genético descubrió mejores ajustes de parámetros para los parámetros atacados en cada generación.

En las primeras etapas, con solo dos generaciones, la solución empezó a dar buenos resultados aunque no se podía asegurar que dichas configuraciones fueran óptimas y/o seguras aunque si diversas ya que se habían inicializado de forma aleatoria. En las últimas etapas, la mejora en seguridad fue mucho mayor que en las primeras aunque fue reduciéndose la diversidad.

El experimento demostró que la diversidad dentro de la generación mantenía la capacidad de tener una configuración diversa de generación en generación. Cambiar la configuración de generación en generación creó un objetivo móvil que induce a error a un atacante que intenta reconocer determinados patrones en la configuración del servidor.

Quizá emplear un algoritmo genético para generar estas configuraciones usando un número tan pequeño de cromosomas no sea la opción más óptima, ya que este proyecto está más enfocado en la diversidad que en la seguridad, pero uno de los objetivos era comprobar si se podían utilizar los algoritmos genéticos para generar y evolucionar las configuraciones y efectivamente esto ha sido posible.

6.1. Trabajos futuros

Este proyecto se podría evolucionar añadiendo más parámetros así como variando el número de generaciones y el de la población. Debido al poco tiempo para realizar este proyecto y con el propósito de probar el concepto, este experimento incluyó tan solo 13 parámetros de los más de 700. También podría ser una mejora investigar la seguridad agregando aplicaciones web mas avanzadas o incluso con diferentes servidores HTTP como pueden ser `Apache` o `Caddy`.

Otro posible trabajo futuro sería mejorar la generación aleatoria evitando tener individuos erróneos en la población inicial, generando de esta forma una población más segura, y además utilizar algún programa de "benchmark" como puede ser "Apache Benchmark" en nuestro algoritmo de fitness para saber que ademas de segura, nuestra configuración es óptima.

Capítulo 7

Anexos

7.1. Codigo Fuente

7.1.1. Hello World

7.2. Ficheros de configuración

Fragmento de código 7.1: Web sencilla realizada en HTML puro

```
<!DOCTYPE html>
<html lang="es-ES">
  <head>
    <meta charset="utf-8">
    <title>Ejemplo de 2 párrafos</title>
  </head>
  <body>
    <p>Esto es un párrafo.</p>
    <p>Esto es otro párrafo.</p>
  </body>
</html>
```

Glosario de términos

Alpine Linux: Es una distribución Linux basada en BusyBox y la librería `musl`. Debido a su pequeño tamaño, es muy utilizado en plataformas de contenedores ya que proporciona tiempos de arranque muy rápidos.

API (application programming interface): Son un conjunto de funciones que funcionan como una capa de abstracción para poder interactuar con un determinado software.

Apache Benchmark: Herramienta para realizar pruebas de carga de servidores web desarrollada por la Fundación Apache.

Auditoría de seguridad: Estudio que comprende el análisis de sistemas para identificar, enumerar y posteriormente describir las diversas vulnerabilidades que pudieran presentarse en una revisión exhaustiva de las estaciones de trabajo, redes de comunicaciones o servidores.

Backend: Es el motor de una aplicación, se encarga de realizar las funciones en segundo plano que se encargan de que la aplicación funcione.

Balanceo de carga: Técnica de configuración de servidores que permite que la carga de trabajo total se reparte entre varios de ellos para que no disminuya el rendimiento general de la infraestructura.

Caddy: Es un servidor web de código abierto escrito en Go. Utiliza la funcionalidad HTTP de la biblioteca estándar de Go.

Clickjacking: es una técnica maliciosa que consiste en engañar al usuario para que haga clic en algo diferente de lo que el usuario cree, revelando así información confidencial o permitiendo que otros tomen el control de su ordenador.

CVSS: El 'Common Vulnerability Scoring System' es un formato abierto de métricas para la comunicación de vulnerabilidades.

DEAP: Es un *framework* de código abierto abierto para la realización de algoritmos genéticos. Está desarrollado en Python.

Exploit: Es un código que se utiliza con fin de aprovechar una vulnerabilidad en un determinado software consiguiendo un comportamiento no deseado en el mismo.

Expresión regular: Secuencia de caracteres que forma un patrón de búsqueda, principalmente utilizada para la búsqueda de patrones de cadenas de caracteres u operaciones de sustitución.

Frontend: Es la interfaz de la aplicación, es la parte de la aplicación que el usuario utiliza para comunicarse con la misma.

Fundación Mozilla: Organización sin ánimo de lucro que desarrolla el navegador Firefox así como otros programas licenciados software libre.

Hash: Algoritmo que a partir de una entrada genera una salida alfanumérica que representa un resumen de dicha entrada. Un simple cambio en los datos de la entrada generará un resumen totalmente diferente.

HTML (HyperText Markup Language): Lenguaje de marcado que se utiliza para la realización de páginas web.

Iptables: Es un poderoso *firewall* integrado en el núcleo de `Linux` y que forma parte del proyecto `netfilter`.

JavaScript: Lenguaje de programación orientado a objetos interpretado que se utiliza principalmente para cargar programas desde el lado del cliente en los navegadores web.

JSON (JavaScript Object Notation): Formato de texto plano usado para el intercambio de información, independientemente del lenguaje de programación. Es una simplificación del formato XML siendo mucho mas legible por humanos.

LaTeX: Sistema de composición de documentos que permite crear textos en diferentes formatos (artículos, cartas, libros, informes...) obteniendo una alta calidad en los documentos generados.

MIME: Las 'Multipurpose Internet Mail Extensions', por sus siglas en inglés, son una especificación dirigida al intercambio de ficheros de diferentes formatos a través de Internet. Se usa en mayor medida para reconocer el tipo de dichos ficheros.

Musl: es una implementación de la librería estándar de C de pequeño tamaño destinada a sistemas operativos embebidos basados en el kernel de Linux, está publicada bajo la licencia MIT.

OWASP ZAP: 'El Zed Attack Proxy' es una herramienta para realizar análisis de seguridad en aplicaciones web desarrollada por OWASP. Es uno de los proyectos de OWASP más activos.

OWASP: El 'Open Web Application Security Project', por sus siglas en inglés, es una organización cuya misión es mejorar la seguridad del software. Ha desarrollado diversas herramientas así como estándares de comunicación de vulnerabilidades. Además cuenta con un popular *ranking* anual de las 10 vulnerabilidades que consideran críticas en aplicaciones web de mayor y ofrece recomendaciones sobre como evitarlas.

Prueba unitaria: Es una técnica que sirve para comprobar de forma atómica el funcionamiento del codigo de un sistema software. Su uso está aconsejado, *TDD* se basa en este tipo de pruebas.

Ransomware: es un tipo de *malware* que restringe el acceso a los datos almacenados en un sistema informático, comúnmente mediante técnicas de cifrado, y exige que se pague un rescate al creador para restaurar dicho acceso.

RSA: Es un sistema de criptografía basado en clave pública y privada. Es usado para securizar la transferencia de datos.

Sniffer: También conocido como analizador de paquetes, es un programa informático que puede interceptar y registrar el tráfico que pasa por una red.

Software libre: Software cuya licencia permite que este sea usado, copiado, modificado y distribuido libremente según el tipo de licencia que adopte.

SSH (Secure SHell): Protocolo que permite conectarse a máquinas remotas mediante conexiones seguras de red.

SSL (Secure Sockets Layer): Serie de protocolos criptográficos que proporcionan comunicaciones seguras por una red.

StatusCake: Es un servicio web que permite monitorizar la disponibilidad de servicios online.

STIG: Las 'Security Technical Implementation Guides' son unas directrices de ciberseguridad para estandarizar los protocolos de seguridad dentro de redes de ordenadores para mejorar la seguridad general. Estas guías, cuando se implementan, mejoran la seguridad del software, hardware, y arquitecturas físicas para reducir la exposición a vulnerabilidades.

TDD: El 'Test Driven Development', por sus siglas en inglés, es una técnica de desarrollo en las que primero se escriben pruebas unitarias del código y luego se va desarrollando el código de forma incremental hasta que pasa las diferentes pruebas unitarias.

URL (Uniform Resource Locator): nombre y con un formato estándar que permite acceder a un recurso de forma inequívoca.

WireShark: Es un analizador de red utilizado para auditar redes de comunicaciones.

YML: 'YAML Ain't Markup Language' por sus siglas en inglés es un formato de archivo para almacenar datos serializados de forma legible para humanos. Es compatible con el formato JSON con la ventaja de que ademas de legible es fácilmente editable.

Bibliografía

Páginas de consulta sobre licencias, desarrollo y uso del software analizado