Relatório - Um Inspetor HTTP baseado em Proxy Server
Teleinformática e Redes 2 - João José Costa Gondim

Roberta Costa Silva - 14/0170723 Roberto Nishino - 10/0122272

1. Apresentação teórica

1.1. Protocolo TCP

O protocolo TCP atua na quarta camada de abstração da pilha de protocolos Internet, a camada de transporte, que provê comunicação lógica entre aplicações que rodam em diferentes hosts.

O seu funcionamento é orientado à conexão, ou seja, diferente do protocolo UDP, existe uma confirmação de conexão entre os dois participantes de um comunicação antes de se iniciar a transferência de dados. Assim, provê comunicação confiável, com confirmação de recebimento dos dados transmitidos.

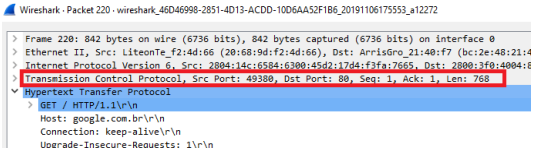
O protocolo TCP permite realizar comunicação full-duplex, sendo possível, após a conexão entre dois hosts ser estabelecida, a troca de dados entre os processos de cada host ser realizada de forma bidirecional. Conexões TCP, além de full-duplex, também são sempre ponto-a-ponto, isso quer dizer, realizada diretamente entre os participantes, sem a intermediação da comunicação, entre um único emissor e um único receptor.[1]

1.2. Protocolo HTTP

O protocolo HTTP atua na quinta camada de abstração da pilha de protocolos Internet, a camada de aplicação, que provê serviços para o usuário final com a interação com as interfaces das aplicações.

A comunicação HTTP é utilizada para requisições de Web clients para Web servers, bem como para a resposta dessas requisições, ou seja, como o servidor transfere Web pages para os clientes.

O protocolo HTTP usa o protocolo TCP como protocolo de transporte, isto é, através dos sockets do Web client e do Web server, após a conexão TCP ser estabelecida, as mensagens HTTP (requisições e respostas) são trocadas. [1]



Há diversos métodos que o HTTP suporta para realizar a comunicação entre aplicações cliente-servidor tais como: GET, CONNECT, POST, OPTIONS, PATCH, TRACE, PUT, DELETE e HEAD. Nas requisições, há um GET não-condicional com a URL requisitada, o nome do host, o campo *Connection* que mostra ao host se a conexão permanece aberta, *keep-alive*, o campo *Accept* que mostra o tipo de arquivo que a requisição aceita, o idioma utilizado pelo cliente, existe também um espaço para realizar o envio de arquivos, textos, JSON, XML, dentre outros, utilizando-se, geralmente, de métodos que realizam operações de persistência e outras coisas como *Accept-Encoding* que mostra quais formatos de codificação são aceitos.

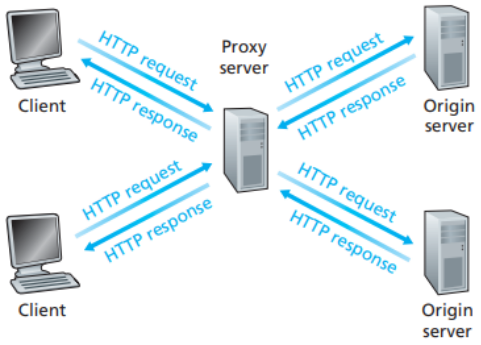
Nas respostas, há a versão do protocolo HTTP utilizada, no caso a 1.1, o tamanho do conteúdo, o tipo de conteúdo, a data de quando foi gerada, o campo *Connection*, assim como na requisição, um código de retorno que mostra o status do servidor após tratar a requisição, esses códigos podem ser: 101 *Switching Protocols*, 200 *OK*, 201 *Created*, 206 *Partial Content*, 304 *Not Modified*, 400 *Bad Request*, 401 *Unauthorized*, 403 *Forbidden*, 404 *Not Found*, 500 *Internal Server Error* e 502 *Bad Gateway*. O 200 *OK* mostra que a requisição foi bem sucedida por exemplo, já o 404 *Not Found* mostra que não foi possível encontrar o recurso, o 500 *Internal Server Error* mostra um funcionamento incorreto do servidor de aplicação e o 502 *Bad Gateway* mostra um funcionamento incorreto da rede.

1.3. Proxy Server Web

Um servidor proxy web recebe requisições HTTP na rede no lugar do servidor web de destino original, funcionando como servidor para o cliente final e como cliente para o servidor de origem. No caso do web, guarda cópias dos objetos recentemente requeridos. Quando a resposta da requisição feita pelo cliente não está guardada no web cache, outra forma de chamar servidor proxy, essa requisição é enviada para o servidor web de origem, também através de conexão TCP, como na imagem a seguir retirada do livro Kurose[1], caso esteja guardada na cache, o conteúdo da requisição é enviado imediatamente. O proxy melhora o tráfego na rede e possibilita uma diminuição no tempo de espera quando se utiliza da cache para tal.

No servidor proxy a ser apresentado neste trabalho, é possível analisar as requisições HTTP entre o cliente e o

servidor de origem, assim como realizar alterações nestas. Esse tipo de intercepção se realizado de forma transparente, ou seja, que alterações sejam realizadas sobre as requisições sem que o cliente perceba, pode ser usado para roubo de informações sensíveis, ataque conhecido como *Man in the Middle*, mas tal ataque e suas técnicas de segurança não serão abordadas neste trabalho.



1.4. Spider

O *Web Crawler*, conhecido como *Spider*, é um rastreador de páginas utilizado para a indexação Web que identifica todos os links de uma página e os adiciona a uma lista de URLs, as quais serão visitadas de forma recursiva.

1.5. Funcionalidades do Inspetor HTTP baseado em Proxy Server

As funcionalidades a serem implementadas são:

- Proxy: Recebendo uma requisição HTTP, deve processar seu cabeçalho e verificar o endereço de destino, também deve ser possível editar tanto a requisição quanto a resposta;
- Spider: Enumerar todas as URLs subjacentes de uma URL definida formando uma árvore hipertextual;
- Cliente Recursivo: Corrigir as referências a uma URL para que a árvore obtida seja visualizada no navegador como local.

2. Arquitetura

O projeto foi implementado com a linguagem C++, sistema operacional macOS Catalina e utilizando o QT como IDE e implementação das telas. O framework utilizado permite que a aplicação funcione de forma assíncrona e orientada a eventos, utilizando-se de listeners de sinais para tal.

2.1. Aracne

Essa classe é responsável pela interação com a aplicação, pela comunicação da UI com esta. Suas funcionalidades envolvem tratar os sinais dos eventos, sinais de enviar requisição para o Proxy, para ser enviada ao host, sinais de enviar respostas para o Proxy, para ser enviada ao browser, sinais para ligar/desligar o servidor *Proxy*, sinais para receber novas requisições e respostas, colocando-as nas suas respectivas listas, possibilitando a edição destas enquanto aguardam os sinais de seus tratamentos e sinais de *callback* das requisições e respostas enviadas.

2.2. HTTP Parser

Essa classe é responsável por tratar o cabeçalho das requisições. Dar get em informações como conteúdo, versão do HTTP utilizada, tipo de requisição, tamanho da requisição, nome do host que a gerou, dentre outros.

2.3. Proxy

Essa classe possui a responsabilidade de se instanciar o servidor, que se comunica por meio de uma porta específica, por *default* 8228. Para cada conexão que o server envia para o *Proxy*, instancia-se um *Client* para tratá-la, passando-se um *file descriptor* da conexão para que esta seja possível. Dessa forma, o *Proxy* não fica como busy e pode tratar novas conexões instanciando novos *Clients*, podendo gerar, se necessário, uma fila de *Clients* com conexões ativas para serem tratados pela aplicação e pelo browser.

2.4. Socket

Essa classe possui a funcionalidade de worker para o *Client*, será a responsável pela interação do host com este. Suas funcionalidades envolvem estabelecer a conexão com o host determinado no header da requisição, enviar a requisição e aguardar a resposta, assim que esta for recebida, será enviada ao *Client* para então ser tratada no Aracne. Após cumprir suas responsabilidades, o socket é destruído.

2.5. Client

Essa classe é responsável por instanciar e administrar *Sockets*. Suas responsabilidades envolvem receber as requisições do *Proxy*, instanciar *Sockets* para tratar a conexão com o host, após receber a resposta do host, ficará aguardando pelo evento do Aracne para enviar a resposta para este, enquanto aguarda pelo evento, é possível realizar a edição da resposta, ao receber o sinal, envia a resposta modificada ou não para o Aracne. Após cumprir suas responsabilidades, o *Client* é destruído.

3. Funcionalidades implementadas

Foram implementadas as funcionalidades básicas do servidor *Proxy*, receber e tratar a requisição do Browser, configurado para utilizar o IP e porta definidos no *Proxy*, sendo possível realizar a edição da requisição antes de ser enviada para o host determinado na requisição, receber a resposta do host, também sendo possível realizar a edição antes de ser enviada ao Browser.

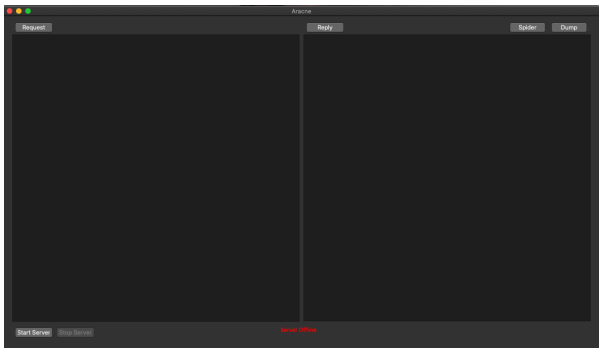
Foi implementada a UI da aplicação utilizando a ferramenta *QT*, esta pode ser vista nos *screenshots* mais ao final deste documento. Nela também é possível ligar/desligar o servidor *Proxy*, verificar seu status, Online e Offline, e apresenta os botões para as futuras funcionalidades Spider e Dump.

Para se realizar a edição das requisições e/ou das respostas nas respectivas listas de requisições e respostas, necessita-se dar um *double click* em alguma e realizar a alteração desejada, atentando-se em manter o padrão da mensagem HTTP a fim de permitir o correto funcionamento da aplicação, apertar a tecla *enter* ao terminar a edição. Para enviá-las, clicar no botão de *Request* para enviar a primeira requisição da lista de requisições e *Reply* para enviar a primeira resposta da lista de requisições.

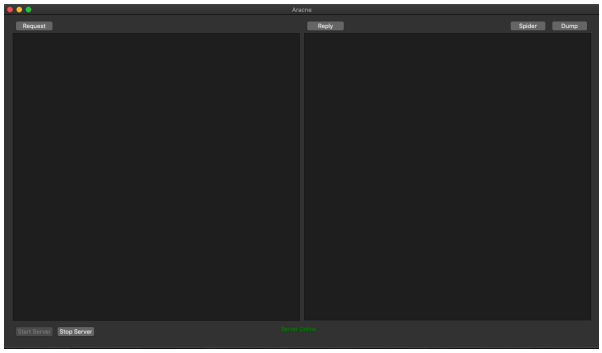
Não foram implementadas as funcionalidades Spider e Dump.

3.1. Screenshot das funcionalidades implementadas

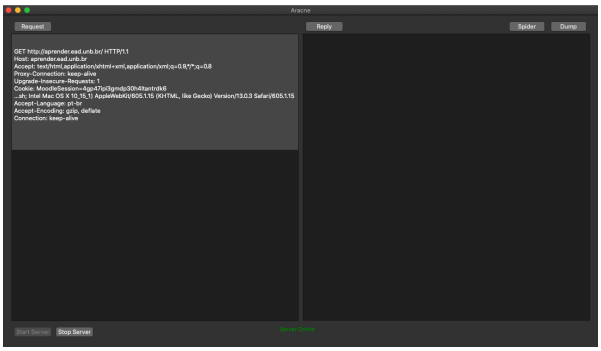
- Servidor com status offline



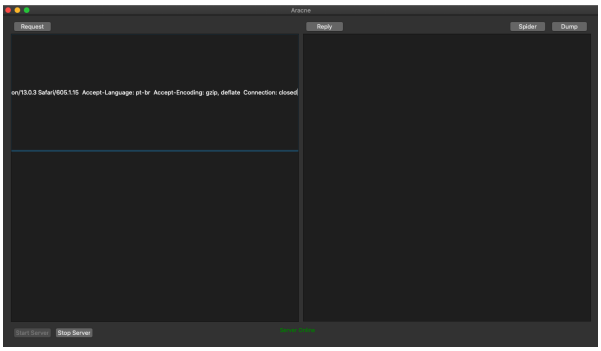
- Servidor com status online



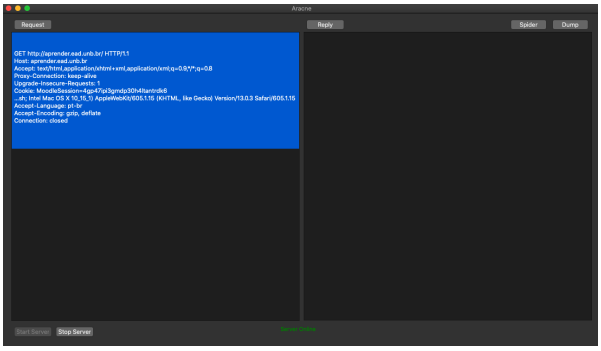
- Exemplo de request na lista de requests



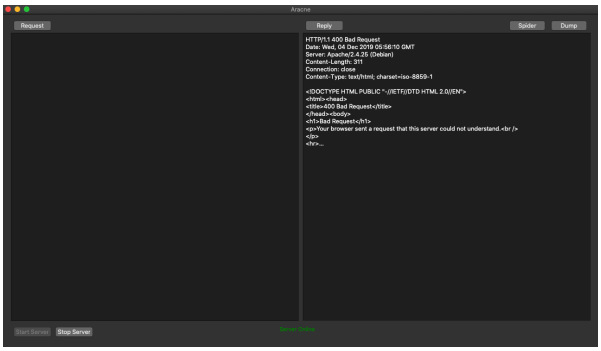
- Editando o request



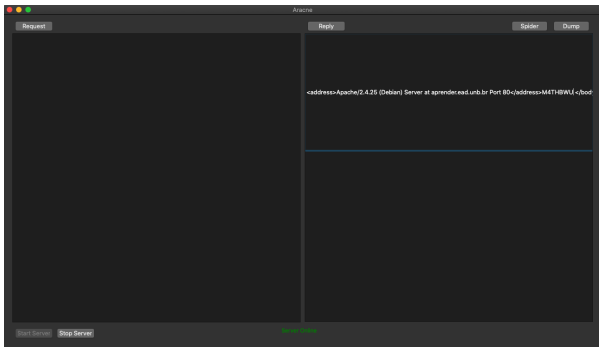
- Request editada



- Reply gerada na lista de replies



- Editando o reply



- Confirmação no browser da reply editada

Bad Request

Your browser sent a request that this server could not understand.

Apache/2.4.25 (Debian) Server at aprender.ead.unb.br Port 80
M4THBWU <

4. Documentação

A documentação foi feita no próprio código, os dois podem ser encontrados em: Repositório do Projeto

Referências

[1] J. F. Kurose. *Computer Networking: A Top-Down Approach*.