Tarea #3

Instituto Tecnológico de Costa Rica

Carrera: Bachillerato en Ingeniería en Computación

Curso: Principios de Sistemas Operativos

Profesor: Kevin Moraga García

Alumnos:

```
Alberto Zumbado Abarca - 2020095172

Jonathan Quesada Salas - 2020023583
```

Tarea Corta 3: Web Server Attack

Introducción

Con lo que respecta a esta tarea 3 de “Web Server Attack” es la implementación de 3 módulos principales, explicados a continuación:

Pre-thread WebServer: Este mismo va a consistir en crear un HTTP server el cual implementa la técnica llamada prethread. Esta técnica consiste en crear previamente varios hilos de ejecución del método que atiende las solicitudes.

Algo a tener en cuenta es que estos hilos se crean utilizando la biblioteca pthreads de Unix. Debe de recibir como parámetro el número de hilos N que se deben pre-crear, el WebServer escuchará en el puerto estándar de HTTP, y tendrán N hilos posibles para atender la solicitud, cada solicitud se atenderá por el primer hilo que esté disponible.

En este mismo módulo debe de ir contemplado de que no existan más disponibles mostrará un mensaje de error, indicando que se ha sobrepasado el número de clientes que pueden ser atendidos.

HTTPClient: Se deberá crear n cliente HTTP el cual permita descargar un binario a través de una lista de comandos en los parámetros o bien interactuar con el servidor HTTP como cualquier otro cliente HTTP; esto mismo se debe utilizar la biblioteca curl en el lenguaje de programación previamente definido.

StressCMD: Este módulo deberá crear una aplicación que reciba como parámetro un ejecutable. Luego debe de crear la cantidad de hilos que el cliente especifique, con el objetivo de lanzar un ataque de Denegación de Servicio.

Adicionalmente se debe recalcar el objetivo que tiene dicho módulo, el cual es aturar los WebServers hasta que estos se queden sin posibilidad de atender otro cliente más.

Este módulo va a contener consideraciones adicionales, las cuales serán: El WebServer deberá de servir los archivos que se encuentren disponibles en la dirección especificada en los parámetros. El WebServer debe de implementar los métodos POST, GET, HEAD, PUT, DELETE de HTTP. Es necesario documentar todos los métodos de HTTP incluyéndose en la introducción de la documentación. Se deberá de permitir cambiar de protocolo, y poder implementar una respuesta válida para una consulta con cualquiera de los siguientes protocolos: FTP, SSH, SMTP, DNS, TELNET, SN En cuanto a la

utilización de los protocolos se puede incluir un parámetro más, al momento de instanciar el servidor HTTP, que defina el protocolo. De lo contrario utilizar el protocolo, descriminándolo a partir del puerto por defecto.

Ambiente de desarrollo

Con lo que respecta al ambiente de desarrollo se usará el sistema operativo Ubuntu por parte de Jonathan Quesada Salas y Alberto Zumbado Abarca, específicamente la versión:

Ubuntu 20.04.2 LTS

En cuanto a los IDLE se que usarán serán el Visual Studio Code para el desarrollo de la tarea y adicionalmente Eclipse para la funcionalidad del debugger que tiene dicho IDLE para el lenguaje de programación Rust.

Se usará el lenguaje de programación Rust para la resolución de dicha tarea 3 Web Server Attack.

Adicionalmente se usará Python3 para la realización del módulo StressCMD.

Se contará con un repositorio llamado “ic-6600-t3-webServerAttack” en la plataforma de Github para poder establecer el código fuente de dicha tarea.

Estructuras de datos usadas y funciones

Con lo que respecta a las estructuras de datos pueden resaltar principalmente 3 estructuras, las cuales serán:

- client
- web-server
- stresscmd

Iniciando con el ***client*** se resaltan la carpeta de recursos la cual va a contener los archivos de extensión html y adicionalmente contendrán **src** se encontrarán los siguientes archivos explicados a continuación:

- **client.rs:** Este archivo se encargará de las tareas que pueda realizar el cliente en cuanto respecta a GET, PUT, DELETE, POST, HEAD. Dichos métodos se podrá notar una gran similitud en el cuanto a código fuente.
- **get_options.rs:** Se encargará de parsear el comando ingresado en la terminal, por parte del cliente para poder llegar a solicitar un recurso.
- **main.rs:** Se encargará de organizar la selección de los get options por medio de vectores y las solicitudes del cliente se verán dependientes del host, el método y el recurso.

En cuanto respecta a ***web-server*** se resaltan la carpeta de recursos que la cual va a contener los archivos de extensión html que se utilizarán para la realización de pruebas y adicionalmente en la carpeta de src contendrá la siguiente estructura de archivos explicados a continuación por carpetas:

- **get_ops:** En esta carpeta se encontrará el siguiente archivo:
  - **get_options.rs:** Este archivo se encargará de parsear el mensaje por parte del usuario para la ejecución del web server, en cuanto se refiere la formato que debe de tener el comando.
- **http_server:** Esta carpeta tendrá el siguiente archivo:

    - **http_handler.rs:** Este archivo tendrá en cuenta las siguientes funciones necesarias para su funcionamiento:
        - **create_connection:** La entrada de esta función será un string la cual el objetivo de la misma será crear una conexión en la web.
        - **handle_connection:** Esta función se encargará de manejar la conexión la previamente creada con ayuda de threadpool la cual permite utilizar hilos de una manera eficiente proporcionando a una aplicación un área.
        - **handle_stream:** Se encargará de manejar la determinada solicitud que pueda ingresar el cliente en cuanto el método especificado en la tarea, como puede ser GET, POST, HEAD, PUT y DELETE-
        - **get_method:** Se encargará de manejar especificamente la solicitud GET por parte del Web Server
        - **head_method:** Se encargará de manejar especificamente la solicitud HEAD por parte del Web Server
        - **delete_method:** Se encargará de manejar especificamente la solicitud DELETE por parte del Web Server
        - **put_method:** Se encargará de manejar especificamente la solicitud PUT por parte del Web Server
        - **post_method:** Se encargará de manejar especificamente la solicitud POST por parte del Web Server

- La carpeta de **parser** tendrá el siguiete archivo:
    - **request_parser.rs:** Este archivo solo tendrá una funció la cual será explicada a continuación:
        - **build_request:** Esta función se encargará de darle forma al request, de manera que pueda llegar a retornar un determinado recurso, por ejemplo puede ser index.html

- La carpeta **request** solo contendrá solo un archivo, sería el siguiente:
    - **request.rs:** Este archivo solo contendrá una estructura de datos la cual está formada por el método, recurso y todo el cuerpo del request.

- La carpeta **response** contendrá solo un archivo el cual será:
    - **response.rs:** Este archivo contendrá un ENUM el cual comprenderá mensajes de OK y NotFound, en cuanto a la busqueda de recursos. Y adicionalmente se comprenderá las siguientes funciones de dicha implementación del ENUM:
        - **get_status:** Esta función se encargará de obtener el estado del servidor el cual serán dos posibilidades: OK o NotFound
        - **get_code:** Esta función obtendrá el código dado por el estado del servidor.
        - **get_message:** Esta función se encargará de a partir del código generado del estado se procederá a obtener el respectivo mensaje de la solcitud.
        - **generate_response:** Esta función se encargará simplemente de generar la respuesta a la solicitud enviada. Adicionalemente se encuentra el **main.rs** este mismo archivo se encargará de primeramente generar un vector de todos los argumentos, para luego posteriormente obtener las opciones ingresadas para parsearlas de manera que puedan ser procesadas, después poder establecer el puerto y los hilos a usar. Adicionalmente se ocupa ThreadPool para el manejo de los múltiples hilos, para posteriormente crear una

conexión y que esta misma conexión pueda ser manejada por medio del socket.

Por último se encuentra la carpeta ***bin*** en esta se encuentra los dos binarios que son los de Web-Server y el de HTTPCliente, adicionalmente esta contendrá un archivo llamado stresscmd, el cual su función es en base a los parámetros que debe de tener la línea de comando tendrá una función llamada **attack** la cual correrá un subproceso, para luego ejecutarlo múltiples veces en un **while**.

Instrucciones para ejecutar el programa

Con lo que respecta a las intrucciones de uso se establece primeramente la generación de los binarios respectivos del programa que van a ser el del WebServer y HTTPClient de la siguiente forma:

```
cargo build
```

Ya habiendo creado los binarios se podrá ejecutar el funcionamiento del WebServer con el siguiente comando:

```
prethread-WebServer -n <cantidad-hilos> -w <HTTP-root> -p <port>
```

Para la ejecución del cliente se necesita digitar el siguiente formato de comando para poder hacer la conexión del cliente con el servidor de la siguiente manera para poder llamar a la lista de comandos que se desea ejecutar:

```
HTTPclient -h <host-a-conectar> [<lista-de-comandos-a-ejecutar>]
```

Y por último se determina la sintaxis para el StressCMD, el cual su función es estresar al servidor con hilos de ejecución para poder probar el funcionamiento del mismo, para esto se deben establecer los parametros del cliente y la cantidad de hilos de ejecución del programa, por lo cual se necesita el siguiente comando:

```
stress -n <cantidad-hilos> HTTPclient <parametros del cliente>
```

Actividades realizadas por estudiante

Fecha	Inicio	Fin	Avance Realizado
06/09/2022	6:00 pm	8:00 pm	Empezar el Kick-off de la tarea 3 (Introducción, Ambiente de desarrollo y Control de versiones)
07/09/2022	6:00 pm	9:00 pm	Terminar el Kick-off de la tarea 3 (Diagrama UML)
08/09/2022	6:00 pm	10:00 pm	Empezar a realizar el WebServer
09/09/2022	8:00 pm	11:00 pm	Hacer correr el WebServer de manera que pueda retornar un html en un puerto
10/09/2022	9:00 am	2:00 pm	Empezar a realizar el HTTPClient para el WebServer
11/09/2022	9:00	12:00	Continuar con retoques con el WebServer y el

	am	md	HTTPCliente acorde a la especificación
12/09/2022	8:00 am	11:00 md	Implementación del cliente
12/09/2022	1:00 am	3:00 pm	Agregar incisos faltantes en la documentación

---

Autoevaluación

Estado final

El estado final de esta tarea fue completada al 100% en cuanto a los requerimientos técnicos como tal fueron completados satisfactoriamente.

Problemas y limitaciones

1. Con lo que respecta a las limitaciones la primera a resaltar fue buscar como realizar busquedas con lo que respecta a lo que es un WebServer, ya que ambos habíamos visto un funcionamieno de HTTP de forma de empezarlo de cero, pero al insvestigar sobre el funcionamientos sobre el HTTP en Rust no percatamos que dicho funcionamiento puede realizarse más modular, ya que Rust tiene un mejor manejo para esta tarea.
2. Al iniciar el WebServer nos topamos con el problema con el problema sobre unwrap la cual consiste en algo en Rust es decir: "Dame el resultado del cálculo y, si hubo un error, entra en pánico y detén el programa". Sería mejor si mostráramos el código para desenvolver porque es muy simple, pero para hacer eso, primero necesitaremos explorar los tipos de opción y resultado. Y dicho funcionamiento no lo habíamos contemplado sobre esta tarea
3. Con el HTTPClient uno de los problemas que nos llegó es que no sabíamos exactamente como poder conectarlo al WebServer o que el HTTPClient llegará a llamarlo, entonces al final encontramos la referencia [9] la cual nos ayudó a poder establecer de mejor manera el problema
4. Uno de los principales problemas que se llegó a contemplar a la hora de implementar el WebServer fue el funcionamiento sobre threadpool el cual consiste en Un grupo de subprocesos que se utiliza para ejecutar funciones en paralelo. Genera un número específico de subprocesos de trabajo y repone el grupo si alguno de los subprocesos de trabajo entra en pánico.
5. Adicionalmente otro problema fue que es imposible exagerar la importancia de los clientes HTTP, por lo que debe hacer su tarea al elegir el cliente HTTP adecuado para su tarea en cuestión. Según la pila tecnológica en la que se basa su proyecto o la cantidad de llamadas HTTP que debe realizar, ciertas bibliotecas se adaptarán bien a usted, mientras que otras podrían ser más problemáticas de lo que valen.

Reporte de commits

```

```

Auto-evaluación

Rubro	Porcentaje
WebServer	40%

Implementación de Protocolos	15%
HTTPclient en Rust	15%
Stress-Client	15%
Documentación	20%
Kick-off **(EXTRA)**	5%

Evaluación

Rubro	Puntaje
Aprendizaje de pthreds	5
Aprendizaje de comunicacion entre procesos	5
Aprendizaje de sockets	5
***	

Lecciones Aprendidas

1. Una de las principales lecciones aprendidas se pueden comprender en la creación de un web server en rust, ya que fue algo totalmente nuevo que se hemos llegado a hacer, en cuanto a lo que llegamos a averiguar es que la documentación y los tutoriales que pueden haber en la red puede ser muy útiles, ya que explican muy bien dicho funcionamiento.
2. Adicionalmente crear un cliente para HTTP fue una tarea curiosa ya que todas las solicitudes que puedan generar la estructura que puedan es muy similar en el código.
3. En cuanto se refiere a la creación del script de python fue una tarea relativamente ya que simplemente sera establecer los argumentos de manera que cumpla los requerimientos y adicionalmente a esto, es que al inicio el stressCMD no funcionaba ya que procesaba los hilos de manera secuencial dado error en el requerimiento técnico sobre que el servidor tenía que caer, por lo cual lo que se propuso fue la creación de una función que fuera attack que tuviera la ejecución de un subproceso y que esta función fuera llamada en un ciclo ejecutando los hilos de golpe.
4. Algo importante que fue bueno aprender y aplicar fue la funcionalidad de los get options, los cuales servieron para la creación del parseo de los argumentos que puedan llegar a procesarse en el programa.
5. En general esta tarea fue importante establecer una modularización adecuada para el manejo del parseo, las solicitudes y las respuestas que pueda tener el servidor dependiendo de las decisiones que tome el cliente, y por parte del stressCMD fue interesante hacer que un servidor pueda caerse con tan pocas lineas de código, se siente poderoso.

Bibliografía

[1] Final Project: Building a Multithreaded Web Server - The Rust Programming Language. (s. f.). Learn Rust - Rust Programming Language. https://doc.rust-lang.org/book/ch20-00-final-project-a-web-server.html

[2] How to choose the right Rust HTTP client - LogRocket Blog. (s. f.). LogRocket Blog. https://blog.logrocket.com/the-state-of-rust-http-clients/

[3] http_client - Rust. (s. f.). Docs.rs. https://docs.rs/http-client/latest/http_client/

[4] method.rs - source. (s. f.). Docs.rs. https://docs.rs/http/latest/src/http/method.rs.html#85

[5] pthreads(7) - Linux manual page. (s. f.). Michael Kerrisk - man7.org. https://man7.org/linux/man-pages/man7/pthreads.7.html

[6] Rhymu's Videos. (2020, 1 de noviembre). 0419 -- Rust: HTTP client/server [Video]. YouTube. https://www.youtube.com/watch?v=ve6ZYiLfHCs

[7] Rust. (2022a, 31 de enero). Rust Linz, January '22 - curl with Rust by Daniel Stenberg [Video]. YouTube. https://www.youtube.com/watch?v=HFH2vZRTKrA

[8] Rust. (2022b, 31 de enero). Rust Linz, January '22 - curl with Rust by Daniel Stenberg [Video]. YouTube. https://www.youtube.com/watch?v=HFH2vZRTKrA

[9] Technologies In Industry 4.0. (2021, 20 de octubre). Rust HTTP Client. Medium. https://medium.datadriveninvestor.com/rust-http-client-736c1a84acf7

[10] thread 'main' panicked at 'called `Result::unwrap()` on an `Err` value: UrlRequest(Error(Url(RelativeUrlWithoutBase)))', src\libcore\result.rs:1165:5. (s. f.). Stack Overflow. https://stackoverflow.com/questions/60912754/thread-main-panicked-at-called-resultunwrap-on-an-err-value-urlreque

[11] TPPZ builds things. (2021, 28 de septiembre). How to write an HTTP client in Rust with the ureq library [Video]. YouTube. https://www.youtube.com/watch?v=Lh037qqW92A

[12] threadpool - Rust. (s. f.). Docs.rs. https://docs.rs/threadpool/latest/threadpool/

[13] Python DDOS Script for novice - VPSHELPDESK.COM. (s. f.). VPSHELPDESK.COM. https://vpshelpdesk.com/2021/10/06/python-ddos-script-novice/