Instituto Tecnológico de Costa Rica

Principios de Sistemas Operativos

Kick-off de la Tarea 3 - Web Server Attack

Profesor:
Kevin Moraga García

Alumnos:
Alberto Zumbado Abarca
Jonathan Quesada Salas

08 de septiembre

2022

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.

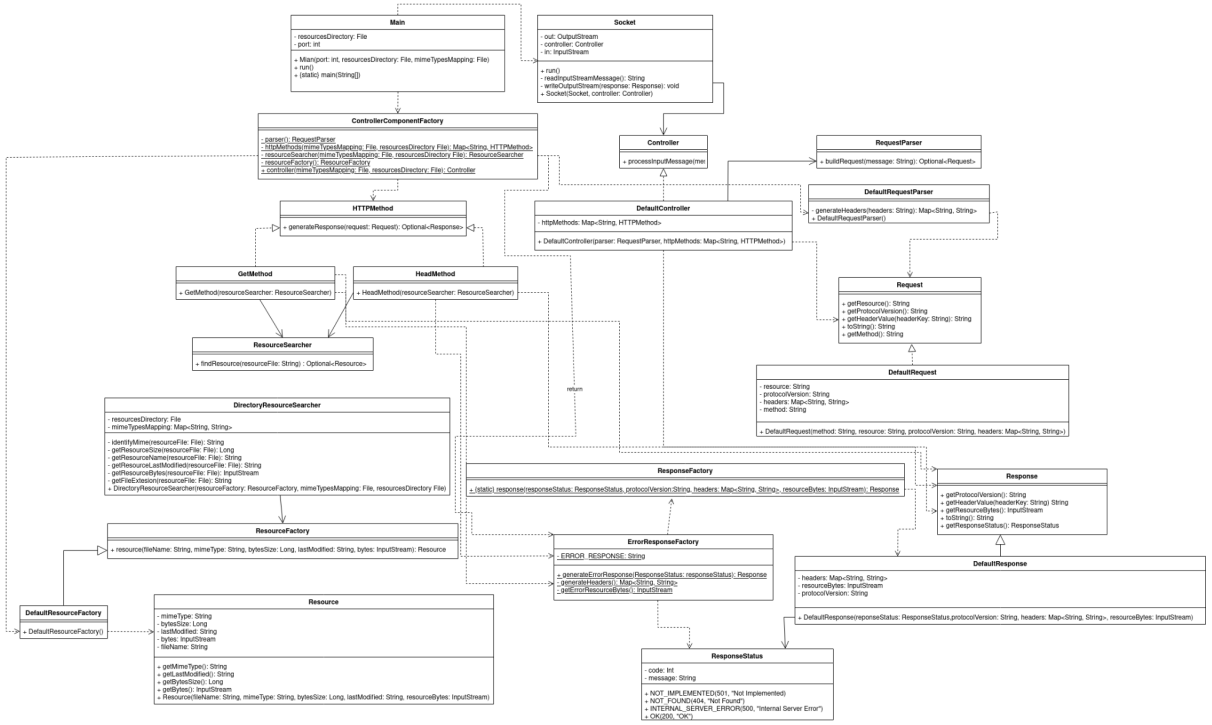
Se contará con un repositorio llamado “ic-6600-t3-webServerAttack” en la plataforma de Github para poder establecer el código fuente de dicha tarea.

Control de Versiones

A continuación se adjunta el link para el repositorio de la tarea 3 del curso de Principios de Sistemas Operativos llamada Web Server Attack:

https://github.com/betozumbado/ic-6600-t3-webServerAttack

Diagrama UML



En caso que no se pueda apreciar bien, se adjunta el link de draw.io para una mejor visualización:
https://drive.google.com/file/d/1jDgCTAnCxZlSD8niylkKG_7WMVBypXp-/view?usp=sharing

Con lo que respecta a las decisiones de diseño de esta tarea fue contemplar respectivamente las necesidades principales que tiene un Web Server en base a los protocolos de HTTP ya que estos mismo van a llegar a poder comportarse de una manera dinámica dependiendo de que protocolo se ejecute o bien si se puede llegar a producir algún error.
Por lo cual primeramente se decidió establecer una clase principal la cual va tener una conexión directa con el Socket para que este mismo pueda llamar al controlador de manera pueda este mismo poder obtener una respuesta de la solicitud que puede ser GET o HEAD, estos mismos métodos son clases aparte que tienen conexión directa con los recurso y la búsqueda del mismo, ya que esta misma clase servirá para como un directorio de recursos que pueden verse contemplado búsquedas del mismo, para que de esta manera poder comprender la creación de un HTTP server y adicionalmente se puede ver contemplada la atención de la solicitudes por parte de las clases Request, que previamente fue diagnosticada por parte del controller y enviado al socket, para poder establecer hilos de ejecución del mismo, para que esto mismo pueda ser un poco más modularizado y eficiente.
Algo importante es que las respuestas deberá ser impresas en pantalla por el usuario, entonces se estableció la creación de clases de parser para poder establecer el formato de los mensajes impresos en pantalla y también que los request pueden tener un formato específico del mismo para que cuando se llegue a buscar un recurso se pueda determinar un respectiva respuesta válida o de error del mismo, mencionando los errores, se estableció una clase para el manejo de errores para que no influya en los patrones de diseño de otras clases aparte con responsabilidades específicas que no tendría sentido intervenir en ese flujo de trabajo.