"DESARROLLO DE UN SISTEMA DE SEGURIDAD PARA UN JUEZ DE MARATONES DE PROGRAMACIÓN TIPO ACM-ICPC, QUE SOPORTE UN CONJUNTO DE ATAQUES PREVIAMENTE DELIMITADOS"

JHON BERNARDO JIMENEZ BECERRA

JOHN SEBASTIAN VAHOS HERRERA

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA

OCTUBRE DE 2013

"DESARROLLO DE UN SISTEMA DE SEGURIDAD PARA UN JUEZ DE MARATONES DE PROGRAMACIÓN TIPO ACM-ICPC, QUE SOPORTE UN CONJUNTO DE ATAQUES PREVIAMENTE DELIMITADOS"

JHON BERNARDO JIMENEZ BECERRA

JOHN SEBASTIAN VAHOS HERRERA

Informe de proyecto de grado de pregrado

Director

Ingeniero Sebastián Gómez González

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA

OCTUBRE DE 2013

DEDICATORIA

Quiero dedicar este proyecto especialmente a Dios porque me dio la oportunidad de vencer todos los obstáculos que se me fueron presentando durante el camino, dándome la oportunidad de alcanzar esta importante meta. A mi familia por brindarme el apoyo durante toda mi vida y a todos mis profesores y compañeros que hicieron parte de mi proceso de formación como profesional.

John Sebastian Vahos Herrera

AGRADECIMIENTOS

Al Ingeniero Sebastián Gómez González y a Manuel Felipe Pineda Loaiza integrantes del Semillero In-Sílico, por contribuir en el desarrollo de este proyecto con su amplio conocimiento y su experiencia en maratones de programación.

A nuestras familias por su apoyo incondicional.

Finalmente agradecemos a la Universidad Tecnológica de Pereira por los conocimientos que nos brindaron durante la carrera, con los cuales fue posible la realización de este proyecto.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Las maratones de programación son eventos que en los últimos años han tenido gran impacto dentro del campo de las ciencias de la computación, tomando cada vez mas importancia entre estudiantes y profesionales debido a que son competencias que desarrollan habilidades como el análisis de algoritmos, la resolución de problemas y el trabajo en equipo.

En este trabajo se propone el desarrollo de un sistema de seguridad para un juez de maratones de programación, es así como a lo largo del documento se expone la importancia de la realización de dicho trabajo, de la identificación de los tipos de ataques además del análisis de mecanismos que puedan neutralizarlos. Entre otros puntos importantes se documenta el diseño y la realización de las pruebas con el fin de determinar y probar si el sistema de seguridad soportaba los diferentes tipos de ataques, además de documentar el proceso de desplegar la aplicación desarrollada en un servidor en producción para llevar a cabo la primera maratón de programación de la UTP abierta a nivel mundial (*The First Open UTP Programming Contest*).

Finalmente se analizan los resultados obtenidos al momento de probar el sistema de seguridad después de la realización de una serie de ataques, mostrando excelentes resultados en la neutralización de dichos ataques. Se plantean las conclusiones generales de todo el proyecto y se establecen los proyectos futuros que se pueden derivar sobre la temática desarrollada.

CAPÍTULO 1

INTRODUCCIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA

Una maratón de programación es una competencia en la que priman el trabajo en equipo, el análisis de problemas, el conocimiento de algoritmos y el desarrollo rápido de una solución computacional. Las maratones de programación son un recurso muy importante en cuanto a la creación de conocimiento, pues para resolver los problemas se requieren un conjunto integral de habilidades, empezando por fundamentos teóricos y terminando con la implementación práctica de las soluciones a los problemas propuestos.

Las maratones de programación promueven el Aprendizaje Basado en Problemas (ABP), una metodología de enseñanza que ha cobrado importancia en los últimos años; en esta metodología el protagonista es el estudiante ya que es él quien toma la responsabilidad de aprender de manera autónoma, a diferencia de la metodología tradicional de enseñanza en donde el docente es el principal protagonista y es el encargado de transmitir el conocimiento a los estudiantes. En la metodología de enseñanza basado en problemas, cuando un estudiante se enfrenta a un problema, es posible que no tenga el conocimiento necesario para resolver dicho problema, pero a medida que éste va buscando información se va dando cuenta de lo que se requiere para atacar cada problema; y es posible que llegue a la solución él mismo. El trabajo en equipo se destaca de manera importante, ya que al compartir ideas y conocimiento, se puede tener una visión más completa de lo que se requiere para dar una solución a un problema específico, el trabajo en equipo se da entre los mismos alumnos y también entre alumnos y docentes.

Una maratón tipo ACM-ICPC[1] es una competencia que se realiza por equipos con un máximo de 3 participantes. La competencia generalmente tiene una duración de 5 horas, tiempo durante el cual los competidores deben resolver la mayor cantidad posible de problemas[2], la cantidad de problemas por competencia está entre 8 y 12. Las soluciones son enviadas a un juez[3] para su juzgamiento y recibir un veredicto[4]. Cada equipo solo cuenta

[1] ACM-ICPC ACM International Collegiate Programming Contest es una competencia de programación y algoritmia que se realiza cada año entre universidades de todo el mundo, patrocinada por IBM (International Business Machines).

[2] Algunos de los temas más importantes son: Teoría de grafos, Teoría de Números, Programación dinámica, Problemas trigonométricos, entre otros.

[3] Un juez para maratones es una aplicación que se encarga de verificar cada una de la soluciones enviadas y devuelve un veredicto para cada solución.

[4] Existen diferentes veredictos: Accepted(La solución es correcta), Wrong Answer(La solución no es correcta), Compile Error(Error de compilación), Runtime Error(Error durante la ejecución), Time Limit Exceeded(La solución se demora más del tiempo establecido), entre otros.

con 1 computador, es por esta razón que el trabajo en equipo y el uso efectivo del recurso de computo se vuelven cruciales.

Un juzgamiento comienza cuando una solución es enviada al juez, la solución puede estar escrita en diferentes lenguajes[5] de programación, luego de que se tiene el código fuente, éste es compilado y ejecutado en el juez (servidor), al ejecutable se le pasa un archivo de entrada, la salida que éste genera es comparada con la salida esperada, es decir, la salida correcta.

El proceso de juzgamiento trae consigo un problema de seguridad bastante importante, pues no se tiene control sobre lo que se puede escribir en un código fuente. Es así como autores[6] con gran conocimiento en el tema han establecido que el tema de la seguridad en los sistemas de juzgamiento se ha descuidado bastante, siendo éste uno de los principales aspectos a tener en cuenta en la construcción de un sistema para maratones de programación.

Al no tener control sobre lo que se puede escribir en un código fuente, es posible entonces que alguien pueda crear un código malicioso[7] y enviarlo al juez, con la intención de hacer algo indebido que puede ir desde tratar de hacer trampa o provocar la caída del juez en tiempo de maratón.

La disponibilidad del juez durante una competencia es crucial, ya que en cualquier momento se debe tener la posibilidad de enviar una solución, desde el primer minuto de competencia hasta el último minuto. El hecho de que por algún motivo el juez no se encuentre disponible, afectaría los resultados finales de una competencia, pues los puntajes de cada equipo se calculan teniendo en cuenta los tiempos en los que se envía cada solución.

1.1.1 Definición del problema

La compilación y posterior ejecución de un código fuente en el juez (servidor) creado por un tercero, acarrea un grave problema de seguridad; teniendo en cuenta que la disponibilidad y el correcto funcionamiento del juez son cruciales en cada maratón de programación.

[5] En las maratones tipo ACM-ICPC se permiten los lenguajes: Java y C/C++.

[6] Michal Forišek. Security of programming contest systems. Informatics in Secondarys Schools, Evolution and Perspectives.

[7] Un código malicioso es un código creado con la intención de causar algún tipo de daño informático o simplemente para hacer algo que no es permitido.

1.2 JUSTIFICACIÓN

Teniendo claro el grave problema de seguridad que representa la ejecución de un código hecho por un tercero en un servidor, y conociendo las posibles consecuencias, es necesario tratar de implementar una solución que minimice dichos riesgos de seguridad.

Con la realización de este proyecto se tienen los siguientes beneficios:

- Se tiene la disponibilidad y el correcto funcionamiento del sistema de juzgamiento.
- Las maratones de programación fomentan la generación de conocimiento y la metodología de Aprendizaje Basado en Problemas. Si bien la metodología de enseñanza tradicional ha contribuido de manera importante a la generación de conocimiento, es importante explorar otros métodos de enseñanza complementarios; tales como la metodología de Aprendizaje Basado en Problemas (ABP).
- Fortalece la participación en las maratones de programación, éstas son un motivo de reconocimiento para el programa de Ingeniería de Sistemas y computación, también para la Universidad Tecnológica de Pereira tanto a nivel Nacional como a nivel Internacional.
- Se hace un aporte importante al conocimiento, ya que los resultados producto de esta investigación pueden ser usados por otros investigadores.

Es importante resaltar el gran logro obtenido por el equipo de maratonistas UTP-01[8], el cual fue la consecución de un cupo al mundial. Éste se logró en la Maratón Regional Latinoamericana del año 2012, donde el equipo UTP-01 logró el segundo puesto, lo que significó la clasificación a las ACM-ICPC World Finals 2013[9].

[8] Integrantes: Sebastián Gómez, Santiago Gutierrez, Diego Alejandro Agudelo. Coach: Ingeniero Hugo Humberto Morales Peña.

[9] Fue llevada a cabo en San Petersburgo - Rusia del 30 de Junio al 4 de Julio.

1.3 OBJETIVO

1.3.1 Objetivo general

Desarrollar un sistema de seguridad para un juez de maratones de programación con juzgamiento ACM-ICPC, que soporte un conjunto delimitado de ataques, previamente seleccionados y clasificados.

1.3.2 Objetivos específicos

- Realizar un estudio sobre los distintos tipos de ataques identificados, sobre un sistema con juzgamiento tipo ACM-ICPC.
- Realizar un estudio sobre sistemas de juzgamiento existentes y sobre las medidas de seguridad que se tienen en cuenta para su construcción.
- Definir los requerimientos para el desarrollo de un sistema de seguridad para un juez de maratones de programación con juzgamiento tipo ACM-ICPC.
- Desarrollar el sistema de seguridad con los requerimientos previamente establecidos.
- Diseñar un conjunto de pruebas, es decir, un conjunto de ataques que abarquen los distintos tipos previamente seleccionados y clasificados.
- Implantar el sistema en un servidor para probar su funcionalidad y puesta a punto.
- Generar la documentación necesaria sobre la aplicación.
- Poner a prueba el sistema, haciendo el envío de cada uno de los códigos del conjunto de pruebas previamente diseñado, para la medición de las variables.

1.4 MARCO REFERENCIAL

1.4.1 MARCO HISTÓRICO

Como referente histórico se puede encontrar las competencias ACM-ICPC las cuales inician en la universidad A&M[10] en texas USA desde 1970, donde posteriormente en 1977[11] pasa a ser una competición con varias rondas clasificatorias organizando la final con la colaboración de la ACM Computer Science Conference[12]. Durante la década de los 80's, estas competencias solo se realizaban principalmente entre equipos de Estados Unidos y Canadá. En los años siguientes aumentó el número de participantes y por ende fueron mas los países que apoyaron este tipo de competencias. Solo durante el periodo de competencias entre 2011 y 2012 participaron 25016 participantes, para un total de 8300 equipos que representaron a 2219 instituciones en 85 países[13]. Su sede principal está ubicada en la Universidad de Baylor[14] desde 1989. Además desde el año 1997, el evento es patrocinado por la empresa *International Business Machines* [15] (IBM).

1.4.2 MARCO TEÓRICO

Principales competencias de programación a nivel mundial

A continuación se darán a conocer cuales son las principales competencias de programación a nivel mundial, con el fin de contextualizar un poco en los diversos tipos de competencias que se desarrollan año tras año.

International Olympiad in Informatics (IOI)

Esta competencia inicia en 1989, con el fin de estimular en los estudiantes el interés por las ciencias de la computación y las tecnologías de la comunicación. Los participantes son jóvenes pertenecientes a instituciones educativas de nivel de enseñanza media a nivel mundial, donde cada país participante maneja una competencia a nivel nacional con el fin de escoger cuatro estudiantes que participaran posteriormente en la IOI.

La dinámica del concurso consiste que cada participante compite de forma individual durante dos días, resolviendo por cada día tres retos computacionales de carácter algorítmico. La forma de calificar estos retos se miden con un puntaje máximo de 100 puntos para cada uno de los retos computacionales presentados en la competencia; finalmente la premiación otorga medallas a prácticamente la mitad de los participantes entre medallas de oro, plata y bronce.

[10] Pagina oficial de la universidad Texas A&M University, <http://www.tamu.edu>

[11] Dato obtenido de ACM ICPC *The Early Years ACM Contest Finals* [en línea], <http://cm.baylor.edu/ICPCWiki/Wiki.jsp?page=The%20Early%20Years>, [consultado el 25 de mayo de 2013].

[12] Es una serie de conferencias promovidas por la ACM (Association for Computing Machinery), desde el año de 1973. Estas conferencias han abarcado diferentes tópicos de interés dentro de las ciencias de la computación

[13] Cifras adquiridas de *Final UCI 2012 del ACM-ICPC*, [boletín en línea], <http://coj.uci.cu/downloads/files/finaluci2012/01_Convocatoria.pdf>, [consultado el 25 de mayo de 2013].

[14] sitio oficial, [en línea] <http://www.baylor.edu/>.

[15] Sitio oficial, [en línea], <http://www.ibm.com/us/en/>.

ACM International Collegiate Programming Contest (ACM-ICPC)

Cualquier estudiante que este matriculado en una universidad puede llegar a participar en estas competencias, habiendo restricciones sobre el número de veces que el concurso permite a cada estudiante participar en cada una de las rondas de la competencia. Los participantes compiten en equipos de tres integrantes representando solamente una institución. La configuración de las rondas esta dada, por una clasificación previa en la universidad, una posible competencia subregional, [16] una competencia regional y la gran final. Los ganadores de cada regional tienen asegurada su participación en la gran final organizada en alguno de los países miembro de la ACM-ICPC.

TopCoder Algorithm Competitions

Topcoder se caracteriza por ser un sitio online que administra diferentes maratones de programación, estableciendo en promedio competencias cada 2 semanas. Cualquier persona puede participar en Topcoder excluyendo sus propios empleados, aunque hay restricciones en las ocasiones en que se entregan premios en efectivo, donde solo pueden participar personas mayores a 18 años.

Una competencia consiste en varias rondas; en la primera ronda los participantes se les entrega tres problemas de naturaleza algorítmica con diferentes grados de dificultad y por ende diferentes puntuaciones por la solución de cada problema, contando con 75 minutos para resolverlos. Luego en la segunda ronda se da un tiempo de 5 minutos para que cada participante pueda revisar código de otros participantes, con el objetivo de encontrar fallas o errores y así poder ganar puntos extras. Por último en la tercera ronda el sistema prueba una vez mas todas las soluciones poniéndolas a prueba a través de una considerable cantidad de casos de prueba. Finalmente la puntuación se da de acuerdo a los problemas desarrollados y a las fases que halla alcanzado a superar cada participante.

Terminología común en las competencias de programación tipo ACM-ICPC

En esta sección se presentan cuales son los términos usados en el contexto de las maratones de programación tipo ACM-ICPC.

Conjunto de problemas

En cada maratón de programación cada participante se le entrega un conjunto de problemas en medio físico, con el fin de facilitar la lectura de los mismos, es importante resaltar que estos problemas están escritos en inglés, debido a las reglas de la competencia[17] que por lo general oscilan entre 8 a 12 problemas.

¿Cómo es la descripción de un problema?

Un problema se podría decir que es la unidad básica de una maratón de programación, en el cual cada grupo trabaja en equipo con el objetivo de darle solución a dicho problema. Aquí se pueden identificar varias características que trae la redacción de un problema, inicialmente los enunciados utilizados para la descripción del problema tienden a utilizar

[16] A nivel nacional, esta competencia esta organizada por la Asociación Colombiana de Ingenieros de Sistemas (ACIS) y la Red de Decanos y Directores de Ingeniería de Sistemas y Afines (*REDIS*)

[17] Sin embargo, las últimas maratones internas de la UTP, el conjunto de problemas se ha escrito en español, con el fin de facilitar la lectura de los problemas e incentivar a los nuevos estudiantes que están iniciando el proceso como maratonistas.

un conjunto específico de términos que contextualizan la narración de una historia y que en la mayoría de las veces indican una motivación para resolver el problema. Luego de la introducción del problema, se describe la forma que tienen los datos de entrada(*input*) para el programa que soluciona el problema, describiendo posteriormente el formato que tienen los datos de salida(*output*) de dicho programa y finalmente, se da un ejemplo de algunos casos de prueba indicando datos de entrada y salida del problema.

Tipos de problemas

A la hora de un participante abordar un problema, es necesario analizar inicialmente la naturaleza del problema y por ende su complejidad,[18] dependiendo de esto se podrá determinar que técnicas o algoritmos seguir con el fin de llegar a una solución óptima para el problema. Es común ver algoritmos que lógicamente llegan a la solución de un problema pero que computacionalmente no son correctos, ya que sobrepasan el tiempo necesario de ejecución o el límite de memoria establecido por la competencia.

Ahora se enunciarán varios tipos de problemas que son usualmente encontrados en las maratones tipo ACM-ICPC.

- Programación Dinámica
- La ruta mas corta
- Técnicas de búsquedas recursivas
- Problemas geométricos
- Problemas con números grandes
- Búsquedas de aproximación
- Búsquedas Heurísticas

Límites

Una maratón tipo ACM-ICPC se caracteriza por utilizar un sistema que se encarga de realizar las pruebas a las soluciones de los problemas planteados de forma automática, lo que se conoce como sistema de juzgamiento, concepto que se trabajará en las secciones siguientes. Estas pruebas automatizadas son vitales a la hora de definir los problemas de la competencia, puesto que aquí se define cuales son las condiciones mínimas para que el sistema de juzgamiento pueda aceptar la solución a un problema.

Entre las restricciones mas importantes, encontramos el *límite de memoria*, el *límite de tiempo* y el *límite de tamaño de entrada*. Los primeros dos límites son usados por el sistema de juzgamiento cuando juzga los diferentes problemas enviados por los participantes, que en la mayoría de las veces es una información que no es proporcionada por la competencia. Mientras que el límite de entrada, es un límite que es conocido por los participantes con el fin de codificar una solución que lea los datos de entrada hasta el límite establecido, que puede ser un número, un carácter o el indicador de fin de archivo EOF.

[18]Complejidad Algorítmica, [en línea], http://www.infor.uva.es/ jvalvarez/docencia/tema5.pdf

Software para maratones de programación

A continuación se enunciarán algunos de los sistemas de juzgamiento utilizados para realizar maratones de programación, estas aplicaciones se pueden descargar y además están disponibles para el público en general.

- *Programming Contest Control System*, PCˆ2, es uno de los principales y más famosos sistemas de juzgamiento de acceso público. Es utilizado para organizar maratones tipo ACM-ICPC en varias regiones del mundo. Se caracteriza por ser una aplicación que se establece sobre una arquitectura cliente-servidor con clientes para los equipos participantes en la maratón y el administrador de la competencia de programación en curso. [19]

- *Domjudge*, al igual que PCˆ2 también es muy conocido en el contexto de las maratones a nivel mundial. Es un sistema automatizado para llevar a cabo maratones de programación tipo ACM-ICPC, [20] el cual está capacitado para juzgar de forma automática las soluciones de los participantes de una maratón. Entre sus características más importantes se tiene: [21]

  - Interfaz Web.
  - Cuenta con juzgamiento distribuido.
  - Su estructura es modular.
  - Su diseño esta pensado en la seguridad.
  - Es un software de código abierto.

- *Debuxxe - Computer Olympiad Trainer*, es un sistema de juzgamiento enfocado básicamente para ayudar a entrenar y mejorar las habilidades a maratonistas en forma individual, como estudiantes que participan en la IOI. Este software de código abierto esta desarrollado en Visual Basic 6.0. [22]

- *The Moe Contest*, es un sistema de juzgamiento que se fundamenta principalmente en las competencias de la IOI, aunque su estructura modular, hace de esta herramienta muy adaptable a cualquier tipo de competencia. Es un sistema que fue desarrollado y probado en sistemas operativos tipo Unix manejando la licencia GNU2. En secciones siguientes se hablará un poco más en detalle sobre este software para maratones de programación. [23]

- *BOCA Online Contest Administrator*, es un sistema de administración de competencias de programación, principalmente de tipo ACM-ICPC.[24] Esta herramienta ha sido diseñada en PHP y PostgreSQL para su administración web donde cuenta con características como portabilidad, buen control de concurrencia y de tener una sencilla interfaz web. Este software esta liberado bajo la licencia GNU3. [25]

[19] PCˆ2, [aplicación descargable], <http://www.ecs.csus.edu/pc2/>.
[20] Es el software usado en la final de programación de la ACM-ICPC en 2012
[21] DomJudge, [aplicación descargable], <http://domjudge.sourceforge.net/intro>.
[22] [Aplicación descargable], <http://sourceforge.net/projects/debuxxe/>
[23] [Aplicación descargable], <http://www.ucw.cz/moe/>.
[24] Es el software principal donde se llevan a cabo las maratones nacionales y regionales de la ACM-ICPC en Brasil, organizadas por la Sociedad Brasileña de Computación SBC
[25] [Aplicación descargable], <http://www.ime.usp.br/ cassio/boca/>.

Principales sistemas de juzgamiento online

Actualmente en las competencias de programación, es muy común ver que varias universidades a nivel mundial crean y administran sistemas de juzgamiento disponibles para el público en general, esto con el propósito de tener un espacio de práctica para sus usuarios, donde mejoran gradualmente sus habilidades y destrezas en las maratones de programación.

Un *Online Judge System (OJs)*[26] es un sistema online que puede compilar y ejecutar códigos enviados por los participantes que están inscritos en el sistema, donde posteriormente son validados con un conjunto de casos de prueba. Estos sistemas están en la capacidad de manejar todas las restricciones como el límite de memoria, límite de tiempo y restricciones en la seguridad entre otras [27].

Entre los Online Judge que se pueden encontrar actualmente están:

- *UVa Online Judge*,[28] fue el primer sistema de juzgamiento, creado en 1995 en la universidad de Valladolid en España, actualmente es uno de los Online Judge más conocidos a nivel mundial el cual alberga la mayoría de los problemas usados en las competencias de la ACM-ICPC.
- *Peking University Online Judge (POJ)*,[29] junto con *Zhejiang University Online Judge*[30] es uno de los sistemas de juzgamiento más famosos de China, actualmente cuenta con más de 2800 problemas donde 200 de ellos están soportados para lenguajes C/C++, Pascal, Java y Python.
- *Timus Online Judge*,[31] este juez se caracteriza por tener una cantidad considerable de problemas. Su administración y mantenimiento está dado por la universidad Ural State University[32] en Rusia.
- *USACO Training Program Gateway*, [33] es un sistema de juzgamiento online dirigido principalmente a estudiantes de Estados Unidos que se preparan para el IOI.
- *Sphere Online Judge (SPOJ)*,[34] es uno de los jueces que cuenta con la mayor cantidad de usuarios, aproximadamente 100000 usuarios registrados y con un archivo de más de 10000 problemas soportados para 40 lenguajes de programación.
- *Caribbean Online Judge (COJ)*,[35] es uno de los jueces existentes a nivel latinoamericano el cual está en línea desde el año 2010. Es una excelente herramienta para los participantes de América Latina puesto que cuenta con su módulo de administración en español y portugués.

[26] Término muy común en la literatura de los sistemas de juzgamiento
[27] Peng Ying, Wang Fang. *Development and Application of Online Judge System*
[28] [Aplicación en línea], <http://uva.onlinejudge.org/>.
[29] [Aplicación en línea], <http://poj.org/>.
[30] [Aplicación en línea], <http://acm.zju.edu.cn/onlinejudge/>.
[31] [Aplicación en línea], <http://acm.timus.ru/>.
[32] Sitio oficial, [en línea], http://www.eng.usu.ru/usu/opencms/
[33] [Aplicación en línea], <http://cerberus.delos.com:790/usacogate>.
[34] [Aplicación en línea], <http://www.spoj.com/>.
[35] [Aplicación en línea], <http://coj.uci.cu/index.xhtml>.

- *TJU Online Judge*,[36] sistema de juzgamiento administrado por la universidad Tianjin University de China, este juez brinda una variedad de problemas tipo ACM-ICPC.
- *HDU Online Judge System*,[37] perteneciente a la universidad Hangzhou Dianzi University de China.

Características comunes de los Online Judge

Entre los sistemas de juzgamiento nombrados, además de ser aplicaciones web, en la gran mayoría se pueden ver ciertas características que son comunes.

- Poseen un archivo de problemas que abarcan los diferentes tipos enunciados en la sección 1.4.2.
- Tienen la capacidad de habilitar sus propias maratones de programación.
- Tienen sus propios casos de prueba, para juzgar las soluciones de los diferentes problemas.
- Manejan estadísticas, con el fin de comparar el desempeño de los usuarios.
- Cada participante puede manejar una cuenta de usuario con el fin de participar en las diferentes competencias del online judge.

1.4.3 MARCO CONCEPTUAL

- Casos de prueba: Consiste en archivos de entrada y salida, con el fin de probar el envío de una solución de un problema propuesto durante una competencia de programación. El archivo de entrada, son todos los datos suficientes que el algoritmo desarrollado debe de recibir, con el fin de dar unos datos de salida, de acuerdo a la solución propuesta. Estos se comparan con las respuestas alojadas en el juez, debiendo ser iguales para que el programa sea correcto.
- ACM: *Association for Computing Machinery* es una organización a nivel mundial que reúne toda una sociedad científica y educativa en torno a las ciencias de la computación. Publica revistas y periódicos científicos, además de apoyar el desarrollo de conferencias y eventos como el *ACM International Collegiate Programming Contest*.
- ICPC: *International Collegiate Programming Contest* es una competencia de programación anual entre diferentes universidades del mundo. Sus participantes deben de tener amplios conocimientos sobre algún lenguaje de programación[38], y el manejo de algoritmos. Este tipo de eventos es patrocinado principalmente por IBM *International Business Machines (IBM)* y organizado por la ACM. En la competición prima básicamente el trabajo en equipo, el análisis de problemas y el desarrollo rápido de software.

[36] [Aplicación en línea], <http://acm.tju.edu.cn/toj/>.
[37] [Aplicación en línea], <http://acm.hdu.edu.cn/>.
[38] Durante una competencia de programación, es muy usual ver que los problemas se solucionen en lenguajes como JAVA y C++

- Código Malicioso[39]: se considera código malicioso cualquier fragmento de código, que en el momento de ejecutarse cause algún tipo de problema en el sistema de cómputo en el que se ejecuta, interfiriendo en el debido funcionamiento del mismo[40].
- Desarrollo ágil de software: Son métodos de ingeniería del software que básicamente permite el desarrollo iterativo e incremental donde hay una constante evolución entre los requerimientos y las evoluciones mediante la colaboración de grupos organizados y multidisciplinares. Buscando que todos los esfuerzos se empleen en la creación del mejor software que satisfagan la creación del cliente [41].
- Aprendizaje basado en problemas: Es un método de aprendizaje donde se introducen problemas relevantes utilizados para proporcionar el contexto y la motivación necesaria para el proceso de aprendizaje[42].

[39]También puede ser llamados badware, software malicioso o software malintencionado

[40]Presentación Modulo 1 - Malware ESR II Lic. Fernando Alonso, CISSP, CCNA, GSEC

[41]Colusso Ricardo y Gabardini Juan "Desarrollo Agil de Software, una introducción a las metodologías agiles de desarrollo de software"

[42]Patricia Morales Bueno y Victoria Landa. "Aprendizaje Basado en Problemas"

CAPÍTULO 2

METODOLOGÍA DE LA INVESTIGACIÓN

2.1 DISEÑO METODOLÓGICO

2.1.1 Hipótesis

Es posible desarrollar un sistema de seguridad para un juez con juzgamiento ACM-ICPC que soporte un conjunto delimitado de ataques, previamente clasificados (Ver población) sin que ello impida el correcto funcionamiento del mismo.

2.1.2 Tipo de investigación

En esta investigación se utilizará un enfoque cuantitativo.

2.1.3 Población

Cada una de las peticiones que llegan al sistema para ser juzgadas, clasificados en peticiones No-ataques[43] y peticiones Ataques[44]. En la sección 3.4 se establecen tres tipos de ataques en base al momento en que ocurren, *tiempo de compilación*, *tiempo de ejecución* y *tiempo de competencia*. Esta investigación se va a centrar en los dos primeros tipos de ataques, los cuales ocurren durante la compilación y la ejecución de cada código fuente.

Las peticiones ataque están contemplados en la siguiente clasificación:

Ataques en tiempo de compilación

- Forzar un alto consumo de tiempo durante la compilación
- Consumo de recursos durante la compilación
- Acceso a material restringido

Ataques en tiempo de ejecución

[43]Son peticiones que no tienen la intención de generar algún tipo de daño al sistema
[44]Son peticiones creadas con la intención de generar algún tipo de daño al sistema

- Acceso a material restringido
- Modificando o dañando el entorno de pruebas (Juzgamiento)
- Saltarse la medición del tiempo (Time Limit)

2.1.4 Variables

Las variables a tener en cuenta son:

- Estado del sistema de juzgamiento después de cada intento de ataque.
- Tiempo de respuesta por cada petición al sistema de juzgamiento.

Se consideran dos estados posibles para el sistema después de cada intento de ataque:

- El sistema funciona correctamente, es decir, el sistema puede atender más peticiones.
- El sistema no se encuentra disponible, es decir, el sistema no puede atender más peticiones.

2.1.5 Unidad de análisis

Peticiones con las condiciones contempladas en la población.

2.1.6 Definición de las muestras

Se tomará 1 código fuente considerado No-ataque, dicho código fuente debe ser la solución correcta a un problema, es decir, con veredicto *YES*, con el cual se harán 100 peticiones al sistema, tomando así una muestra etiquetada como M_1. Luego se tomarán 20 códigos fuentes considerandos Ataques, diseñados de tal forma que abarquen los distintos tipos de ataques descritos en la población y por cada código se hará 1 petición, lo que equivale a 20 peticiones ataques en total. Para después tomar el mismo código fuente considerado No-ataque y realizar 100 peticiones y así tomar la segunda muestra etiquetada como M_2. Todos los códigos fuente estarán escritos en lenguaje C++.

2.2 PROCEDIMIENTO ESTADÍSTICO PARA EL TRATAMIENTO DE LOS DATOS

En esta sección se pretende ilustrar un procedimiento mucho mas formal, con el propósito de tomar una decisión objetiva de acuerdo a los datos recopilados en el experimento. Se sabe que el papel de la estadística en la ingeniería es crucial, puesto que permite tomar decisiones respecto al objeto de análisis, en este caso poder validar la hipótesis de estudio formulada en 2.1. Es así como se hace uso de técnicas de la estadística inferencial para validar el experimento a través de la estimación de parámetros estadísticos y la formulación de una hipótesis estadística la cual es una conjetura con respecto a una o más poblaciones.

Durante este procedimiento se mostrará el uso de la prueba T combinada de dos medias, procedimiento estadístico que permitirá verificar si efectivamente el sistema de juzgamiento es capaz de soportar los diferentes tipos de ataques establecidos previamente.

La recolección de datos, se divide en dos conjuntos o muestras que se se etiquetan de la siguiente manera:

- M_1 : conjunto de datos tomados del sistema libre de ataques
- M_2 : conjunto de datos tomados del sistema atacado

La primera muestra M_1 consiste en obtener los tiempos que demora el sistema en dar respuesta, el tamaño de la muestra está dado por $n_1 = 100$, y la segunda muestra o conjunto de datos M_2 serán los tiempos que demora el sistema en dar repuesta al mismo problema, pero esta vez el sistema de juzgamiento ha sido atacado previamente con el conjunto de ataques descritos mas adelante en el capítulo 7, para esta segunda muestra se considerará también un tamaño de muestra $n_2 = 100$.

Teniendo los datos de M_1 y M_2 se procede a medir sus respectivas medias $\bar{X}_1$ y $\bar{X}_2$ y desviaciones estándar S_1 y S_2.

Con lo anterior se pretende comprobar si existen diferencias significativas entre los tiempos medios de repuesta a un problema juzgado, teniendo en cuenta que:

- Se supone que las poblaciones muestreadas tienen una distribución normal, esto gracias al teorema 2.2.1, el cual se nombra como *teorema del límite central*.
- Las dos muestras tomadas son independientes, es decir, la primera muestra no influye en los datos tomados en la segunda muestra.
- Se supone que las desviaciones estándar de ambas poblaciones son iguales.

Teorema 2.2.1. *Sea $X_1, X_2 \ldots, X_n$ un conjunto de variables aleatoria, independientes e idénticamente distribuidas de una distribución con media μ y varianza $\sigma^2 \neq 0$. Entonces, si n es suficientemente grande, la variable aleatoria.*

$$\bar{X} = \frac{1}{n} \sum_{i=0}^{n} X_i$$

tiene aproximadamente una distribución normal con $\mu_{\bar{X}} = \mu$ y $\sigma^2_{\bar{X}} = \sigma^2$

Con el experimento se desea establecer que no existe diferencia significativa entre los tiempos medios de respuesta a un problema juzgado con el sistema libre ataques y el sistema atacado, por lo tanto se toma μ_1 y μ_2 como las medias poblacionales de los tiempos medios del sistema libre de ataques y la media de los tiempos del sistema atacado respectivamente. Con esta notación el anterior contraste de hipótesis equivale a formular:

- H_0: No existe una diferencia significativa en los tiempos de respuesta entre el sistema libre de ataques y el sistema atacado.

- H_1: Sí existe una diferencia significativa en los tiempos de respuesta entre el sistema libre de ataques y el sistema atacado.

Formalmente se tiene:

$$\begin{cases} H_0 : \mu_1 - \mu_2 = d_0 \\ H_1 : \mu_1 - \mu_2 < d_0 \end{cases} \tag{2.1}$$

Donde $d_0 = 0$, esto debido a que se va a probar que las medias poblacionales μ_1 y μ_2 son iguales dentro de la hipótesis nula (H_0). En la hipótesis alternativa (H_1) se encuentra $\mu_1 - \mu_2 < d_0$, esto se da porque se espera que el tiempo medio μ_2 del sistema atacado sea mayor.

Ahora teniendo en cuenta el siguiente teorema:

Teorema 2.2.2. *Si se extraen al azar muestras independientes de tamaños n_1 y n_2 de dos poblaciones, discretas o continuas, con media μ_1 y μ_2 y varianzas σ_1^2 y σ_2^2 respectivamente, entonces las distribución muestral de las diferencias de las medias $\bar{X}_1 - \bar{X}_2$, está distribuida aproximadamente de forma normal con media y varianza dadas por*

$$\mu_{\bar{X}_1 - \bar{X}_2}, \quad y \quad \sigma^2_{\bar{X}_1 - \bar{X}_2} = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}$$

De aquí,

$$Z = \frac{(\bar{X}_1 - \bar{X}_2) - (\mu_1 - \mu_2)}{\sqrt{(\sigma_1^2/n_1) + (\sigma_2^2/n_2)}}$$

es aproximadamente una variable normal estándar.

$$\bar{X}_1 - \bar{X}_2 \simeq N\left(\mu_1 - \mu_2, \sqrt[2]{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}\right)$$

Se podría tomar el estadístico Z para llevar a cabo el experimento y probar la hipótesis establecida, sin embargo las varianzas poblacionales σ_1^2 y σ_2^2 son desconocidas, en consecuencia con fines prácticos de este experimento se suponen las varianzas poblacionales iguales $\sigma_1^2 = \sigma_2^2$ a lo que lleva usar la *Prueba T combinada de dos muestras*, de manera que se utilizará una distribución t de Student con v grados de libertad.

Entonces el estadístico de prueba estaría definido así:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - d_0}{s_p\sqrt{1/n_1 + 1/n_2}} \tag{2.2}$$

Donde s_p^2 es la varianza ponderada definida como:

$$s_p^2 = \frac{s_1^2(n_1 - 1) + s_2^2(n_2 - 1)}{n_1 + n_2 - 2} \tag{2.3}$$

Y los grados de libertad se definen:

$$v = n_1 + n_2 - 2 \tag{2.4}$$

Con el objetivo de validar el experimento y de comprobar la hipótesis de que no hay diferencia entre las medias de los tiempos entre las dos poblaciones se trabajará con un nivel de significación del 5 % lo cual equivale $\alpha = 0{,}05$.

Ahora solo queda definir la región crítica en la cual se decide si rechazar la hipótesis nula H_0 a favor de la hipótesis alternativa H_1 o por el contrario no rechazar la hipótesis nula. Por consiguiente se tiene una hipótesis unilateral con su región critica en la cola izquierda gracias a como se define en 2.1, por la desigualdad dada en H_1 al tomar los valores menores al parámetro d_0.

Entonces teniendo en cuenta α con v grados de libertad definido por la ecuación 2.2, se calcula el valor crítico $t_{\alpha,v}$ con la tabla t de Student, comparado con t dado por la ecuación 2.2. No se rechaza H_0 si:

$$t > -t_{\alpha,v}$$

Se rechaza H_0 a favor de H_1 si:

$$t < -t_{\alpha,v}$$

CAPÍTULO 3

TIPOS DE ATAQUES EN LOS SISTEMAS DE JUZGAMIENTO

3.1 INTRODUCCIÓN

Al hablar de sistemas seguros es importante dar una conceptualización sobre el campo de la seguridad y los diferentes tipos de ataques que hacen que estos sistemas no sean seguros. Es por eso que en este capítulo se mostrará en la primera sección con un concepto muy general, la seguridad informática, abarcando sus aspectos fundamentales: confidencialidad, integridad y disponibilidad, para luego contextualizar lo que son los ataques informáticos y sobre sus diferentes etapas.

Por otro lado, se hace una descripción de algunas técnicas usadas para atacar sistemas de juzgamiento y finalmente, se presentan los diferentes tipos de ataques que son comunes en los sistemas de juzgamiento, dando una perspectiva general desde el análisis hecho en la tesis doctoral de Michal Forišek,[45] autor muy referenciado en cuanto a literatura sobre sistemas de juzgamiento a nivel mundial.

3.2 SEGURIDAD

3.2.1 Seguridad Informática

La seguridad informática tiene como objetivo mantener la integridad, los aspectos fundamentales de la privacidad, el control y la autenticidad de la información manejada en un medio como el computador. [46]

Por lo tanto la seguridad informática se refiere en la manera de salvaguardar los datos en un sistema informático. No obstante este concepto no suele ser muy concreto puesto que la seguridad informática abarca diversos enfoques en áreas relacionadas con los sistemas de información, ya que se puede enfocar en la protección de los medios físicos que alojan estos sistemas, en la infraestructura de las redes que comunican la información o simplemente en la estructura de los sistemas diseñados para manipular la información.

[45] Michal Forišek, Theoretical and Practical Aspects of Programming Contest Ratings, Ph.D. thesis, Comenius University Bratislava, 2009

[46] ALDEGANI, Gustavo. Miguel. Seguridad Informática MP ediciones. Argentina. 1997 Página 22.

La seguridad informática abarca tres aspectos fundamentales, la confidencialidad, la integridad y disponibilidad, donde las políticas de seguridad de una organización giraran en torno a estos tres principios, unos más que otros dependiendo del tipo de sistema de información utilizado.

- *Confidencialidad*, este principio establece que el sistema debe asegurar que las personas no autorizadas no deben de tener acceso a la información almacenada allí, además de restringir el acceso a los usuarios del sistema de acuerdo a su perfil.
- *Integridad*, asegura que la información almacenada en un sistema no sea modificada, eliminada o alterada ya sea por el propio sistema o por personas ajenas a él, por lo tanto este principio garantiza que la información es verídica en todo momento.
- *Disponibilidad*, determina el grado en que la información se encuentra en su lugar, garantizando a los usuarios autorizados un tiempo aceptable de acceso al sistema con todos sus servicios funcionando correctamente.

Además de estos tres principios a nivel de seguridad informática se manejan otras propiedades como la autenticidad la cual garantiza que el origen de la información proviene de una fuente autentica. Se tiene también la imposibilidad de rechazo, indicando que el emisor o receptor de la información no pueda negar que no envió o recibió la información. Por otro lado está la consistencia, que garantiza que el sistema mantenga su funcionalidad durante el tiempo y por último la propiedad de aislamiento impidiendo que personas no autorizadas ingresen al sistema, principio análogo a la confidencialidad, aunque es un concepto que se centra principalmente al acceso al sistema que a la misma información almacenada allí.[47]

3.2.2 Ataques Informáticos

Según [48] los ataques son definidos como técnicas que los atacantes utilizan para explotar las vulnerabilidades en las aplicaciones informáticas. Muchas veces cuando se habla de términos como ataque y vulnerabilidad se tienden a confundir, quizás porque van muy relacionados uno del otro, pero es debido diferenciar que un ataque es precisamente lo que un atacante puede hacer, en cambio las vulnerabilidades se consideran las debilidades que tienen las aplicaciones.

A la hora de la realización de desarrollos de servicios web y otros recursos basados en Internet, nace una gran preocupación, el tema de la seguridad. Diariamente se reportan ataques a redes informáticas que gradualmente vulneran a un grado mayor el sistema atacado.

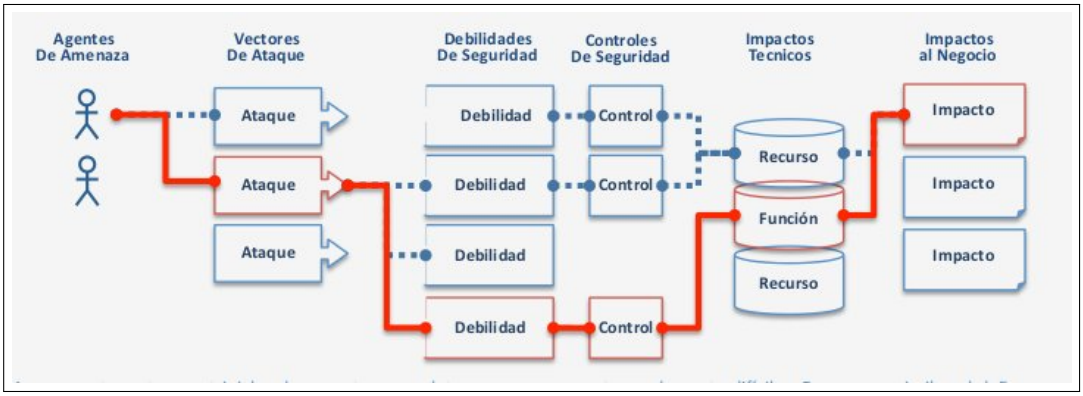
Por lo tanto las aplicaciones basadas en servicios Web tienen la necesidad de garantizar el valor esperado por sus usuarios, como también mecanismos de confianza que garanticen su seguridad.

[47] Cristhian V. Llumiquinga M., Patricio F. Vallejo. *Diseño de las políticas de seguridad de la información y desarrollo del plan de contingencia para el área de sistemas de la cooperativa de ahorro y crédito alianza del valle.*

[48] Definición de Ataque, [en línea], <https://www.owasp.org/index.php/Category:Attack>, [Consultado el 22 de junio de 2013].

Estos ataques hacen que los administradores de estos sistemas gasten horas y en ocasiones días enteros, reconfigurando el sistema con el fin de lograr la confianza en la integridad del mismo.

Figura 3.1: Riesgos de seguridad en aplicaciones, imagen tomada de OWASP [49]



Un ataque informático por lo tanto es aprovechar alguna debilidad o vulnerabilidad que puede existir a nivel de software o hardware, en la figura 3.1 claramente se puede evidenciar como las aplicaciones poseen una cantidad determinada de vulnerabilidades, dando oportunidades a los atacantes de aprovechar una de tantas evadiendo así los controles de seguridad y provocando finalmente un grave impacto en el negocio. [50] En estos ataques se busca entonces un beneficio que por lo general es de índole económico, corrompiendo la seguridad del sistema atacado y por ende afectando los activos de las organizaciones. Es necesario identificar las diferentes etapas que conforman un ataque informático, esto brinda la ventaja de aprender a pensar como los atacantes, que desde el punto de vista del profesional de seguridad, es de gran ayuda a la hora de neutralizar sus ataques. Estas etapas son:

- Reconnaissance (Reconocimiento): En esta etapa el atacante obtiene información respecto a una víctima potencial ya sea una persona o una organización.
- Scanning (Exploración): En esta segunda etapa, el atacante después de conseguir la información relevante durante la primera etapa, es utilizada para sondear el objetivo, tratando de obtener información sobre el sistema víctima como direcciones IP, nombres de host datos de autenticación entre otros.[51]
- Gaining Access (Obtener acceso): En este punto el ataque comienza a materializarse, explotando las vulnerabilidades y defectos del sistemas descubiertos durante las etapas de reconocimiento y exploración.[52]

[49] *The Open Web Application Security Project* proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

[50] Ataques informáticos Mieres, Jorge *Debilidades de seguridad comúnmente explotadas* Disponible en <www.evilfingers.com/publications/white_AR/01_Ataques_informaticos.pdf>

[51] En esta etapa el atacante puede usar herramientas como *network mappers, port mappers, network scanners, port scanners, y vulnerability scanners.*

[52] Algunas de las técnicas que el atacante puede utilizar son ataques de Buffer Overflow, de Denial of Service (DoS), Distributed Denial of Service (DDos), Password filtering y Session hijacking.

- Maintaining Access (Mantener el acceso): En esta etapa el atacante ya está dentro del sistema, lo siguiente que busca es tener un acceso al sistema en el futuro desde cualquier lugar con acceso a internet.
- Covering Tracks (Borrar huellas): Una vez que el atacante ha logrado obtener y mantener el acceso al sistema, por consiguiente intentará borrar todas las huellas que dejó durante la intrusión, para así evitar ser detectado por el profesional de seguridad o los administradores de la red. En consecuencia, buscará eliminar los archivos de registro (log) o alarmas del Sistema de Detección de Intrusos (IDS).

3.3 TÉCNICAS DE ATAQUES IDENTIFICADOS EN LOS SISTEMAS DE JUZGAMIENTO

En esta sección se hace una descripción de las principales técnicas de ataques usados en los sistemas de juzgamiento, los cuales se mencionan en la próxima sección 3.4.

3.3.1 Ataque de denegación de servicios.(DoS)

Consiste en un ataque que se genera a un sistema o red de computadoras, causando principalmente que un recurso sea inaccesible a los usuarios propios de dicho sistema. Este tipo de ataques han ido evolucionando gracias al progreso que ha tenido la internet en los últimos años, encontrándose que tradicionalmente ataques como SYN Flooding e ICMP Flooding son llevados sobre la capa de red y de transporte del modelo OSI. [53]

Aunque los ataques por acceso web vulneran principalmente la capa de aplicación, es común ver que los atacantes envían, ya sea una gran cantidad de peticiones HTTP[54] al servidor web o provocan que el servidor corra scripts complejos consumiendo los recursos limitados del servidor web y así lograr que la capacidad de los servicios del servidor disminuyan o sean denegados.[55]

En el momento de identificar este tipo de ataques se encuentran tres tipos básicos de denegación de servicio:

- Consumo de recursos: El atacante trata de consumir la máxima cantidad de recursos que pueda del servidor con el propósito de agotarlos.[56]
- Destrucción o alteración de la configuración: Aquí el atacante busca principalmente alterar la configuración de la máquina y por lo general los métodos utilizadas requieren de técnicas avanzadas.
- Destrucción o alteración física de los equipos: El propósito es buscar la denegación de los servicios del servidor destruyendo físicamente el servidor o varios de sus componentes.[57]

[53]OSI (Open System Interconnection): Es un marco de referencia para la definición de las arquitecturas en interconexión de los sistemas de comunicaciones

[54]Hypertext Transfer Protocol o HTTP: Protocolo de red usado en todas las transacciones de la World Wide Web

[55]Jianpeng Zhao,Shize Guo,Kangfeng Zheng,Xinxin Niu,Yao Jiang. An Active Defense Model for Web Accessing Dos Attacks

[56]Este tipo de recursos son el ancho de banda, tiempo de cpu, la capacidad de memoria etc.

[57]Esto se podría conseguir por ejemplo cortando el cable de conexión o de la fuente eléctrica.

3.3.2 Escalada de privilegios

El escalamiento de privilegios o escalada de privilegios es un ataque que consiste en aprovechar ya sea un error, una mala configuración o un mal diseño, bien sea dentro de un sistema operativo o dentro de una aplicación, de tal manera que se logre obtener acceso a recursos que requieren mayores privilegios de los que se tienen, es decir, que es posible realizar acciones no autorizadas o acciones que requieren mayores privilegios.

Muchos sistemas o aplicaciones, han sido creados para que múltiples usuarios los usen, en estos sistemas se habla de privilegios, que generalmente son de lectura, escritura y ejecución sobre sistemas de archivos o programas. La escalada de privilegios, significa que un usuario que no está autorizado para acceder a esos recursos lo logra, pudiendo entonces disponer de datos, ya sea para modificarlos o borrarlos, o simplemente tener acceso de lectura y en el caso de programas la posibilidad de instalar o desinstalarlos.

Este tipo de ataques se han clasificado dentro de dos categorías:

Escalada de privilegios vertical: Este es quizás el más peligroso y consiste en tratar de obtener acceso a un sistema a nivel del usuario administrador, haciéndolo desde un usuario con menos privilegios. Si un ataque de este tipo tiene éxito, el atacante tendría acceso total al sistema.

Escalada de privilegios horizontal: Este ataque consiste en tratar de obtener acceso a otra cuenta de otro usuario, con el propósito de tener acceso a sus datos; es de resaltar que ambas cuentas, tanto desde donde se hace el ataque como la atacada, se encuentran en un mismo nivel de privilegios.

3.3.3 Ataques destructivos

El único propósito de este tipo de ataque es dañar de alguna manera un sistema, impidiendo su correcto funcionamiento de manera parcial o total. Generalmente estos ataques se hacen eliminando, modificando o mediante la corrupción de archivos importantes para el correcto funcionamiento del sistema.

3.3.4 Ataques No destructivos

A diferencia de los ataques destructivos, estos ataques no buscan dañar el sistema. Su propósito es simplemente intentar sacar ventaja de alguna manera, sin que ello implique causar algún daño en el sistema. Por ejemplo desde un programa enviado por un participante, se podría intentar copiar la salida correcta y hacer creer que esa salida la dio el programa del participante, obviamente si el sistema permite esto, se tendría un veredicto de Accepted, un código de ejemplo concreto se muestra en 7.1.3.

3.3.5 Explotación de canales encubiertos de comunicación

Un canal encubierto de comunicación es un canal por medio del cual se puede transferir información, utilizando algún medio, el cual no ha sido destinado con tal propósito por el diseñador del sistema.

Para aterrizar este concepto en las competencias de programación, el uso de un medio para compartir información podría ser un Socket,[58] mediante el cual se pueda acceder a información restringida, por ejemplo acceder a los casos de prueba o a las soluciones de otros participantes. Obviamente un socket fue creado con el propósito de servir como canal de comunicación para compartir información a nivel general, pero para el caso particular de un sistema de juzgamiento se debe tener cuidado con este tipo de comunicación, por los problemas de seguridad que traen.

3.4 CLASIFICACIÓN DE LOS TIPOS DE ATAQUES EN JUECES DE MARATONES DE PROGRAMACIÓN

En los sistemas de juzgamiento para maratones de programación se pueden identificar varios tipos de ataques, que gracias al trabajo realizado por Forišek, [59] se logra formar una base de conocimiento dentro del ámbito de la seguridad en las maratones de programación. Inicialmente se pueden identificar tres categorías de ataques, basados principalmente en el tiempo en que se pueden llegar a dar estos ataques:

- *Ataques en tiempo de compilación*: Se da cuando un participante envía una solución a un determinado problema y este es compilado en el juez.
- *Ataques en tiempo de ejecución*: Estos ocurren cuando los códigos enviados por los participantes están siendo ejecutados en el juez, para evaluar las soluciones a los problemas propuestos.
- *Ataques en tiempo de competencia*: Estos ataques pueden ocurrir en cualquier momento dentro del rango de tiempo que se desarrolla una maratón de programación.

Adicionalmente algunos de estos ataques, clasifican perfectamente dentro de categorías ya conocidas en el contexto de los ataques informáticos y que previamente fueron definidos en este capítulo como los ataques de denegación de servicios y ataques de escalada de privilegios, los cuales son un ejemplo claro de tipos de ataques que se pueden dar dentro de una maratón de programación. A continuación se nombran algunos de estos ataques donde están clasificados por el tipo de ataque que previamente fueron definidos, siguiendo con una descripción y por último dando formas de prevenir cada ataque.

[58] Socket documentación oficial, [en línea] <http://man7.org/linux/man-pages/man7/unix.7.html>.

[59] Michal Forišek. 'Security of programming contest systems. Informatics in Secondarys Schools, Evolution and Perspectives'

Tabla 3.1: Descripción ataque: Consumo de tiempo durante la compilación

Forzar un alto consumo de tiempo durante la compilación	
Tipos de ataque	Tiempo de compilación, Denegación de servicios
Descripción	Un programa podría usar un indeterminado tiempo durante la compilación, Si no se maneja este tiempo de compilación, esto podría ser usado para bloquear el compilador poniendo en peligro el proceso de pruebas automatizado que manejan los sistemas de juzgamiento.
Prevención	Configurar un límite de tiempo de compilación previene este tipo de ataque. Además si existen varias maquinas atendiendo las soluciones en paralelo ayuda a mitigar los efectos de este ataque.

Tabla 3.2: Descripción ataque: Consumo de recursos durante la compilación

Consumo de recursos durante la compilación	
Tipos de ataque	Tiempo de compilación, Denegación de servicios
Descripción	Cuando al compilador se le da un archivo grande, es muy posible que tenga que leerlo todo en memoria. Por ende si el sistema operativo es obligado a ejecutar fuera de la memoria esto podría tener un resultado impredecible e incluso fatal para el sistema. Por ejemplo esto causaría que cualquier proceso pueda morir al requerir mas memoria que ya esta agotada.
Prevención	Para prevenir este tipo de ataque se haría necesario ejecutar las soluciones de los participantes dentro de un ambiente restringido, donde el compilador solo tiene acceso a una parte limitada de los recursos del sistema.

Tabla 3.3: Descripción ataque: Acceso a material restringido

Acceso a material restringido	
Tipos de ataque	Tiempo de compilación/Tiempo de ejecución, Denegación de servicios
Descripción	Busca como objetivo acceder a recursos no autorizados. Por ejemplo durante una competencia el atacante podría lograr el acceso a los casos de prueba logrando así tomar las soluciones, con el objetivo de hacer trampa durante la competencia
Prevención	En el diseño del sistema debería ser de tal forma que cuando se ejecuta un programa en el sistema de juzgamiento el participante no tenga acceso en lo absoluto al material restringido.

Tabla 3.4: Descripción ataque: Saltarse la medición del tiempo (Time Limit)

Saltarse la medición del tiempo (Time Limit)	
Tipos de ataque	Tiempo de ejecución, otros
Descripción	Cuando un juez evalúa un algoritmo, este cuenta con un tiempo para ser ejecutado, esto con el fin de conseguir algoritmos óptimos a los problemas planteados durante una maratón. Un ejemplo sencillo de este ataque es por ejemplo suponer que el sistema mide el tiempo de duración del proceso que se inicia al ejecutar el algoritmo. Esta forma de evaluar el tiempo podría ser vulnerada, si el atacante crea un proceso hijo del proceso inicial, por lo tanto ahora el programa tendría tiempo ilimitado para evaluar los casos de prueba.
Prevención	El diseñador del sistema debe conocer muy bien el funcionamiento interno del sistema y saber que módulos usar en determinadas competencias. Debe también asegurar de que el sistema puede medir exactamente el tiempo de los procesos ejecutados. Adicionalmente es recomendable obligar los programas de los participantes a que solo puedan ejecutarse en un solo hilo de ejecución evitando así llamadas al sistema por medio de comandos como *fork() y clone()*.

Tabla 3.5: Descripción ataque: Modificando o dañando el entorno de pruebas

Modificando o dañando el entorno de pruebas	
Tipos de ataque	Tiempo de ejecución, DoS
Descripción	Este tipo de ataques busca vulnerar el entorno de pruebas, encargado de comparar las soluciones de los algoritmos enviados por los participantes. Muchas son las formas de realizar este ataque en un sistema de juzgamiento, por ejemplo, un simple programa que se encarga de eliminar todo lo que pueda dentro del sistema; otro ejemplo se presenta cuando se guardan soluciones precalculadas en un archivo temporal y en un próximo envío se usa con el fin de violar el tiempo limite de ejecución.
Prevención	Desde la Configuración de un entorno restringido el diseñador del sistema de juzgamiento debe ser muy cuidadoso para evitar este tipo de problemas.

Tabla 3.6: Descripción ataque: Uso indebido de la red

Uso indebido de la red	
Tipos de ataque	Tiempo de competencia, Escalada de privilegios
Descripción	Si un participante puede monitorear y almacenar el tráfico de red, especialmente el tráfico que ingresa al sistema de juzgamiento podría ser capaz de interceptar envíos de otros participantes para posteriormente enviarlos como propios. Por otro lado cuando existen competencias que se desarrollan con todos los participantes reunidos en un mismo sitio se puede llegar a presentar comunicación entre ellos en la red o el uso indebido de contraseñas para acceder a las soluciones de otros participantes.
Prevención	Maneras de prevenir estas irregularidades es por ejemplo realizar la comunicación de forma segura, por medio del protocolo HTTPS. Por otro lado las competencias que se desarrollan en un mismo sitio los organizadores pueden tomar medidas preventivas como la configuración del hardware de red permitiendo que haya solo comunicación entre cliente y servidor, hacer seguimiento de tráfico de red por parte de los organizadores y el llevar registros de las acciones que hacen todos los participantes en sus respectivos equipos.

Tabla 3.7: Descripción ataque: Explotación de canales encubiertos de comunicación

Explotación de canales encubiertos de comunicación	
Tipos de ataque	Tiempo de competencia
Descripción	Es muy común ver que un sistema de juzgamiento envíe información al participante sobre el estado de la solución presentada, con el fin de evaluar su éxito en la resolución de los problemas. Muchas veces esta información suministrada puede dar a conocer un poco más, como información referente a los casos de prueba, y entre más información sea suministrada (estado de ejecución, tiempo y memoria consumida por el programa) alguien podría realizar gracias a esta información programas con lógica incorrecta pero que da soluciones correctas como por ejemplo un programa que retorna la secuencia de palabras *SI* y *NO*.
Prevención	En muchos sistemas de juzgamiento no es posible mitigar esta vulnerabilidad. Sin embargo es posible reducir el éxito de este tipo de ataques por medio de la creación de buenos casos de prueba.

Tabla 3.8: Descripción ataque: Uso indebido de servicios adicionales

Uso indebido de servicios adicionales	
Tipos de ataque	Tiempo de competencia, Denegación de servicios
Descripción	Este tipo de ataques se presenta principalmente en competencias que se organizan en un mismo sitio. Allí es común ver servicios de impresión de código, con el fin de hacer depuración, y la facilidad de generar copias de seguridad. Un participante podría enviar una petición de impresión de un archivo de texto de 1000 paginas consumiendo todo el papel de la impresora y negando el servicio a otros participantes. Y en cuanto a la creación de copias de seguridad un atacante podría tratar de salvaguardar una cantidad inconsiderada de datos provocándole al sistema quedar sin espacio en disco.
Prevención	Es aconsejable la fijación de un limite sobre las paginas que puede imprimir un participante en una competencia de programación, además del tamaño permitido para la creación de sus copias de seguridad.

Tabla 3.9: Descripción ataque: Explotación de errores en el sistema operativo

Explotación de errores en el sistema operativo	
Tipos de ataque	Tiempo de ejecución, Tiempo de competencia, Escalada de privilegios
Descripción	Si un participante es capaz de acceder al servidor en donde el sistema de juzgamiento prueba las soluciones de los participantes en competencia podría tratar de escanear la máquina en busca de vulnerabilidades para tratar de explotarlas. Como los programas de los participantes se ejecutan dentro del sistema con los privilegios de algún usuario local es considerado un gran riesgo de seguridad, considerado de mayor riesgo que un acceso desde el exterior a los servicios de la máquina. Por lo tanto un participante podría ejecutar cualquier código arbitrariamente dando lugar a la exposición de los casos de prueba hasta el punto de poder llegar a dañar el sistema por completo.
Prevención	Se debe aislar por completo las partes del sistema que no deberían ser accedidas por un participante. Donde la comunicación debe ser a través de una interfaz para todo el sistema de juzgamiento y llevar todas las ejecuciones a correr sobre un sandbox con el fin de prevenir accesos innecesarios a otras partes del sistema operativo.

CAPÍTULO 4

LA SEGURIDAD EN LOS SISTEMAS DE JUZGAMIENTO

4.1 INTRODUCCIÓN

En este capítulo se evaluará la manera en cómo se enfoca la seguridad en los diferentes sistemas de juzgamiento. Es así como en la primera sección, se estudiará con cierto detalle cómo se aplica un modelo de seguridad a un sistema de juzgamiento con el uso de módulos de seguridad de Linux. Por consiguiente se da a conocer en la sección 4.3 la estructura general del juez Moe, juez que se caracteriza por estar construido a través de módulos que buscan la estandarización de un sistema muy general para desarrollar competencias de programación, luego se hace un análisis del funcionamiento del juez Boca.

4.2 MODELO DE SEGURIDAD ENFOCADO CON MÓDULOS DE SEGURIDAD DE LINUX

En esta sección se exponen las ideas del trabajo de Bruce Merry, [60] trabajo que describe el modelo de seguridad con el enfoque implementado para la competencia anual de programación en Sudáfrica, [61] modelado sobre la olimpiada internacional de informática (IOI). La idea se centra en la creación de un ambiente restringido para el sistema de juzgamiento a través de técnicas sandbox, planteado a través de diferentes enfoques que permiten la mejora de la seguridad en un sistema de juzgamiento para maratones de programación. Complementando sus ideas en otro de los trabajos del mismo autor, [62] en donde compara de una manera más profunda dos de los enfoques para un adecuado modelo de seguridad en un sistema de juzgamiento.

Para continuar con las ideas expuestas por este autor es importante aclarar uno de los conceptos muy utilizado en sus trabajos, es el término de sandbox, concepto que además es muy utilizado en el contexto del manejo de la seguridad de los sistemas de juzgamiento de maratones de programación, como se verá en secciones siguientes.

[60] Bruce Merry. "Using a Linux Security Module for Contest Security". Olympiads in Informatics, 2009, Vol. 3, 67–73

[61] Sitio oficial, [en línea], <http://www.olympiad.org.za/>.

[62] Bruce Merry. "Performance Analysis of Sandboxes for Reactive Tasks". Olympiads in Informatics, 2010, Vol. 4, 87–94

4.2.1 Sandbox

Cuando en el campo de la seguridad informática se habla de sandbox, se está refiriendo a un mecanismo de seguridad para la ejecución de programas de una manera segura y aislada de otros procesos. Esta técnica de seguridad es muy utilizada a la hora de ejecutar código nuevo o software de dudosa procedencia que proviene de terceros, es decir, puede llegar a ejecutar aplicaciones y programas con seguridad, con la capacidad de aislarlos dentro de una especie de contenedor virtual, desde el cual puede controlar los distintos recursos que solicita dicha aplicación (memoria, espacio en disco, privilegios necesarios, etc).

Este estricto control al que se somete el proceso sirve para discernir si el código a ejecutar es malicioso o no puesto que, por norma general, se restringirá cualquier tipo de acceso a dispositivos de entrada, de inspección del sistema anfitrión o la capacidad de acceso a redes. Entre los ejemplos, además de los sistemas de juzgamiento que se pueden encontrar en la utilización de este mecanismo de seguridad están las maquinas virtuales, applets,[63] HTML5, celdas,[64] etc.

4.2.2 Requerimientos planteados en el modelo de seguridad enfocado con Módulos de Seguridad de Linux

En la implementación del modelo de seguridad, se plantean una serie de requerimientos que se consideraron esenciales a la hora de diseñar dicho sistema. A continuación se muestra en la tabla 4.1, los requerimientos planteados para el diseño de un sistema de seguridad usando Módulos de Seguridad de Linux, permitiendo así, analizar que tipo de requerimientos se deben de tener en cuenta para la construcción de un sistema de seguridad que solucione el problema establecido como objeto de estudio en este proyecto (ver sección 1.1), esto se puede ver ya reflejado en la sección 5.

[63] Son programas que se ejecutan en contención dentro de una máquina virtual o un intérprete de scripts que haga el aislamiento

[64] Es un conjunto de límites en los recursos impuesto a los programas por el núcleo de un sistema operativo

Tabla 4.1: Requerimientos funcionales para un sistema de seguridad usando Módulos de Seguridad de Linux

Requerimientos	
Limitar los recursos	Se establece principalmente recursos como CPU y memoria con el fin de prevenir que códigos maliciosos bloqueen completamente el sistema.
Limitar de forma precisa las restricciones	Indica que es necesario que el sistema cumpla en forma cabal con las condiciones de una competencia, por ejemplo el *time limit* de un problema debe ser exacto de acuerdo a las condiciones del problema.
Ejecutar los códigos en un hilo de ejecución	Con el fin de evitar que los programas tomen ventaja de un sistema multinúcleo.
Restringir la creación de procesos.	Los códigos ejecutados no deben poder crear otros procesos dentro del sistema de juzgamiento.
Restringir la comunicación entre procesos	Se establece que el sistema no debe permitir que un programa pueda usar mecanismo de comunicación como por ejemplo el uso de Sockets TCP/IP.

Durante el análisis de los requerimientos el autor [65] fundamenta que el principal objetivo es la seguridad del sistema, no obstante a la hora del análisis de un determinado enfoque para llevar a cabo la seguridad, una solución potencial podría carecer de importancia por razones de usabilidad. En consecuencia se establecieron también una serie requerimientos no funcionales

Tabla 4.2: Requerimientos no funcionales para un sistema de seguridad usando Módulos de Seguridad de Linux

Requerimientos no Funcionales	
Tiempo de configuración mínimo	Debido a que no se posee un grupo de máquinas evaluando las soluciones de los participantes el rendimiento es de vital importancia
No afectar significativamente el rendimiento	Los tiempos de CPU de la máquina no se deben ver afectados por el sistema de seguridad
Permitir operaciones comunes de las librerías para los compiladores usados.	En este requerimiento se toma el cuidado de restringir librerías de I/O que usan llamados al sistema con el fin de no vulnerar otros requerimientos establecidos

Una vez establecidos los requerimientos, se evaluaron tres posibles enfoques de implementación de cómo llevar a cabo todas las restricciones planteadas, como el uso de intercepción de llamadas al sistema, la virtualización y el uso de módulos de seguridad de Linux, acto seguido se mostrará en que consisten estos enfoques.

[65] Bruce Merry. phD in computer science from the University of Cape Town and senior software engineer at ARM.

4.2.3 Intercepción de llamadas al sistema

En los sistemas operativos basados en Linux las aplicaciones no tienen acceso a dispositivos como puertos ethernet, unidades de disco u otros procesos, accediendo estrictamente a la memoria asignada para su ejecución. Así pues, si una aplicación necesita interactuar con otros dispositivos u otras aplicaciones es debido el uso de las llamadas al sistema. Las llamadas al sistema son similares a las funciones, con la diferencia de que esta vez el control de la misma la toma el kernel del sistema operativo.

Por lo tanto si se necesita la limitación de las llamadas al sistema de una aplicación es precisamente llegar a interceptarlas, Linux proporciona la llamada al sistema *ptrace*, tecnología que es introducida en el kernel de Linux ofreciendo la posibilidad de investigar procesos, recuperando información de los datos que usa, además de permitir notificar a un proceso de la llamada al sistema de otros procesos.[66] Ahora aplicando esto al sistema de seguridad se podría controlar las llamadas al sistema que no son seguras y así cumpliría las restricciones que se habían planteado.

Sin embargo, el uso de las intercepciones al sistema mediante *ptrace* en el trabajo de Merry,[67] se analiza una serie de desventajas para este sistema de seguridad. Una desventaja que presenta, es la sobrecarga al sistema de juzgamiento, debido a que cada llamada al sistema se genera un cambio de contexto adicional sobre el kernel hacia el proceso que restringiría las llamadas al sistema. Además esta técnica es propensa a agujeros en la seguridad, debido a que se puede presentar modificación por medio de otro hilo a los valores pasados al kernel, gracias al proceso que hace las intercepciones en el momento en que el interceptor hace sus comprobaciones y el tiempo en que el kernel los ve. Se deben de tener en cuenta otra limitante como el número de los llamados al sistema que en varios sistemas operativos basados en Linux llegan hasta los 300, y por lo tanto una misma operación se podría realizar mediante el uso de varias llamadas, complicando el nivel de abstracción y de control de estas llamadas.

4.2.4 Virtualización

En este enfoque se permite que un sistema operativo llamado cliente se ejecute dentro de otro llamado sistema anfitrión. Por lo tanto cuando un programa potencialmente malicioso llega al sistema de juzgamiento, éste se ejecuta en el sistema operativo cliente, permitiendo restringir los recursos del sistema anfitrión, proporcionando a su vez un alto nivel de seguridad debido a que los daños provocados se harían en el sistema virtualizado. Sin embargo la principal desventaja de este enfoque es que es necesario que el sistema operativo invitado se inicie cada vez que alguna ejecución lo requiera, añadiendo una sobrecarga significativa al proceso de la evaluación de las soluciones en el sistema de juzgamiento. Entonces según lo referenciado en la tabla 4.2, el uso de este enfoque de sandbox provocaría el incumplimiento de uno de los requerimientos previamente establecidos, lo que hace un enfoque no adecuado para las pretensiones anteriormente mencionadas.

[66] Espías de procesos, [recurso en línea], <http://www.Linux-magazine.es/issue/44/059-062PerlLM44.pdf>.

[67] Bruce Merry. "Using a Linux Security Module for Contest Security". Olympiads in Informatics, 2009, Vol. 3, 67–73

4.2.5 Módulos de seguridad de Linux

Una manera importante de mitigar las vulnerabilidades de software es a través del uso de los controles de acceso, no obstante muchos de estos mecanismos de control de acceso son inadecuados para proporcionar un fuerte sistema de seguridad. [68] Para mejorar el sistema de seguridad a través de los controles de acceso surgen una gran cantidad de métodos que son llevados en actualizaciones sobre el kernel de Linux, pero sin llegar a ser parte estándar del kernel. Este problema de estandarización se soluciona a través de los módulos de seguridad Linux, que proporcionan un framework de propósito general para el manejo de las políticas de seguridad que son establecidas directamente en el kernel. Cada vez que el núcleo está a punto de emprender alguna acción privilegiada en nombre del usuario, una alerta en el módulo de seguridad es notificado para determinar el acceso o no.

Implementación

Durante el análisis de estos enfoques para llevar las respectivas restricciones de seguridad se establece que la implementación mas apropiado para el sistema de juzgamiento, es el uso de módulos de seguridad de Linux. Rechazando el enfoque por medio de intercepción de llamados al sistema, precisamente por las desventajas mencionadas para esta solución, además de que resultaba muy tedioso la construcción de un nuevo sistema que interceptara las llamadas. También se rechaza la opción de ejecutar las soluciones de los problemas mediante una máquina virtual, precisamente a que las pretensiones del trabajo expuesto incumplían requerimientos no funcionales al no lograr una configuración ligera.

Ahora se hará una breve descripción sobre los pasos implementados para montar el módulo de seguridad de Linux y configurar el ambiente restringido; para esto se vale inicialmente de un proceso que se encarga de hacer una llamada al sistema por medio del comando *exec*[69] entre otros que se muestran en la figura 4.1 encargados de ejecutar el programa que esta listo para el juzgamiento, trayendo a su vez todas las restricciones de seguridad para la ejecución de dicho programa [70].

Figura 4.1: Comandos para el módulo de seguridad implementado por *B. Marry*

Commands that can be issued to the security module

Syntax	Meaning
`version` *major*`.`*minor*	Mark this process as restricted, and check for version mismatches
`allow threads` *n*	Set the number of threads the process may have at any time
`allow exec` *n*	Allow *n* calls to `exec`
`allow write`	Remove any extra restrictions on filesystem access
`allow write` *file*	Allow creation and write access to *file*

Restricción de Recursos, es importante la configuración de un nuevo usuario en Linux para

[68] James Morris, Chris Wright, Stephen Smalley "Linux Security Modules: General Security Support for the Linux Kernel.August 17, 2002

[69] EXEC, [en línea], <http://Linux.die.net/man/3/exec>.

[70] Es importante tener en cuenta que toda implementación en el trabajo de Bruce Merry, fue desarrollada sobre Linux 2.6.24

correr el sistema de juzgamiento cada vez que se requiera juzgar un programa, limitando recursos como el tiempo de CPU y memoria, lográndolo a través de la llamada al sistema con el comando *setrlimit*.[71] Por lo tanto se usa un programa wrapper[72] que aprovecha la comunicación entre procesos que provee el módulo de seguridad de Linux a través del canal de comunicación *(/proc/self/attr/exec)*[73] para configurar la limitación de los recursos y el módulo de seguridad, ejecutando finalmente el código para ser juzgado, esperando hasta determinar los resultados del tiempo de compilación y ejecución.

Comunicación entre procesos, otro problema que se encarga el módulo de seguridad de Linux es la comunicación entre procesos, bloqueando medios como sockets y archivos contenidos en un área común, como el directorio */proc*[74] donde se evidencia que el intento de restringir dicho directorio le proporciona inestabilidad a el módulo de seguridad. Sin embargo estos problemas son menores debido a que en el sistema de juzgamiento las ejecuciones llegan serialmente quedando encoladas para su respectiva ejecución. No obstante existen riesgos de dejar archivos en el sistema de archivos, y permitir a un programa malicioso tomar ventaja de ello caso que se evidenció en [75] donde se toma las precauciones restringiendo procesos de escritura para una lista de archivos y que además son vueltos a su normalidad después de cada ejecución.

Intercambio de directorios principales, en sistemas operativos tipo UNIX es posible correr procesos donde el directorio que se supone es el principal, es tan solo un subdirectorio del verdadero directorio principal, haciendo que los procesos ejecutados allí sólo sean capaz de manipular archivos que sólo se encuentren en dicho subdirectorio. Sin embargo en el desarrollo de este sistema de seguridad, se toma como una medida que no es tan necesaria debido a que su aporte en seguridad no era muy significante.

Máquina Virtual de Java, en el contexto de las maratones de programación es muy común encontrar lenguajes para resolver los problemas que son predilectos entre los participantes como C/C++ y Java,[76] siendo casi imprescindible dar soporte a estos tres lenguajes en cualquier sistema de juzgamiento. No obstante en el enfoque que se le da, este sistema de seguridad se encontró que no era muy práctico permitir las llamadas al sistema que hacia la máquina virtual de java (JVM) dejando agujeros en la seguridad del juez para ejecuciones programadas en java mientras que todo era seguro para los programas ejecutados en C y C++. Por lo tanto es necesario tomar otras medidas de seguridad en cuanto a los códigos ejecutados en la máquina virtual de java. Estas medidas adicionales son a través del gestor de seguridad de java[77] y el comando de consola *Xmx* para limitar el máximo tamaño del heap[78] de java.

[71] Documentation oficial, [en línea], <http://Linux.die.net/man/2/setrlimit>.

[72] Es un programa que controla el acceso a un segundo programa, con el fin de cubrir la identidad del segundo programa, obteniendo un nivel mas alto de seguridad. Son muy usados dentro de la seguridad de sistemas tipo UNIX.

[73] Process Attribute API for Security Modules, [en línea], <http://lwn.net/Articles/28222/>.

[74] Existe mucha información disponible en este directorio que puede usar un proceso con propósitos maliciosos. [en línea], <http://www.Linuxtotal.com.mx/index.php?cont_info_admon_016>.

[75] Gurrero Yisel. Gonzales Leandro "*Concepción y desarrollo de un módulo para la detección de plagio en el juez en línea caribeño*",Serie Científica de la Universidad de las Ciencias Informáticas, No. 9, Vol. 5, Año: 2012

[76] En la final mundial de maratones de programación solo se permite programas en estos tres lenguajes. Dato extraído de <http://icpc.baylor.edu/worldfinals/programming-environment>, [consultado el 4 de julio de 2013].

[77] Java Security Manager, [en línea], <http://docs.oracle.com/javase/tutorial/essential/environment/security.html>.

[78] Es el área de memoria usada para la asignación dinámica de memoria.

El gestor de seguridad de java trae un archivo de configuración describiendo los privilegios y acciones que pueden realizar las aplicaciones desarrolladas en java. Pero su Configuración por defecto se encuentra permisiva al permitir operaciones que no son permitidas en una competencia de programación como el permitir la escritura de cualquier archivo. Entonces para solucionar dicho problema se usa el comando *Djava.security.policy*[79] que permite una definición específica de las propias políticas de seguridad que llevará la Máquina Virtual de Java con el fin de sólo permitir una lista de los permisos que tienen los programas desarrollados en java.

4.3 THE MOE CONTEST ENVIRONMENT

El sistema de juzgamiento Moe se caracteriza por estar construido de forma modular permitiendo facilitar su adaptación a las características específicas de una maratón de programación en particular, permitiendo una fácil configuración en el tipo de problemas u otros lenguajes de programación.

A continuación se hará una breve descripción sobre el trabajo desarrollado principalmente por Mareš,[80] donde se describe brevemente cuales son los módulos implementados en Moe. Cabe resaltar que el diseño de este juez fue pensado en la flexibilidad y en la estandarización, características que no poseían los sistemas de juzgamiento que fueron construidos durante las dos últimas décadas, ya que se concentraban en las necesidades específicas y dando lugar a la reimplementación de las mismas cosas una y otra vez cayendo también en errores comunes que ya habían sido resueltos por alguien mas.[81] Por lo tanto el analizar la implementación de este juez dará una idea muy acertada de la estructura de un sistema de juzgamiento para competencias de programación.

4.3.1 Módulos implementados en Moe Contest Environment

Este sistema de juzgamiento nace inicialmente con el propósito de atender las olimpiadas nacionales de programación en República Checa bajo el nombre de Mo-Eval atendiendo únicamente a las necesidades de esta competencia, pero que posteriormente fue reestructurado y consecuentemente generalizado. Su estructura principal esta encaminada a sistemas operativos tipo Linux y sistemas operativos bajo el estándar POSIX.[82] No obstante la estandarización se pierde un poco en el módulo que implementa el sandbox porque depende del sistema operativo y de la arquitectura de la máquina.[83] Por otro lado Moe es de código abierto con toda su documentación disponible bajo los términos de la licencia GNU.[84]

Se continuará por hacer una descripción de cada módulo que posee Moe, interconectados entre si como se observa en la imagen 4.2.

[79] [En línea], <http://pic.dhe.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=2Fcom.ibm>.

[80] M. Mareš. Professor at the Department of Applied Mathematics of Faculty of Mathematics and Physics of the Charles University. Prague

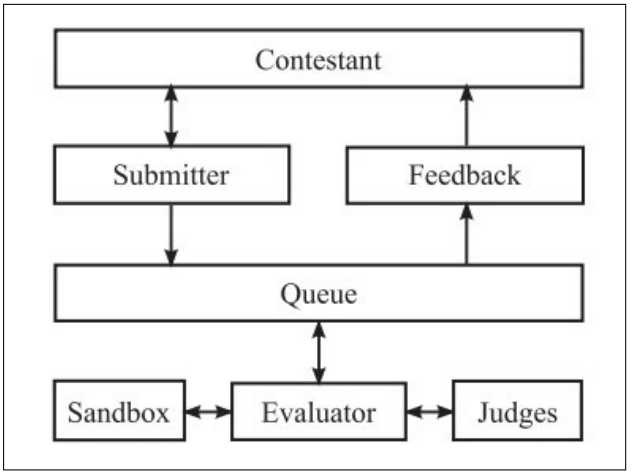
[81] M. Mareš Moe.Design of a Modular Grading System. Olympiads in Informatics, 2009, Vol. 3, 60–66

[82] Son estándares de llamadas al sistema operativo definidos y especificados formalmente en el IEEE 1003. Su propósito es generalizar las interfaces de los sistemas operativos para que una misma aplicación pueda ejecutarse en distintas plataformas, [en línea], <http://www.unix.org/version3/ieee_std.html>.

[83] Este sandbox esta implementado para Linux en una arquitectura i386

[84] General Public License versión 2, [información en línea], <http://www.gnu.org/copyleft/gpl.html>

Figura 4.2: Interconexión de los módulos de Moe



- *Sandbox*: Este módulo se encarga de la seguridad del juez, por medio del control de las ejecuciones de las soluciones de los participantes de la competencia a través de un ambiente seguro, limitando el tiempo de ejecución, el consumo de memoria y las llamadas al sistema.

- *Judges*: Se refiere a un conjunto de herramientas usados para comparar las soluciones de los participantes con la solución verdadera a través de los casos de prueba contenidos en el sistema de juzgamiento dado un determinado nivel de dificultad en el problema. Este módulo se basa en una biblioteca de funciones para el análisis rápido y riguroso de archivos de texto.

- *Evaluator*: Se encarga principalmente del proceso de la calificación de los problemas, es decir, tiene la capacidad de dar un veredicto a las diferentes soluciones que los participantes envían para ser juzgados en el juez. Esto lo logra a través de llamadas a los diferentes compiladores, el uso del sandbox y las herramientas del judge.

- *Queue manager*: Este módulo fue pensado por la necesidad que representan las competencias grandes que inscriben a una gran cantidad de participantes, a lo que es necesario realizar el juzgamiento en paralelo, por lo tanto este gestor de colas, se encarga de llevar el debido orden de las soluciones que envían los participantes y repartir cada solución entre los diferentes evaluadores que corren en diferentes maquinas

- *Submitter*: Es común ver que la mayoría de los sistemas de juzgamiento manejan una interfaz web para la presentación de las soluciones entre los participantes y el propio sistema de juzgamiento. Pero aunque Moe no maneja una interfaz web este módulo puede ser usado como una interfaz limpia entre los módulos web y el resto del sistema.

- *Test suit*: Una de las principales preocupaciones de Moe es la exactitud y la seguridad con el fin de tener éxito en cualquier competencia. Con este fin construye un conjunto de pruebas que trata de abarcar todos los posibles casos y escenarios de ataques que atentan contra la integridad del sistema de seguridad. Dentro de estas pruebas

se consideran pruebas unitarias para diversas funciones, pruebas de regresión para errores históricos y pruebas de seguridad de los posibles ataques que se enunciaron en la sección 3.4.

- *Feedback*: Procesa los diferentes resultados obtenidos por los participantes a través de la generación de varios reportes como tablas de clasificación y reportes sobre la retroalimentación de los participantes.
- *Scoring*: La forma de clasificar los participantes esta pensada en la implementación de varios módulos pequeños que usan diferentes estrategias de puntuación, como permitir tiempos de ejecución graduales, dando puntuación dependiendo de que tan cerca estuvo el tiempo de ejecución del programa al limite permitido y dependiendo de que aproximación tuvo una determinada solución con la solución real del problema.

4.4 SISTEMA DE JUZGAMIENTO BOCA ONLINE CONTEST ADMINISTRATOR

En esta sección se pretenden mostrar los aspectos de seguridad del juez Boca,[85] para ello es necesario conocer los módulos o las partes más importantes del sistema y cuál es el trabajo que desarrolla cada parte. Es importante mencionar que fue posible estudiar el funcionamiento de éste sistema gracias a que el código fuente está disponible y fácilmente puede ser descargado y usado, bajo las condiciones de la licencia GNU General Public License 3 [86] a partir de la versión 1.5.

Se han identificado los siguientes elementos: Sistema operativo GNU/Linux, Entorno Chroot, Safeexec, scripts para la compilación y ejecución, y para comparar la salida del programa con la salida esperada.

4.4.1 Sistema Operativo GNU/Linux

Boca fue diseñado para ejecutarse en un sistema operativo basado en Linux. Los sistemas operativos basados en Linux son sistemas bastante robustos, que incorporan un amplio conjunto de características de seguridad comunes a todos los sistemas operativos tipo Unix; dichos sistemas ofrecen una gran flexibilidad y capacidad de ser adaptados según las necesidades propias. Otras características importantes son el manejo de grupos y usuarios, para los cuales se pueden establecer permisos que facilitan la administración de los recursos, para ello pueden ser establecidas cuotas y límites, además los ficheros tienen atributos de control de acceso tales como: permisos de lectura, escritura y ejecución.

[85] Sitio oficial, [en línea], <http://www.ime.usp.br/ cassio/boca/>.
[86] Términos de la licencia disponible en <http://www.gnu.org/licenses/gpl.html>.

4.4.2 Entorno Chroot

Cuando se habla de *chroot* se puede hacer referencia a dos conceptos, uno es la llamada al sistema *chroot* y el otro es el Entorno *chroot* o Jaula *chroot*.

El entorno *chroot* es el directorio como tal que va a ser la nueva raíz para los procesos enjaulados, una vez un proceso es enjaulado dicho proceso no tendrá acceso al sistema raíz real, tampoco serán accesibles los programas, variables de entorno, etc. Es por esta razón que se opta por que el nuevo sistema raíz sea un sistema completo, es decir, que sea un sistema autónomo y que allí se puedan hacer tareas como las que se hacen en el sistema real, tales como: ejecutar programas, crear usuarios, programar tareas, etc. Esta independencia se puede lograr instalando un sistema base, con su propio árbol de directorios; una herramienta que permite hacer esto es *debootstrap*[87] utilidad con la cual es posible instalar un sistema base de Debian.[88] En ocasiones se requiere que desde el sistema enjaulado se pueda acceder a recursos del sistema real, tales como por ejemplo los dispositivos del equipo u otros recursos, lo que también es posible de hacer. Un entorno *chroot* provee de un gran número de opciones que se pueden configurar según sean las necesidades.

La llamada al sistema *chroot*[89] lo que hace es cambiar para el proceso principal y sus hijos el directorio raíz, es decir, que para esos procesos no será visible el directorio raíz real del sistema operativo, sino que el nuevo directorio raíz será el que se indique a través del comando *chroot*. Esto implica que no se podrá acceder desde un entorno enjaulado a otros directorios que se encuentren por encima de éste, únicamente a los directorios que se encuentren en niveles inferiores del árbol de directorios. Para muchos sistemas solamente el root puede hacer esta llamada.

4.4.3 Safeexec

Safeexec es un programa escrito en lenguaje C cuyo propósito es permitir la ejecución de un programa en un ambiente protegido o controlado. Para lograr ese control es necesario restringir los recursos usados por dicho programa, Safeexec limita los recursos a nivel del kernel, mediante las llamadas al sistema[90] *getrlimit()* y *setrlimit()* para obtener y establecer el límite de recursos respectivamente. Cada recurso tiene asociado un límite *soft* y un límite *hard*. El límite *soft* es el valor que el núcleo le aplicará y el *hard* es el tope del límite *soft*, es decir, que un proceso podrá acceder a un recurso desde el valor 0 hasta el límite *hard*. A continuación se muestra la cabecera de estas funciones y algunos de los parámetros que reciben:

```
#include <sys/time.h>
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlim);
```

[87] Información disponible en <http://packages.debian.org/sid/debootstrap>.

[88] Página oficial de Debian, <http://www.debian.org>.

[89] Mas documentación disponible sobre *chroot* en <http://www.gnu.org/software/coreutils/manual/coreutils.html#chroot-invocation>.

[90] Mas información sobre los comandos que restringen el sistema en <http://www.gnu.org/software/libc/manual/html_mono/libc.html#Limits-on-Resources>.

```
int setrlimit(int resource, const struct rlimit *rlim);
```

El parámetro *resource* puede ser:

- RLIMIT_CPU: Límite de tiempo de CPU (segundos).
- RLIMIT_DATA: Máximo tamaño del segmento de datos del proceso.
- RLIMIT_STACK: Máximo tamaño stack del proceso (bytes).
- RLIMIT_CORE: Máximo tamaño del core file.
- RLIMIT_NPROC: Máximo número de procesos.
- Entre otros como: RLIMIT_RSS, RLIMIT_FSIZE, RLIMIT_NOFILE, etc.

También se puede conocer los recursos usados por un proceso con la función *getrusage*. [91]

4.4.4 Compilación y ejecución de un programa

Para cada lenguaje de programación permitido existe un archivo llamado <*lenguaje*>.run este archivo recibe los parámetros necesarios para cada juzgamiento, tales como el código fuente y los casos de prueba (input y output) y el timelimit; este archivo es el responsable de la compilación y la ejecución de cada código fuente. El primer paso es la compilación del código fuente, si la compilación es correcta se sigue con el siguiente paso que es la ejecución, sino es así se termina la ejecución con un mensaje de *compilation error*, todo esto se hace por fuera del entorno *chroot*. Para el paso de la ejecución se hace el llamado al comando Safeexec al cual se le deben pasar los argumentos necesarios como el timelimit, memory limit, el usuario y el grupo, el archivo de entrada (input) y el ejecutable creado en el paso de compilación. En el paso de ejecución se ingresa al entorno *chroot*. La salida de la ejecución del safeexec es un número que indica, si tuvo éxito o no la ejecución del programa. Por ejemplo *0* significa que la ejecución del programa terminó con éxito, *2* rumtime error, *3* timelimit exceeded, etc.

4.4.5 Validación de la respuesta a un problema

Durante el proceso de ejecución de cada programa, se genera un archivo *Output* que es la salida dada por un programa al pasarle un archivo de entrada, ese archivo es la solución a un problema en específico, lo que resta por hacer es validar si dicha respuesta es correcta o no, para ello existe para cada lenguaje un archivo llamado <*lenguaje*>.compare que a través de el comando *diff*[92] evaluará si existen o no diferencias entre la salida del programa *Output* y la *Salida esperada*, si no hay diferencias entonces la solución es correcta.

[91] Esta función se encuentra documentada en,
<http://www.gnu.org/software/libc/manual/html_mono/libc.html#Limits-on-Resources>.

[92] Una mayor referencia sobre este comando se encuentra disponible en,
<http://www.gnu.org/software/diffutils/manual/html_node/index.html#Top>.

CAPÍTULO 5

DEFININICIÓN DE REQUERIMIENTOS PARA EL SISTEMA DE SEGURIDAD

5.1 INTRODUCCIÓN

En el capítulo 4 se mostraron algunos enfoques existentes para la creación de un sistema de seguridad para un juez de maratones, y también se mostraron dos sistemas, The Moe Contest Environment y el Boca Online Contest Administrator; esto con la idea de tener una visión más amplia sobre los aspectos necesarios a tener en cuenta para la creación de un sistema de seguridad y algunas soluciones existentes. En este capítulo se muestra una descripción del proceso de desarrollo del sistema de seguridad para el juez de maratones UTP,[93] primero planteando una serie de requerimientos funcionales y no funcionales, para luego establecer las soluciones adoptadas para la construcción de éste sistema.

5.2 REQUERIMIENTOS

Se establecieron los siguientes requerimientos funcionales, el sistema:

1. El sistema debe restringir los recursos para los determinados problemas de una maraton de programación como la CPU y la memoria.
2. El sistema debe ejecutar los programas de manera secuencial dejando en cola las peticiones entrantes con el propósito de tener un solo hijo de ejecución.
3. El sistema debe permitir el juzgamiento de soluciones escritas en los diferentes lenguajes que permite una maraton tipo ACM-ICPC.

Se establecieron los siguientes requerimientos no funcionales, el sistema:

1. Debe correr bajo un sistema operativo Gnu/Linux
2. Debe ser independiente y modular, de fácil integración con otros sistemas
3. Debe ser fácilmente configurable

[93] El juez para maratones de la UTP, Utpjudge es una iniciativa de crear un juez de maratones para la Universidad Tecnológica de Pereira, desarrollada por el semillero In-sílico

4. Debe permitir las operaciones comunes entre librerías

5. Debe tener un rendimiento óptimo

Una vez establecidos los requerimientos del sistema, el siguiente paso fue la implementación de una solución a esas necesidades. A continuación se muestra para cada requerimiento funcional, la solución que se adoptó:

1. Para restringir los recursos se optó por usar el programa safeexec como sandbox para permitir la ejecución en un ambiente restringido a cada uno de los juzgamientos, en la sección 4.4.3 se explica el funcionamiento de este programa.

2. Se decidió ejecutar secuencialmente cada juzgamiento, ya que se desea que cada juzgamiento sea independiente, es decir, que el tiempo de CPU y la memoria disponible, sean lo más equitativos posible entre cada juzgamiento.

3. Como parte del protocolo que se debe seguir para que el sistema pueda hacer juzgamientos, se estableció que se debe especificar una línea de compilación y una línea de ejecución. Esto se hizo para permitir el juzgamiento en diferentes lenguajes de programación, basta con especificar para cada lenguaje una plantilla de cómo se compila, si es un lenguaje compilado y cómo se ejecuta.

Para los requerimientos no funcionales:

1. Como se explicó en la sección 4.4.1 los sistemas tipo Unix ofrecen una serie de ventajas, entre las cuales están la configuración de usuarios y permisos, debido a estas características se decidió usar el sistema operativo Debian Gnu/Linux. Se creó un usuario limitado para el juzgamiento de cada solución, además se configuraron los permisos de lectura, escritura y ejecución, de acuerdo al análisis hecho, de los directorios y archivos importantes del sistema y de los usuarios que podían acceder o no a ese material.

2. Para la creación del sistema se tuvo en cuenta que debería ser un sistema modular y de fácil integración con otros sistemas, para ello se estableció un protocolo que consta de unos parámetros de entrada, necesarios para cada juzgamiento y la salida del sistema, la cual es el veredicto. De esta manera para que el sistema se integre con otros sistemas solo es necesario seguir el protocolo de los parámetros de entrada y el sistema devolverá una salida en un formato fijado. El sistema es completamente modular y cada módulo se encarga de una tarea muy específica.

3. La configuración e instalación del sistema son bastante sencillas para ello se desarrolló un script de configuración, el cual se encarga de instalar los paquetes necesarios que utiliza el sistema, y la creación y configuración de los permisos para archivos y carpetas del sistema.

4. Inicialmente se había establecido que se usaría un entorno Chroot dentro del cual se harían todos los juzgamientos, pero al final se decidió no usarlo. Algunas de las razones por las cuales no se usó son: le agrega complejidad a la instalación y configuración del sistema, ya que se debe configurar un ambiente que permite la ejecución

sin problemas de los compiladores y debe permitir la interacción con diferentes librerías, por otro lado agrega un costo en el rendimiento del sistema, pues en cada llamado Chroot, el sistema operativo debe hacer un cambio de contexto, no es un mecanismo completamente seguro y es posible saltarse las restricciones de estar enjaulado. Al no hacer uso del entorno chroot las operaciones con librerías y la llamada a los compiladores se hace de manera transparente desde el sistema real.

5. Las operaciones son bastante rápidas, el sistema atiende uno a uno cada juzgamiento, lo procesa y da un veredicto en un tiempo razonable.

CAPÍTULO 6

DESARROLLO Y PRUEBAS DE LA APLICACIÓN

6.1 ANÁLISIS

Para el establecimiento de los requisitos del sistema se hizo un análisis sobre las funcionalidades que debería tener el sistema de seguridad, luego de ese análisis que corresponde al levantamiento de requerimientos, se encontró que el sistema debe permitir cargar un problema y debe permitir juzgar un problema.

Luego del levantamiento y de la definición de los requerimientos se ha determinado que es posible el modelamiento del sistema mediante dos casos de uso, ya que desde el punto de vista del actor esas son las funciones que puede desempeñar con el sistema. El diagrama se muestra la figura 6.1. Se ha identificado únicamente un actor, el cual es el script *testerbot*, ya que es este quien interactúa con el sistema.

Los dos casos de uso identificados son:

Cargar problema: Para que el sistema pueda realizar un juzgamiento a un problema en particular necesita dos cosas, los casos de prueba y los límites para ese problema, por lo tanto permitir cargar un problema debe ser una de las funcionalidades del sistema.

Juzgar problema: Luego de que se tienen los datos necesarios para un juzgamiento, es decir, luego de realizado el caso de uso "Cargar problema", el sistema puede llevar a cabo el juzgamiento de ese problema, que corresponde a compilar, ejecutar y verificar si la solución es correcta o no.

Es importante resaltar que durante el desarrollo de la aplicación no se siguió rigurosamente alguna metodología en específico, sin embargo para el desarrollo del mismo se propuso un desarrollo basado en iteraciones, en las cuales se fueron cubriendo cada uno de los requerimientos, verificando durante cada iteración si se cumplían los objetivos planteados inicialmente o si por el contrario era necesario reconsiderar la orientación del desarrollo. Una vez alcanzada cierta madurez se proseguía con otro requerimiento, de esa manera y de forma incremental se fue desarrollando completamente el sistema.

6.2 DOCUMENTACIÓN DE LOS CASOS DE USO

En las tablas 6.1 y 6.2 se muestra la documentación de los casos de uso del sistema.

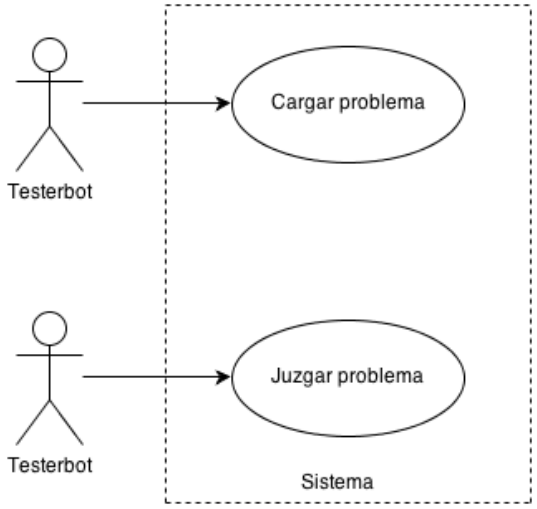
Tabla 6.1: Caso de uso 1: Cargar problema

Caso de uso	Cargar problema
Actor : Testerbot	**Referencia : 1**
Objetivo : Permitir en el sistema de juzgamiento poder cargar un determinado problema.	
Resumen : Este caso de uso es iniciado por el script de envíos automáticos de pruebas *Testerbot*, el cual le permite al sistema poder cargar un problema ubicado en un determinado directorio con el propósito de ser debidamente juzgado.	
Precondición : No tiene ninguna precondición.	**Postcondición :** El sistema puede realizar el juzgamiento del problema que ha sido cargado.
Acción de los actores :	**Respuesta del sistema :**
1. El script testerbot le indica al sistema la ruta de la carpeta con los casos de prueba y los límites del problema.	**2.** El sistema guarda la ruta de los datos y genera un identificador único para cada problema.
3. El script testerbot recibe el identificador único del problema.	**4.** El sistema queda libre para cargar o juzgar problemas.

Tabla 6.2: Caso de uso 2: Juzgar problema

Caso de uso	Juzgar Problema
Actor : Testerbot	**Referencia : 2**
Objetivo : Permitir en el sistema de juzgamiento poder juzgar un determinado problema.	
Resumen : Este caso de uso es iniciado por el script de envíos automáticos de pruebas *Testerbot*, el cual se encarga de pasarle al sistema de juzgamiento los datos necesarios para que el sistema pueda juzgar debidamente un problema, aquí el sistema se encarga de determinar el lenguaje de compilación y la restricción de los recursos que son requerimientos funcionales del sistema.	
Precondición : Caso de uso 1.	**Postcondición :** El problema es juzgado.
Acción de los actores	**Respuesta del sistema**
1. El script testerbot le indica al sistema el id (identificador único) del problema a juzgar, junto con el código fuente.	**2.** El sistema lee los datos necesarios para el juzgamiento de ese problema en particular determinando su lenguaje de compilación.
3. El script testerbot recibe un número que identifica a cada submission.	**4.** El sistema juzga el problema.
5. El scritp testerbot recibe el veredicto del problema juzgado.	**6.** El sistema queda libre para cargar o juzgar problemas.

Figura 6.1: Casos de uso



En el capítulo 5 se establecen los requerimientos funcionales del sistema de seguridad, dichos requerimientos corresponden a operaciones que el sistema debe realizar, para describir esas operaciones de manera más detallada, se hizo un modelamiento mediante contratos, pues los casos de uso aunque son el principal mecanismo para describir el comportamiento del sistema, en este caso no son suficientes. En las tablas 6.3, 6.4 y 6.5 se establecen los contratos de las operaciones del sistema.

Tabla 6.3: Contrato 1: Compilation

Contrato	Compilation
Operación	compilation(id,src)
Referencias cruzadas	Caso de uso: Juzgar problema
Responsabilidades	El sistema compila el código fuente y genera un ejecutable.
Notas	En este paso se escoge el compilador de acuerdo al lenguaje, en caso de que sea un lenguaje interpretado se omite la compilación, este proceso se hace de manera secuencial.
Precondición	Debe existir un código fuente con id del juzgamiento actual.
Postcondición	Se asocia el ejecutable con el id del juzgamiento actual.

Tabla 6.4: Contrato 2: Execution

Contrato	Execution
Operación	execution(id,in,limits,exec)
Referencias cruzadas	Caso de uso: Juzgar problema
Notas	Durante el proceso de ejecución se limitan los recursos de CPU y memoria, y de acuerdo al lenguaje se usa una línea diferente de ejecución, este proceso de hace de manera secuencial.
Responsabilidades	El sistema debe aplicar las restricciones y límites durante la ejecución y generar un archivo de salida.
Precondición	Debe existir un ejecutable con id del juzgamiento actual.
Postcondición	Se asocia la salida con el id del juzgamiento actual.

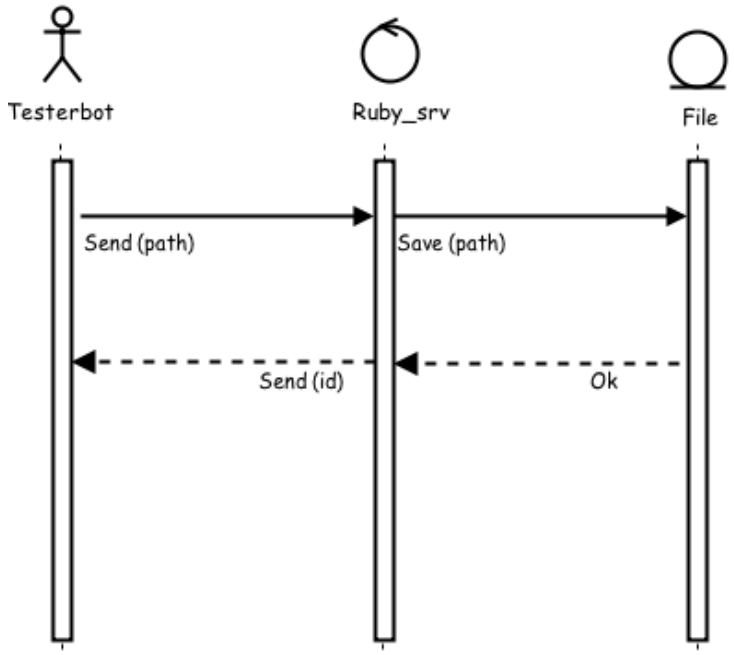
Tabla 6.5: Contrato 3: Judge

Contrato	Judge
Operación	judge(in,out)
Referencias cruzadas	Caso de uso: Juzgar problema
Responsabilidades	El sistema analiza si la respuesta dada por un programa es correcta y da un veredicto
Notas	Durante este proceso se chequea si existen diferencias entre la salida del programa con la salida esperada
Precondición	Estado del juzgamiento actual debe ser "Judging".
Postcondición	El estado del juzgamiento cambia de "Judging" al nuevo estado que corresponde al veredicto.

6.3 DISEÑO

El diseño del sistema se realizó con los diagramas de secuencia y de clases pertenecientes al estándar UML (Lenguaje Unificado de Modelado). En la figura 6.2 se muestra el diagrama de secuencia para el caso de uso *Cargar problema*. El script Testerbot envía la ruta en donde se encuentra el problema a cargar al script Ruby_srv, este script quien almacena la ruta y genera un identificador que es devuelto al script Testerbot.

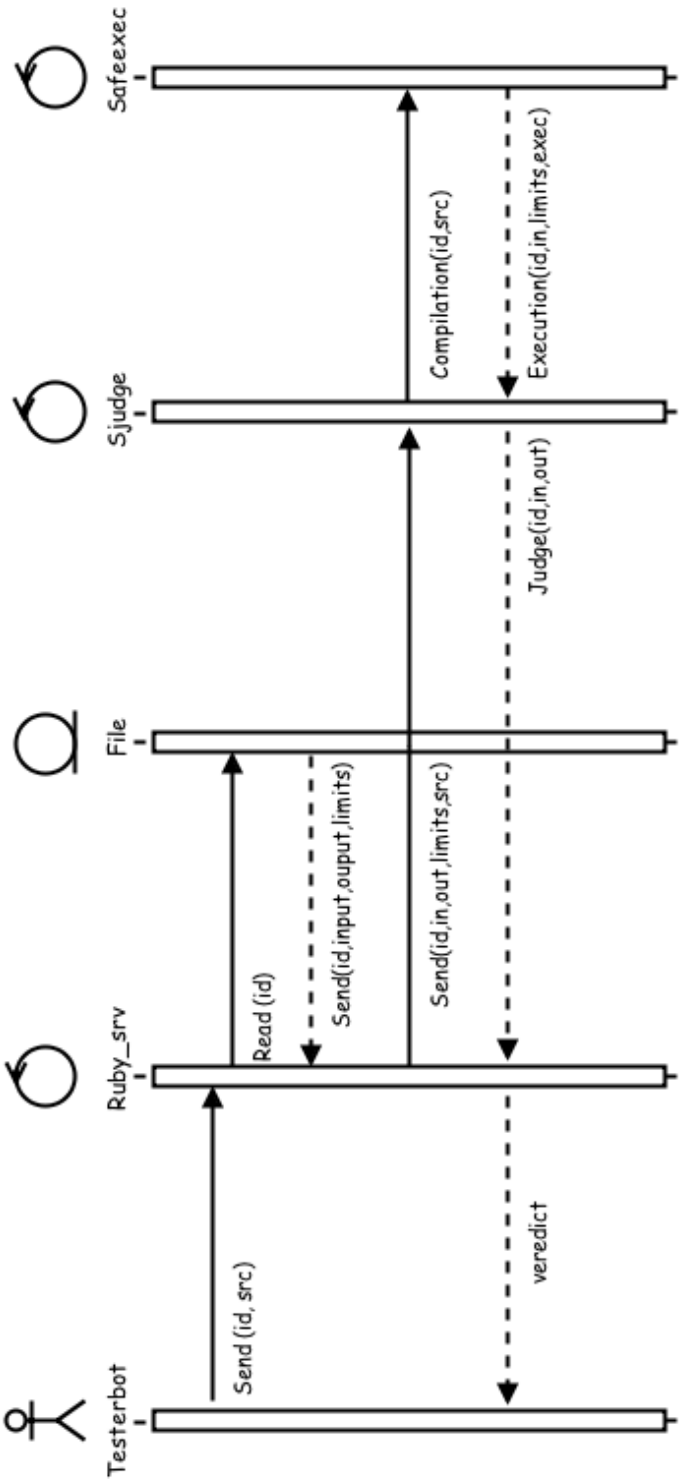
Figura 6.2: Diagrama de secuencia de caso de uso *Cargar problema*



Fuente: Autores

En la figura 6.3 se muestra el diagrama de secuencia para el caso de uso *Juzgar problema*. El scritp Testerbot envía un identificador junto con el código fuente, el scritp Ruby_srv almacena esos datos y hace una consulta de los limites y los casos de prueba, una vez se tiene todo lo necesario para un juzgamiento el script Ruby_srv envía todos los datos al script Sjudge, el cual luego de compilar (Si es necesario) el código fuente, y tener el ejecutable llama al programa Safeexec quien se encarga de la ejecución, el Safeexec devuelve la salida de programa ejecutado y el scritp Sjudge chequea si la salida correcta es igual a la salida generada por el programa y devuelve un veredicto.

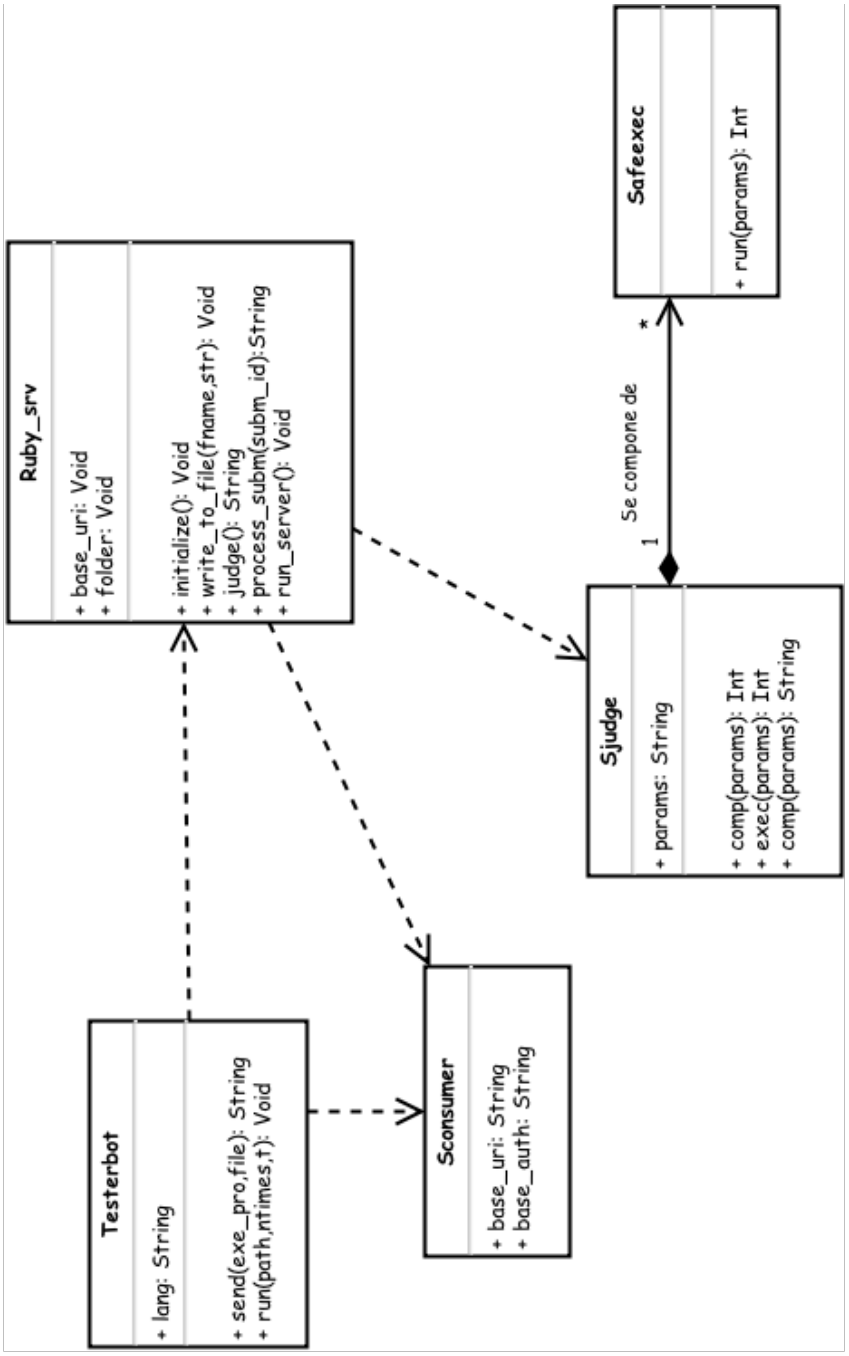
Figura 6.3: Diagrama de secuencia del caso de uso *Juzgar problema*



Fuente: Autores

En la figura 6.4 se encuentra el diagrama de clases. La clase *Testerbot* es la responsable de enviar y correr un conjunto de pruebas, para ello debe comunicar con la clase *Ruby_srv*, la clase *Ruby_srv* funciona como un servidor a la espera de que lleguen envíos para ser procesados, esta clase es responsable de obtener todos los datos necesarios para un juzgamiento, incluso debe escribir en el disco duro los casos de prueba para cada submission, para finalmente hacer el juzgamiento y devolver el correspondiente veredicto.

Figura 6.4: Diagrama de clases



Fuente: Autores

6.4 PRUEBAS REALIZADAS AL SOFTWARE

En esta sección no se pretende mostrar si los resultados sobre el aplicativo fueron los esperados o no, pues esto se encuentra explicado en el capítulo 10; lo que se quiere mostrar es la manera en que se llevaron a cabo las pruebas al software para verificar su correcto funcionamiento.

Las pruebas de software son uno de los pasos más importantes para desarrollar un software de mejor calidad, durante el proceso de pruebas es posible encontrar errores en el software y corregirlos. Generalmente cuando se hacen las pruebas se conoce cómo debería responder el sistema frente a cada una de las pruebas que se le realiza, es decir, que se hace un comparativo entre la respuesta esperada y la salida que arroja el sistema. De esta manera es posible corregir la mayoría de errores que son comunes en el proceso de desarrollo, y aunque no garantiza un software libre de errores permite desarrollar un producto de mejor calidad.

El aplicativo fue desarrollado con el propósito de soportar un conjunto de ataques específicos, por otra parte es fundamental que el sistema juzgue correctamente, estos dos aspectos fueron tenidos en cuenta para el desarrollo de las pruebas.

Para la realización de las pruebas se montó la Cuarta Maratón Interna UTP [94] en el sistema, pues para poder juzgar, es necesario cargar previamente los problemas con sus respectivos casos de prueba y límites.

6.4.1 Soporte a los ataques

Tras el diseño del conjunto de pruebas como se muestra en la sección 7 para cada una de las pruebas se conocía la salida esperada por parte del sistema, por ejemplo: para un código ataque cuyo propósito era consumir recursos durante la compilación, el sistema debería responder con el veredicto *Compilation Error*, así para cada uno de los ataques creados se definió el veredicto esperado.

A continuación se muestra el procedimiento que se siguió:

1. Se crearon un total de 20 códigos ataques, los cuales se colocaron dentro de una carpeta.
2. Al script para pruebas automáticas se le indicó la ruta de la carpeta con los ataques y se programó para que se enviara un código cada 5 segundos, esto con el propósito de tener tiempo suficiente para todo el proceso de envío y juzgamiento, y evitar que haya más de 1 juzgamiento pendiente.
3. Luego de ejecutar el script, se hizo la respectiva comparación entre los veredictos esperados y los veredictos arrojados por el sistema.

[94] Cuarta maratón interna de la Universidad Tecnológica de Pereira, los problemas fueron desarrollados por: Sebastián Gómez, Diego Agudelo, Santiago Gutierrez, Diego Martinez y Hugo Humberto Morales

6.4.2 Juzgamiento

Para verificar que los resultados esperados fueran los correctos, se decidió utilizar el juez Domjudge ampliamente usado y actualmente soportado por sus creadores y por la comunidad de usuarios que lo usan.

El procedimiento que se siguió fue el siguiente:

1. Se escogieron un total de 10 problemas de la Cuarta Maratón interna (UTP), para cada uno de los problemas se crearon 6 códigos con los veredictos, Accepted, Wrong Answer, Compilation Error, Time Limit, Memory Limit y Runtime Error.
2. Se hizo la instalación del Domjudge y luego se montaron los problemas de la Cuarta Maratón Interna UTP con los casos de prueba y límites oficiales.[95]
3. Se hizo el envío de cada uno de los códigos del conjunto de pruebas al juez Domjudge para establecer el veredicto correcto.
4. Luego usando el script para pruebas automáticas se repitió el paso anterior pero en el sistema de seguridad.
5. Finalmente se hizo el comparativo entre los veredictos arrojados por ambos sistemas.

Durante el proceso de pruebas, se encontraron algunos errores en los veredictos arrojados por el sistema, e incluso uno de los ataques logró bloquear el sistema; gracias a que se descubrieron estos errores fue posible identificarlos y corregirlos.

En esta sección se ha hecho énfasis en que para la realización de las pruebas al sistema se debe conocer un salida esperada, para luego compararla con la salida arrojada por el sistema; por ello es importante aclarar que para cierto tipo de juzgamiento esa salida esperada no siempre está definida de manera única, ya que es posible que un juzgamiento tenga varios posibles veredictos y eso depende de algunos factores como el compilador; por ejemplo: Lenguaje C no lanza una excepción de *división por cero* cuando esta división se hace entre números flotantes, al juzgar un código en el cual se haga una división por cero, se esperaría un error en tiempo de ejecución, pero para este caso en particular saldría otro veredicto diferente. Esto no necesariamente debe ser considerado como un problema para determinar si el sistema está funcionando de la manera que se espera, ya que para estos casos se puede establecer que se esperan varios posibles veredictos que podrían cambiar de acuerdo a diferentes factores.

[95] Los casos de prueba fueron los mismos que se usaron en la competencia, creados por los organizadores

CAPÍTULO 7

DISEÑO DEL CONJUNTO DE PRUEBAS Y MEDICIONES

7.1 DISEÑO DE PRUEBAS

En la población (ver sección 2.1.3) se especifican los distintos tipos de ataques que van a ser tenidos en cuenta para las pruebas y se encuentran clasificados de acuerdo al momento en que ocurren. Se tienen básicamente cuatro grupos distintos de ataques especificados como: *consumo de tiempo y recursos durante la compilación*, *acceso a material restringido*, *daños en el entorno de pruebas* y *saltarse la medición del tiempo*; para cada uno de estos items se crearon entre 1 y 6 códigos ataques escritos en lenguaje C++.

A continuación se muestra como a través del uso de directivas de preprocesador, mediante algunas llamadas al sistema y el uso de comandos del sistema es posible crear un código ataque, que puede afectar de diversas maneras un sistema de juzgamiento.

7.1.1 El preprocesador de C

Antes de que se lleve a cabo la compilación real de un programa, el preprocesador analiza el fichero fuente, en donde sustituye macros y procesa las directivas de preprocesador. Una directiva de preprocesador es una línea que comience con el carácter #, tales como: *#include*, *#define*, *#undef*, *#ifdef*, *#ifndef*, entre otras.

Se podría crear un código fuente que contenga ciertas directivas que aumenten considerablemente el tiempo de compilación, por ejemplo se podría crear un archivo muy grande en donde el preprocesador deba evaluar y sustituir una gran cantidad de macros, o también es posible crear un código que genere una recursión infinita como se muestra a continuación:

#include __FILE__

El macro predefinido *__FILE__* es reemplazado por el nombre del archivo actual y con la directiva *#include* la cual incluye un archivo, lo que produce una recursión infinita al incluirse así mismo. En el momento en que el código es compilado muestra el mensaje de error *error: #include nested too deeply*, pero no termina. De esta manera es posible bloquear el compilador, por lo que es necesario establecer un tiempo límite a la compilación, para ello inicialmente se había decidido utilizar *ulimit*,[96] llamada con la cual se puede obtener o establecer límites a un usuario; pero en la práctica este comando no funcionó como

[96]Documentación oficial sobre el comando *ulimit* en, <http://www.ss64.com/bash/ulimit.html>

se necesitaba, es decir, que limitara el tiempo de compilación y en la documentación se advertía que la rutina ya era obsoleta. Por lo que se decidió utilizar *timelimit*,[97] de esta manera fue posible establecer un tiempo máximo de compilación y prevenir ataques de este tipo.

7.1.2 Llamadas al sistema fork() y clone()

Permitir la creación de procesos hijos tiene varias implicaciones de seguridad, por ejemplo se podría crear un ataque conocido como *Bomba fork* que es básicamente un ataque de denegación de servicio; el objetivo principal es saturar lo más rápido posible la lista de procesos mantenida por el sistema operativo, creando procesos lo más rápido posible, evitando con ello que nuevos procesos puedan iniciarse; con lo que se logra no solo saturar la lista de procesos, sino también el agotamiento de recursos como el procesador y la memoria. El funcionamiento del ataque aprovecha los llamados recursivos en los cuales se crean procesos hijos, que a su vez crean otros procesos hijos, hasta que el sistema es bloqueado, pues no hay manera de liberar los recursos ocupados. Para crear un código que genere este ataque basta con escribir un ciclo infinito que contenga dentro la llamada *fork()*. Otra posibilidad de un ataque con uno de estos llamados es saltarse la medición del tiempo, como se hace referencia en la sección 3.4, creando procesos hijos que se encarguen del procesamiento. Para evitar este tipo de ataques se restringió la creación de procesos con RLIMIT_NPROC 4.4.

7.1.3 Uso de comandos del sistema

Mediante el uso de la función *System()* es posible ejecutar comandos del sistema operativo. Los comandos *rm*, *cp* y *mv* pueden ser usados para eliminar, copiar, mover o renombrar archivos y carpetas respectivamente. Usando alguno de estos comandos se puede tratar de dañar el sistema o tratar de hacer trampa, al intentar acceder a los casos de prueba o a las soluciones enviadas por otros equipos. Estos son solo algunos comandos, pero existen una gran variedad de ellos. Para evitar este tipo de ataques, es necesario tener presente los directorios y archivos, que son importantes para el sistema de juzgamiento, y tomarse el tiempo de evaluar los permisos de acceso de dichos archivos, además se deben configurar los permisos estrictamente necesarios para el usuario con el cual se corre cada uno de los submissions.

Lo expuesto anteriormente fueron algunos de los temas que se tuvieron en cuenta durante el diseño del conjunto de códigos considerados Ataques, por último se muestran 2 ejemplos completos de códigos ataques.

[97] Documentación oficial sobre el comando *timelimit* en, <http://devel.ringlet.net/sysutils/timelimit/>

Tabla 7.1: Implementación código ataque: Forzar un alto consumo de tiempo y recursos durante la compilación

Forzar un alto consumo de tiempo y recursos durante la compilación	
Lenguaje	**Tipo de ataque**
C++	Tiempo de compilación
El siguiente programa genera una recursión infinita al incluirse así mismo indefinidamente, mediante la directiva __FILE__.	

```
#include<iostream>

using namespace std;

#include __FILE__

int main(){
    return 0;
}
```

Tabla 7.2: Implementación código ataque: Accediendo a material restringido

Accediendo a material restringido	
Lenguaje	**Tipo de ataque**
C++	Tiempo de ejecución
El siguiente código mediante el uso del comando *cp* copia la salida correcta en la salida del programa, como resultado se obtendría un *Accepted* (Asumiendo que los casos de prueba estén en el mismo directorio y con ese nombre)	

```
#include<iostream>
#include<stdlib.h>
using namespace std;

int main(){
    string line;
    while(cin >> line);
    system("cp correct.OUT Main.OUT");
    return 0;
}
```

7.2 MEDICIONES

Una vez creado el conjunto de pruebas, se procedió a realizar las mediciones. Lo que se desea conocer es la disponibilidad del sistema, para ello es necesario comprobar si después

de que el sistema es sometido a un conjunto de ataques, éste es capaz de seguir atendiendo peticiones.

7.2.1 Desarrollo de script para envíos automáticos

El script fue creado con el objetivo de hacer envíos automáticos (Submissions) simulando envíos hechos por una persona (Participante), además con el objetivo de recolectar de manera más precisa los datos de interés para la medición de las variables.

Para el proceso de medición se plantea hacer primero un conjunto de envíos considerados No ataques, medir los tiempos de respuesta del sistema para atender esos envíos, posteriormente hacer un conjunto de envíos considerados Ataques; para finalmente volver a hacer el conjunto de envíos iniciales. Esto con el objetivo de establecer si el conjunto de ataques afecta la disponibilidad del sistema de juzgamiento.

Con la creación de este script se desea eliminar la posibilidad de algún error por factores humanos en la medición. Se debe garantizar que se envíen todas las peticiones sin que falte alguna, y que se hagan los envíos cada cierto tiempo, ese tiempo debe ser preciso; si esto se hace manualmente sería bastante difícil poder controlar el intervalo de tiempo entre cada envío, además de ser una tarea bastante tediosa, teniendo en cuanta el número total de envíos necesarios para las mediciones.

```
#!/usr/local/bin/ruby

# Script para hacer envios automáticos
Class Testerbot
    @@lang = {'cpp'=>1, 'c'=>2, 'java'=>3, 'py'=>4}

    def send(exe_pro,file)
        ext = file.split('.')[1]
        ur  = "/submissions/testerbot.json"
        res = SConsumer.post(ur, :query => {
            :language_id => @@lang[ext],
            :exercise_problem_id => exe_pro,
            :submission => {:srcfile => File.new(file)}
        })
        return res[0]
    end

    def run(path,n,t)
        problems = Dir[ path + '/*' ].map { |a| File.basename(a) }
        path2 = path + '/'
        n.times do |x|
          for p in problems
            subs = Dir[ path2 + p + '/*' ].map { |a| File.basename(a) }
            path3 = path2 + p + '/'
            for f in subs
              exec_pro = f.split('.')[0].split('_')[1]
              req = send(exec_pro,path3+f)
```

```
            sleep t
          end
        end
      end
    end
end
```

La clase *Testerbot* tiene dos métodos *send()* y *run()*. El método send() es el encargado de hacer un POST a la aplicación en el cual se envían los datos necesarios para un submission válido, *languge_id*, *exercise_problem_id* y el código fuente *srcfile*. El método run() se encarga de iterar sobre cada uno de los códigos que se desean enviar y los envía uno a uno; éste recibe tres parámetros, el nombre del directorio donde están las pruebas *path*, un entero n que indica cuantas veces se quiere enviar cada submission, y otro entero t que indica el intervalo de tiempo entre cada submission.

7.2.2 Escenario de pruebas

Características de la máquina de pruebas:

- 4GB de RAM, procesador Intel(R) Core(TM)2 Duo CPU T5750 @ 2.00GHz, Sistema operativo GNU/Linux Debian 7.1 32 bits con kernel 3.10.5-1.

Las pruebas se realizaron de la siguiente manera:

1. Antes de iniciar con las pruebas se verificó que el sistema operativo estuviese corriendo únicamente el sistema de juzgamiento, junto con los procesos estándar.
2. Se procedió a hacer el conjunto de peticiones iniciales ejecutando el script *Testerbot*.
3. Después se hizo el conjunto de pruebas ataques al sistema de juzgamiento.
4. Finalmente se hizo el ultimo conjunto de peticiones.

CAPÍTULO 8

IMPLANTACIÓN Y PUESTA A PUNTO DEL SISTEMA EN UN SERVIDOR

8.1 INTRODUCCIÓN

El sistema de seguridad desarrollado en el presente trabajo hace parte del proyecto Utpjudge, mencionado en secciones anteriores y que surgió como una iniciativa por parte del semillero In-Sílico para crear un juez de maratones de programación para la UTP; El sistema completo, es decir, el juez de maratones de la UTP se compone básicamente de dos partes, una es la parte web que provee una interfaz con múltiples funcionalidades tanto para el administrador como para los participantes y la otra parte la compone el sistema de seguridad planteado en el actual trabajo. Es por esto que en este capítulo se va a exponer la implantación en un servidor del sistema completamente integrado y no como un módulo independiente, como se planteó durante el desarrollo del presente trabajo; pues para efectos prácticos se requiere de ambos sistemas.

8.2 REQUERIMIENTOS

8.2.1 Requerimientos de hardware

Para los requerimientos de hardware se debe tener en cuenta, que las características de la máquina en donde se van a juzgar las soluciones, tales como la memoria RAM y el procesador, influyen directamente en la ejecución de las soluciones por ese motivo se debe tener cuidado cuando se establecen el *Time Limit* y el *Memory Limit* para cada problema, es decir, se deben ajustar de manera precisa a las características de la máquina.

Las características de la máquina en donde se hizo la instalación son las siguientes:

- **RAM**: 4 GB
- **Procesador**: Intel Xeon de 4 núcleos
- **Disdo duro**: 150 GB

8.2.2 Requerimientos de software

Sistema operativo

Las pruebas de instalación se han hecho sobre Gnu/Linux Debian, pero en general no debería haber inconvenientes si se usa otra distribución de Gnu/Linux.

Paquetes necesarios

curl, git, sudo, g++, gcc, libstdc++6, sharutils, openjdk-6-jdk, openjdk-6-jre, openjdk-7-jdk, openjdk-7-jre, python, nodejs, timelimit.

Nota: Para que el juez soporte nuevos lenguajes se deben instalar los compiladores necesarios para cada lenguaje de programación.

8.3 PROCESO DE INSTALACIÓN

Los pasos de instalación que se muestran a continuación se realizaron en un sistema Gnu/-Linux Debian. El símbolo # antes de un comando indica que se debe ejecutar como usuario root y el símbolo **$** como un usuario normal.

1. *Instalación de paquetes*:
   **# apt-get install curl git sudo g++ gcc libstdc++6 sharutils openjdk-6-jdk **
   openjdk-6-jre openjdk-7-jdk openjdk-7-jre python nodejs timelimit

2. *Instalación Ruby y Rails*:
   $ \ curl -L https://get.rvm.io | bash -s stable −−ruby −−rails
   # echo "source /usr/local/rvm/scripts/rvm" >> /etc/bash.bashrc
   $ source /etc/bash.bashrc

3. *Descargar Utpjudge*:
   $ git clone https://github.com/in-silico/utpjudge.git

4. *Configuración base*:
   $ cd utpjudge/
   # ./config.sh
   # bundle install

5. *Capistrano*: Modificar el archivo *config/deploy.rb* si es necesario, por defecto la aplicación estará en */var/www/apps/YOUR-SITE*
   $ bundle exec cap deploy:setup
   $ bundle exec cap deploy:check
   $ bundle exec cap deploy
   $ bundle exec cap deploy:migrate
   $ bundle exec cap deploy:seed

6. *Instalación Apache:*
   # gem install passenger
   # passenger-install-apache2-module

7. *Configuración Apache:* Luego del paso anterior se obtendrá un mensaje como el siguiente

   LoadModule passenger_module /usr/lib/ruby/gems/1.8/gems/passenger-2.2.2/ext/apache2/mod_passenger.so

   PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-2.2.2

   PassengerRuby /usr/bin/ruby1.8

   Estas líneas deben agregarse al final del archivo */etc/apache2/apache2.conf*

8. *Habilitar mod rewrite en el servidor Apache:*
   # a2enmod rewrite
   https://www.sharelatex.com/logout

9. *Crear Virtual host para Apache en /etc/apache2/sites-available/*
   # vim /etc/apache2/sites-available/YOUR-SITE.conf

   Ejemplo de configuración:

   <VirtualHost *:80>

   RailsEnv production

   ServerName 192.168.1.5 # Debe ser la ip real

   DocumentRoot /your_path/public # Note el directorio “public”

   </VirtualHost>

10. *Habilitar el sitio:*
    # a2ensite YOUR-SITE

11. *Reiniciar el Apache:*
    # service apache2 restart

8.4 THE FIRST OPEN UTP PROGRAMMING CONTEST

El día 26 de octubre de 2013 se llevó a cabo *“The first open UTP programming contest”* iniciativa del Semillero In-Sílico, los objetivos principales para desarrollar dicha competencia fueron: que sirviera como un entrenamiento para la Maratón Regional Latinoamericana, [98] por otro lado se vio la oportunidad de probar el Juez de la Universidad Tecnológica de Pereira UTPjudge, por lo que se realizó la mencionada competencia usando el juez de la UTP. La idea fue crear una maratón abierta para cualquier competidor que deseara participar, sin importar su país de origen o si cumplía o no la reglas de elegibilidad según la ACM-ICPC [99], los participantes tenían la opción de participar como equipo o individualmente.

[98] La Maratón Regional Latinoamericana ACM-ICPC, es una competencia que se realiza cada año la cual sirve como selectivo para escoger los equipos para participar en la Maratón Mundial ACM ICPC

[99] Reglas ICPC, [en línea]
<http://icpc.baylor.edu/regionals/rules>

Los problemas fueron escritos en inglés siguiendo las reglas ICPC, dichos problemas fueron escritos por Santiago Gutierrez y Sebastin Gomez, dos de los miembros del equipo clasificado a las World Finals 2013.

Para la competencia se registraron más de 70 equipos, de los cuales 38 hicieron al menos 1 envío y 32 resolvieron al menos 1 problema, para un total de 383 envíos [100]. Los participantes pertenecían a diferentes países tales como: Eslovaquia, Cuba, México, Colombia, República Dominicana, Argentina, Perú, entre otros.

La competencia se desarrollo sin ningún inconveniente, el juez de la UTP funcionó sin ningún problema y fue posible tener una competencia internacional, integrando participantes de distintos países. Fue también una oportunidad para mostrar el trabajo que se ha estado desarrollando para fomentar las maratones de programación desde la Universidad Tecnológica de Pereira.

[100] Anexo al documento se encuentra la dirección del repositorio en donde se pueden encontrar los resultados finales de la competencia

CAPÍTULO 9

DOCUMENTACIÓN DEL JUEZ

9.1 CONFIGURACIÓN DE LOS LENGUAJES

El juez fue desarrollado con la posibilidad de soportar cualquier lenguaje de programación y no un conjunto limitado de lenguajes, para ello se ideó la manera de incluir nuevos lenguajes, cada lenguaje consta de las siguiente características:

- Tipo de lenguaje: El lenguaje puede ser interpretado o compilado.
- Línea de compilación: Si el lenguaje es de tipo compilado se debe indicar la sintaxis para la compilación.
- Línea de ejecución: Se indica la sintaxis para la compilación.

Ejemplo lenguaje compilado

Tipo de lenguaje = "1"
Línea de compilación = "/usr/bin/g++ -Wall -O2 -static -pipe -o SOURCE SOURCE.cpp"
Línea de ejecución = "SRUN -uCP -F10 -tTL -Ujailu -Gjailg -iINFILE -oSOURCE.OUT -eSOURCE.ERR -n0 -C. -f20000 -d512000 -mML ./SOURCE"

Ejemplo lenguaje interpretado

Tipo de lenguaje = "2"
Línea de compilación = " "
Línea de ejecución = "SRUN -uCP -F10 -tTL -Ujailu -Gjailg -iINFILE -oSOURCE.OUT -eSOURCE.ERR -n0 -C. -f20000 -d512000 -mML – /usr/bin/python SOURCE.py"

La línea de compilación y ejecución son plantillas que contienen strings que posteriormente son reemplazados por los datos de cada submisión en particular por ejemplo: la palabra **SOURCE**, hace referencia al nombre del archivo fuente sin la extensión, la palabra **INFILE** hace referencia al archivo de entrada, y las palabras **TL** y **ML** hacen referencia al Timelimit y Memorylimit respectivamente.

Puede encontrar más ejemplos en el archivo *utpjudge/db/seeds.rb*.

9.2 ARCHIVOS DEL SISTEMA

En la carpeta *judgebot* se encuentran distintos archivos:

- *data*: En este archivo se guardan los datos de conexión.
- *pids*: En este archivo se guardan los identificados de procesos del sistema de juzgamiento.
- *judgebot.sh*: Script para iniciar, parar o mirar el estatus del sistema de juzgamiento.
- *judgebot.log*: Es el log de eventos para el sistema de juzgamiento.
- *files*: Es la carpeta en donde se hacen los juzgamientos.
- *sjudge.sh*: Scritp para juzgar cada solución.

Archivo *data*: Como ya se había explicado en el capítulo anterior el juez de la UTP está compuesto por dos partes, una es el sistema de juzgamiento y la otra es la parte web; para permitir la comunicación entre ambas partes es necesario crear el archivo *data* con los datos de conexión necesarios, como: la ip, el puerto, time (intervalo de tiempo en segundos para mirar si hay nuevos envíos), el usuario y la contraseña, es necesario indicar un dato por línea.

Ejemplo de configuración archivo *data*:

192.168.1.5
80
10
usuario
contraseña

9.3 USO DEL SISTEMA DE JUZGAMIENTO

A través del script *judgebot.sh* se puede iniciar, parar o mirar el estatus del sistema de juzgamiento, y se ejecuta de la siguiente manera:
 # ./judgebot.sh (start|stop|status)

CAPÍTULO 10

ANÁLISIS DE LOS DATOS OBTENIDOS Y VALIDACIÓN DE LA HIPÓTESIS

En este capítulo se pretende mostrar los resultados obtenidos a través de las pruebas realizadas al sistema de juzgamiento con el propósito de validar la hipótesis establecida en la metodología de la investigación y determinar si efectivamente se está soportando el conjunto de ataques definidos en la sección 3.4. Por consiguiente encontraremos en la siguiente sección la estimación del parámetro del intervalo de tiempo el cual fue decisivo para llevar una correcta lectura de los datos. Luego en el análisis de los resultados obtenidos, se llevan a cabo dos métodos diferentes, en el primero se realiza una comparación gráfica entre los datos, con el fin de verificar si el sistema de juzgamiento era capaz de seguir con su proceso de juzgamiento, es decir, era capaz de seguir atendiendo mas peticiones, y segundo se sigue la prueba T combinada de dos muestras con el fin de verificar los primeros resultados obtenidos con el análisis gráfico, procedimiento que es definido formalmente en 2.2 y que posteriormente se aplica con los datos en las siguientes secciones.

10.1 ESTIMACIÓN DEL INTERVALO DE TIEMPO ENTRE CADA PETICIÓN

Durante el experimento, los datos fueron recolectados por medio de un script (*Testerboot*, ver sección 7.2.1). Este script básicamente realizaba envíos automáticos hacia el sistema de juzgamiento, tomando la medida de los tiempos que el sistema tardaba para dar respuesta a un determinado problema. Por lo tanto durante el procedimiento fue necesario estimar el tiempo que el script necesitaba entre cada petición, cabe resaltar que en la sección 5 se establece que el sistema de juzgamiento debía ejecutar los programas de manera secuencial y en un solo hilo de ejecución, encolando las peticiones que llegaban en un intervalo de tiempo muy corto, por lo tanto si el tiempo entre cada petición era muy corto, esto provocaba que los problemas que llegaban al juez, se sobrelaparan, lo que impedía dar una lectura correcta sobre los parámetros estadísticos de la media, la varianza y la desviación estándar de las muestras tomadas.

Tabla 10.1: Contraste de Medias, Varianzas y Desviaciones

t (seg)	**Media** $\bar{X}$	**Varianza** S^2	**Desviación** S
0	10.782691387	22.825474224	4.777601304
1	1.676497904	0.019124743	0.138292246
2	1.591838882	0.008567202	0.092559181
3	1.539547282	0.004186227	0.064701060
4	1.545934724	0.005608679	0.074891116
5	1.522338979	0.002563559	0.050631604
6	1.563914392	0.004240305	0.065117623
7	1.549855175	0.003801774	0.061658528
8	1.535053680	0.008731200	0.093440890
9	1.558625085	0.009036179	0.095058821

En la tabla 10.1 se muestra que al tomar diferentes tiempos para el intervalo entre cada petición de cada muestra[101] se nota una gran dispersión de los datos, cuando este tiempo no fue considerado con $t = 0$; ya con $t = 1$ se logra una mayor estabilidad de datos. Pero con las medidas con $t > 1$ se identifica que los datos eran más estables manejando una tendencia entre ellos con menores rangos de variabilidad. De esto se identifica que es necesario determinar un tiempo prudente, que supere el tiempo de sobrelapamiento con el objetivo de tomar unas mediciones correctas. Por lo que se decidió tomar las mediciones cada 5 segundos. En la siguiente sección se mostrarán los datos recolectados tomando un intervalo de $t = 5$.

10.2 PRESENTACIÓN Y ANÁLISIS GRÁFICO DE LOS DATOS OBTENIDOS

A continuación se verá en la tabla 10.2 los tiempos recolectados en segundos con el scritp (*Testerbot*), con el sistema de juzgamiento libre de ataques a partir de un escenario ideal con ciertas restricciones para el experimento tal como se estableció en la sección 7.2.2. Ya en la tabla 10.3 se pueden ver los tiempos recolectados con el sistema atacado, también los datos fueron recolectados por el mismo script.

[101] Estas mediciones se realizaron todas con el mismo problema, tomando pequeñas muestras de tamaño $n = 20$ y variando el tiempo t entre los envíos $0 \leq t \leq 9$

Tabla 10.2: Tiempos recolectados con el sistema libre de ataques

1.4400285	1.478469203	1.497585435	1.545030503
1.452482096	1.479428246	1.497897822	1.547293971
1.452533541	1.481294804	1.499075826	1.548345408
1.45328247	1.481320319	1.499790868	1.549897663
1.453876009	1.481747938	1.499818083	1.552152422
1.455830245	1.481936166	1.499892754	1.554349576
1.461495211	1.483606868	1.500793571	1.557571571
1.463104174	1.483996333	1.501196595	1.559853839
1.465872816	1.485697865	1.51016976	1.561943259
1.46746235	1.485772055	1.510341767	1.564988496
1.468001473	1.486408362	1.511922577	1.565919472
1.469439185	1.486885848	1.512130528	1.570337833
1.469749542	1.487099338	1.519197701	1.571714125
1.470267042	1.487950099	1.521479414	1.575861829
1.470898225	1.488918956	1.522002536	1.57851418
1.471787063	1.489678336	1.522036618	1.596277544
1.472544158	1.490703563	1.522742519	1.607375037
1.472742452	1.491252685	1.524681199	1.60747999
1.473434452	1.493748112	1.524733914	1.615616363
1.474092698	1.494870987	1.525577317	1.616856126
1.474295943	1.496106133	1.527245705	1.618687123
1.475884382	1.496546824	1.53143999	1.631936795
1.47622845	1.496967047	1.531601182	1.737013206
1.477519641	1.497045817	1.539454106	1.755504942
1.47841469	1.497582262	1.539540318	1.810609848
Media ($\bar{X}$)		1.517597802	
Varianza (s^2)		0.003831409	
Desviación (s)		0.061898377	

Tabla 10.3: Tiempos recolectados con el sistema atacado

1.441670202	1.473872469	1.491152737	1.530824422
1.444763892	1.474252137	1.49281077	1.530928026
1.449622856	1.474401486	1.495862927	1.531361813
1.453338321	1.474949388	1.497082975	1.533799648
1.453820997	1.475004982	1.497618385	1.53555672
1.455927892	1.475471174	1.497707798	1.540183454
1.457443403	1.476747323	1.497974146	1.544695353
1.457812148	1.476996776	1.498166383	1.553166505
1.458052308	1.477358738	1.498902153	1.555151354
1.459069834	1.477362589	1.502655646	1.558492886
1.461854901	1.479722375	1.50300247	1.562247009
1.464204224	1.479800976	1.505483981	1.563689901
1.464368967	1.482261008	1.507057312	1.565415703
1.46506859	1.484083999	1.507131623	1.571563938
1.466222261	1.484604372	1.507703016	1.577593173
1.46645742	1.485897848	1.50791937	1.582168017
1.467990763	1.486246723	1.507936717	1.582556273
1.46803211	1.486439061	1.509025836	1.583701326
1.470112593	1.486656108	1.509181743	1.584918436
1.471014841	1.486880884	1.509715047	1.604528528
1.471424525	1.487791887	1.511818455	1.611223115
1.472308587	1.488036267	1.512602363	1.621157665
1.47260988	1.488039956	1.512662355	1.646414675
1.473102759	1.488054594	1.524191633	1.704697845
1.473640426	1.4908182	1.52765908	1.76805375
Media ($\bar{X}$)		1.507788005	
Varianza (s^2)		0.002883488	
Desviación (s)		0.053698122	

Inicialmente es posible notar que los parámetros estadísticos definidos sobre las dos muestras como la media $\bar{x}$, la varianza s^2 y la desviación s son muy similares, especialmente la media que es tomada como parámetro principal en la comparación de los dos conjuntos de datos.

En la gráfica 10.1 se muestra un comparativo entre las dos distribuciones a través de un gráfico de cajas, mostrando los datos en cuartiles [102] junto con sus valores extremos, en donde se puede dar una buena idea de ambas distribuciones e interpretar que tanto para los datos muéstrales del sistema libre de ataques como los datos del sistema atacado guardan una gran similitud. Inicialmente se observa la mediana de ambos conjuntos de datos con un valor tendiente a 1,5 *segs* lo que asegura que el 50 % de los tiempos de ambas muestras fueron tomados por debajo de 1,5 *segs* , luego al observar cómo están distribuidos los datos desde la mediana hasta el cuartil número 3, es decir, desde desde el 50 % hasta el 75 % de los datos medidos en segundos, para la primera muestra, se tiene un valor de 1,540 y en la segunda muestra es de 1,528, finalmente si se observan las extensiones superiores de la gráfica se podrá notar sus valores extremos y por ende el 25 % de los datos restantes con un cierto grado de dispersión, ya que las extensiones superiores de ambas muestras se esta indicando unos valores máximos muy alejados del conjunto de datos. No obstante al contemplar la gráfica 10.2

[102] Un cuartil es una mediada de posicionamiento no central que divide una distribución en cuatro partes iguales.

se notan valores atípicos, lo que podría llevar a un sesgo en las mediciones, pero se considera que no es necesario eliminar estos valores de las muestras debido a que no afectaban significativamente la media $\bar{x}$ de ambas muestras, esto se puede notar en la tabla 10.4.

Figura 10.1: Gráfico de caja y extensión para las muestras del sistema atacado y el sistema libre de ataques

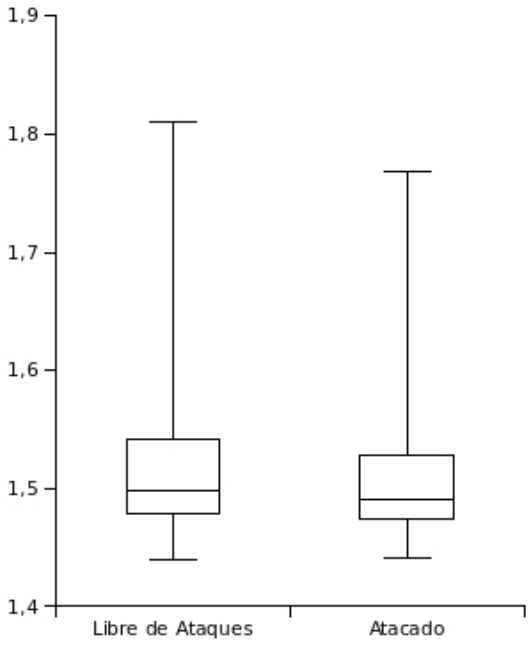
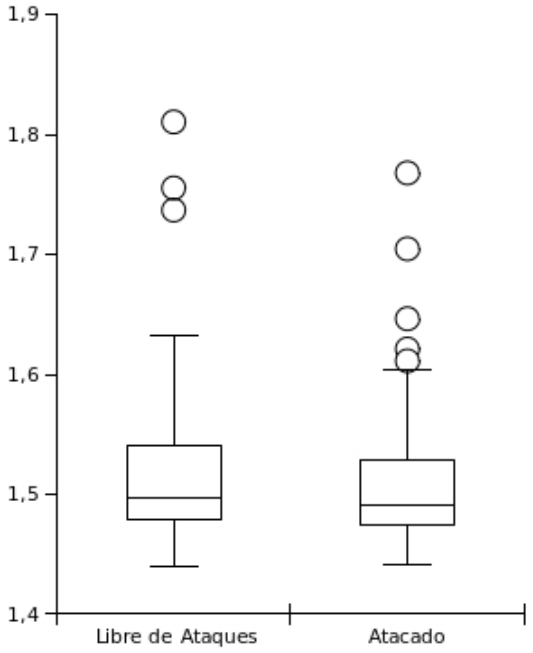


Tabla 10.4: Comparación de medias sin datos atípicos

	Media inicial ($\bar{x}$)	Media sin datos atípicos ($\bar{x}$)
Muestra 1	1.517597802	1,509862394
Muestra 2	1.507788005	1.499234246

Figura 10.2: Gráfico de caja y extensión con valores atípicos para las muestras del sistema atacado y el sistema libre de ataques



De lo anterior se puede concluir que el sistema de juzgamiento no se vio afectado con el conjunto de ataques puesto que siguió respondiendo con los mismos tiempos en ambas muestras, tanto para el sistema libre de ataques como el sistema atacado. En la sección 2.2, se estableció el procedimiento estadístico que se aplicaría con los dos conjunos de datos, ya se vio un acercamiento con las gráficas 10.1 y 10.2 ahora se seguirá a dar los resultados que nos arroja la *Prueba T combinada de dos muestras*.

10.3 PRUEBA T COMBINADA DE DOS MUESTRAS

Puesto que el objetivo es realizar una prueba de hipótesis con el fin de validar los resultados del experimento, como ya se había establecido en la sección 2.2, se tienen la hipótesis nula y alternativa:

- H_0: No existe una diferencia significativa en los tiempos de respuesta entre el sistema libre de ataques y el sistema atacado.
- H_1: Sí existe una diferencia significativa en los tiempos de respuesta entre el sistema libre de ataques y el sistema atacado.

Formalmente como se había establecido en la definición 2.1 tenemos:

$$\begin{cases} H_0 : \mu_1 - \mu_2 = 0 \\ H_1 : \mu_1 - \mu_2 < 0 \end{cases}$$

Para calcular el estadístico t usamos las fórmulas 2.2 y 2.3 con los datos representados en la tabla 10.5

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - d_0}{s_p\sqrt{1/n_1 + 1/n_2}}$$

$$s_p^2 = \frac{s_1^2(n_1 - 1) + s_2^2(n_2 - 1)}{n_1 + n_2 - 2}$$

Tabla 10.5: Datos para la prueba T de dos media

	n	$\bar{x}$	s^2	s
M_1	100	1.517597802	0.003831409	0.061898377
M_2	100	1.507788005	0.002883488	0.053698122

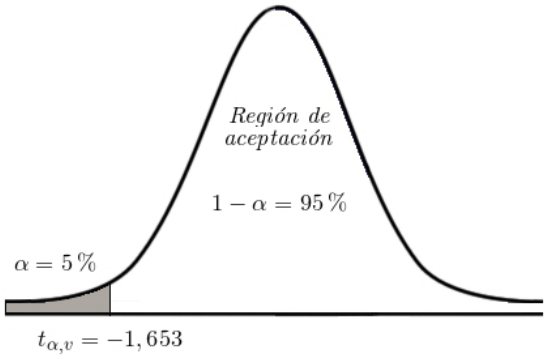
Por lo tanto el valor del estadístico es:

$$t = 1,197127$$

Una vez calculado el valor t, se procede a evaluar el valor crítico $t_{\alpha,v}$ en una distribución *t-student*, donde α es valor de significancia de 5 % y los grados de libertad calculados con la fórmula 2.2 con $v = 198$, en la gráfica 10.3 se puede apreciar la región de aceptación y la región de rechazo de la hipótesis nula H_0.

$$t_{\alpha,v} = -1,652586$$

Figura 10.3: Región de aceptación y de rechazo de la hipótesis H_0



Puesto que se evidencia que la comparación entre el estadístico t calculado con la fórmula 2.2 es mayor que el valor crítico $t_{\alpha,v}$, es decir, $t > t_{\alpha,v}$ no se rechaza la hipótesis nula por lo tanto se concluye que no hay una diferencia significativa entre el sistema atacado y el sistema libre de ataques.

CAPÍTULO 11

CONCLUSIONES

- El principal objetivo de este trabajo fue la realización de un sistema de seguridad que soportara el conjunto de ataques definidos en la sección 3.4, ataques que posteriormente fueron delimitados y clasificados en el capítulo 2. Para tal propósito se realizó un análisis de algunos sistemas de juzgamiento existentes, encontrando características de seguridad comunes a estos sistemas e identificando los requerimientos necesarios para el diseño de un sistema de seguridad para el juez de la UTP, tal como se especifica en el capítulo 5. Con el propósito de validar que efectivamente el sistema de juzgamiento soportara estos ataques, se plantearon dos escenarios para llevar a cabo las pruebas, uno donde el sistema de juzgamiento era libre de ataques y otro en que el sistema era atacado, a fin de recolectar dos conjuntos de datos, para compararlos y posteriormente analizar las variables establecidas en el diseño metodológico a través de los resultados mostrados en las secciones 10.2 y 10.3. Concluyendo finalmente que el sistema no se ve afectado al ser sometido a un conjunto de ataques puesto que sigue respondiendo con los mismos tiempos a las peticiones presentadas durante el experimento.
- Otro aspecto fundamental y que inicialmente no se había planteado como un objetivo de este trabajo, es que el sistema juzgara correctamente las peticiones no ataques, es decir, que para las peticiones consideradas no ataques el sistema devolviera el veredicto correcto; ya que es indispensable para la realización de una maratón de programación, no solo que soporte un conjunto de ataques, sino también que juzgue correctamente. En la sección 6.4 se muestra el proceso de pruebas que se llevó a cabo para verificar el correcto juzgamiento a todos los veredictos definidos, validando efectivamente que la aplicación desarrollada juzga correctamente.
- Se logra evidenciar que existen ataques (ver capítulo 3) que pueden hacer mucho daño a los sistemas de juzgamiento debido a que manejan características similares a diversos tipos de ataques informáticos ya conocidos. Además de que ya se tienen referencia de autores sobre este tipo de ataques, encontrando que se maneja cierta clasificación según el tiempo en que se da dentro de una competencia tales como: *Ataques en tiempo de compilación*, *Ataques en tiempo de ejecución* y *Ataques en tiempo de competencia*.
- Se identificaron diferentes modelos de seguridad que fueron implementados en diferentes jueces de programación, esto permitió tener una idea mucho mas clara de como construir un sistema de juzgamiento propio que pudiera satisfacer los requerimientos para llevar a cabo el presente proyecto.
- Es importante aclarar que el presente trabajo, se concentro básicamente en el desarrollo de un sistema de seguridad para la parte del juzgamiento, el cual es el responsable de calificar debidamente los problemas en una maratón de programación, siendo una pieza imprescindible para un Online Judge que a través de los requerimientos establecidos en el proyecto se logra obtener un excelente juez que puede ser adaptado muy fácilmente a los componentes de un Online Judge.
- Otro punto fundamental que se logra con el presente trabajo fue la creación de un conjunto de pruebas de ataque que recopilara las diferentes vulnerabilidades que se pueden presentar

en una maratón de programación, por lo tanto se logra un aporte en cuanto a la seguridad de este tipo de competencias debido que cualquier juez de maratones puede ser probado con este conjunto de ataques y así poder identificar falencias en la seguridad.

- También se planteo la posibilidad de poder implantar el sistema en un servidor en producción con el fin de probar su funcionalidad y puesta a punto, lo cual se logra con gran éxito con los resultados reflejados tras probar una maratón con problemas de gran nivel tipo ACM-ICPC en *The First Open UTP Programming Contest*.

CAPÍTULO 12

PROYECTOS QUE SE PUEDEN DERIVAR DE ESTA INVESTIGACIÓN

- **Sistema distribuido de juzgamiento**: Un proyecto futuro muy interesante podría ser la creación de un sistema distribuido de juzgamiento, en el cual se disponga de diferentes maquinas para hacer cierta cantidad de juzgamientos y hacer un balanceo de cargas de acuerdo a la cantidad de submissions pendientes y la cantidad de maquinas disponibles.
- **Dar soporte a otro tipo de ataques que superan el alcance de este proyecto**: Para el alcance de este proyecto se limitó a soportar los ataques clasificados en tiempo de compilación y tiempo de ejecución, un proyecto futuro puede extender ese alcance, al tener un soporte de los otros tipos de ataques como lo son: ataques en tiempo de competencia, dentro de los cuales están, uso indebido de la red, explotación de canales encubiertos de comunicación, uso indebido de servicios adicionales, explotación de errores del sistema operativo, los cuales se describen detalladamente en la sección 3.4 del Capítulo 3.
- **Desarrollo de un special judge**: Para cierto tipo de problemas la solución no siempre es única, es decir, que pueden existir múltiples soluciones que resuelven dicho problema, es por esto que para validar si una solución es correcta, es necesario crear un sistema un poco más inteligente que no solamente compare dos archivos, para conocer si la respuesta es correcta o no; los sistemas que evalúan estas soluciones se conocen como *special judges* y se podría continuar el presente proyecto, desarrollando un módulo que soporte esta clase de juzgamiento.
- **Desarrollo de un foro para el juez de maratones UTP**: La creación de un foro dedicado a las maratones, ayudaría a fortalecer el proceso de maratones de programación de la UTP. En el cual se puedan discutir diferentes tópicos y compartir materiales, entre los maratonistas.

A. ANEXOS

- *https://github.com/in-silico/utpjudge*: Repositorio del proyecto UTPJudge
- *https://github.com/jhonber/Judgebot*: Repositorio sistema de juzgamiento, códigos ataques, scripts de pruebas y resultados de la competencia *The First Open UTP Programming Contest*

Bibliografía

[1] Acm icpc the early years acm contest finals. *Disponible en: http://icpc.baylor.edu/ ICPCWiki/Wiki.jsppage=The %20Early %20Years*, 2013.

[2] Acm international collegiate programming contest world finals. *Disponible en: http://www.ibm.com/ developerworks/university/students/contests/acm/index.html*, 2013.

[3] Final uci 2012 del acm-icpc. *Disponible en: http://coj.uci.cu/downloads/files/ finaluci2012/ 01 Convocatoria.pdf*, 2013.

[4] Lina Marcela Jiménez Becerra. Herramienta de estudio para las maratones de programación promovidas en el programa de ingeniería de sistemas y computación de la universidad tecnológica de pereira. *Universidad Tecnológica de Pereira*, 2012.

[5] Joseph Canós, Patricio Letelier, and Mª Carmen Penadés. Metodologías ágiles en el desarrollo de software. *Universidad Politécnica de Valencia, Valencia*, 2003.

[6] Jianwen Wang Cheedoong Drung. Enhance performance of program automatic online judging systems using affinity algorithm and queuing theory in smp environment. *International Conference on Electronic & Mechanical Engineering and Information Technology*, pages 4425–4428, 2011.

[7] DOMjudge. Domjudge administrator's manual. http://home.a-eskwadraat.nl/ domjudge/ docs/ admin-manual.pdf, June 2013.

[8] J Eldering, T Kinkhorst, and P van de Werken. Domjudge website, 2007.

[9] J Garcia-Alfaro and G Navarro-Arribas. Prevención de ataques de cross-site scripting en aplicaciones web.

[10] Song-shan GUO, Lei WANG, and Zi-zhen ZHANG. Acm/icpc and innovative it students cultivating [j]. *Research and Exploration in Laboratory*, 12:059, 2007.

[11] Andres Felipe Marin Rodriguez Juan David Arias Patiño, Miguel Angel Lozano Isaza. Desarrollo de un sistema computacional para el aprendizaje de programación estructurada "scape". *Universidad Tecnológica de Pereira*, 2011.

[12] Peter Loscocco and Stephen D Smalley. Meeting critical security objectives with security-enhanced linux. In *Proceedings of the 2001 Ottawa Linux symposium*, pages 115–134, 2001.

[13] Marcelo Manson. Estudio sobre virus informáticos. *Disponible en: http: http://www. monografias. com/trabajos/estudiovirus/estudiovirus. shtml Acceso*, 10, 2001.

[14] Bruce Merry. Using a Linux security module for contest security. *Olympiads in Informatics*, 3:67–73, 2009.

[15] Bruce Merry. Performance analysis of sandboxes for reactive tasks. *Olympiads in Informatics*, 4:87–94, 2010.

[16] Jorge Mieres. Ataques informáticos. *Debilidades de seguridad comúnmente explotadas). Recuperado http://proton. ucting. udg. mx/tutorial/hackers/hacking. pdf*, 2009.

[17] José Antonio Molina Ortiz, Asunción García González, Azucena Pedraz Marcos, and María Victoria Antón Nardiz. Aprendizaje basado en problemas: una alternativa al método tradicional. *Revista de Docencia Universitaria*, 3(2), 2003.

[18] Sharon L. Myers Keying Ye Ronald E. Walpole, Raymond H. Myers. *Probabilidad y estadística para ingeniería y ciencias*. Pearson, México, octava edicion edition, 2008.

[19] Tocho Tochev and Tsvetan Bogdanov. Validating the security and stability of the grader for a programming contest system. *Olympiads in Informatics*, 4:113–119, 2010.

[20] David A Wheeler. Secure programming for linux and unix howto. 2003.

[21] Jianhua Wu, Shuangping Chen, and Rongrong Yang. Development and application of online judge system. In *Information Technology in Medicine and Education (ITME), 2012 International Symposium on*, volume 1, pages 83–86. IEEE, 2012.

[22] Peng Ying and Wang Fang. Design of and research on distributed oj system based on linux. In *Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on*, volume 1, pages 942–945. IEEE, 2011.