Inspetor de Tráfego HTTP

Guilherme da Silva Fontes Lopes, Rafael Martins Pereira Chianca

11 de dezembro de 2018

Resumo

O objetivo deste relatório é descrever o funcionamento de um inspetor HTTP baseado em um servidor proxy. Ao longo do relatório descreveremos os conceitos teóricos necessários para entendimento do inspetor HTTP assim como detalharemos toda a arquitetura e codificação necessária para implementação do trabalho proposto.

1 Introdução

Um inspetor de tráfego HTTP tem como objetivo analisar o tráfego de rede utilizando o protocolo HTTP. O relatório será organizado da seguinte forma: na sessão 2, será apresentada a fundamentação teórica para o entendimento do trabalho; na sessão 3, será apresentada a arquitetura do sistema implementado; na sessão 4, será apresentada uma descrição do códigio fonte do sistema; por fim, na sesão 5, serão apresentados resultados do funcionamento do sistema.

2 Apresentação Teórica

2.1 TCP

O TCP(*Transmission Control Protocol*), é um protocolo da camada de transporte do modelo OSI, orientado a conexão e responsável pelo funcionamento de protocolos como *SSH, FTP, HTTP* entre outros. Entre as suas principais características estão: **orientação a conexão, conexão ponto-a-ponto, serviço de conexão full duplex, confiabilidade, controle de fluxo e controle de congestionamento**, as quais seram descritas a seguir.

O TCP é dito **orientado a conexão** pois, para que haja transmissão de dados entre dois processos, é necessário previamente ocorra o chamado *handshake*, geralemnte em 3 passos, o qual consiste em um envio prévio de alguns segmentos para estabelecer a conexão e garantir a transmissão dos dados. Os passos para que ocorra o *handshake* são descritos na Figura 1

Estabelecida a conexão após o **handshake**, temos que a característica de conexão **ponto-a-ponto** do TCP, faz com que somente seja possível a transmissão de dados entre **um único transmissor e um único receptor** em um determinado momento. Não é possível que haja uma comunicação simultânea entre três processos no TCP. O sistema **full-duplex**, faz com que seja possível a transmissão de dados simultâneos entre os dois participantes da conexão.

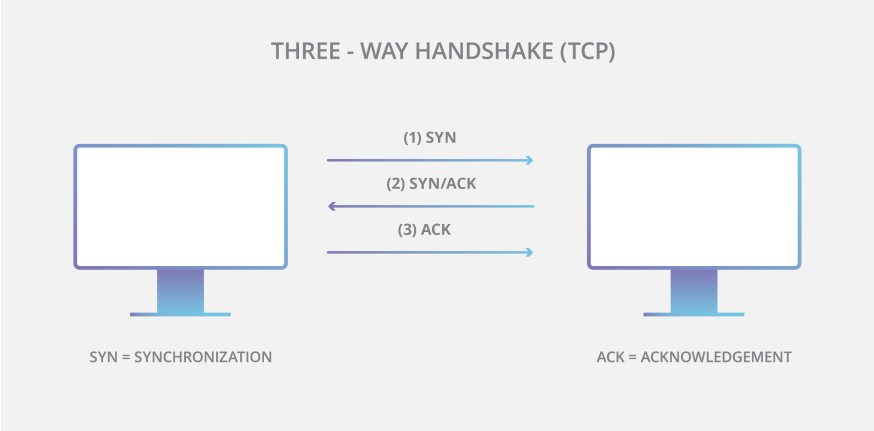

Figura 1: Diagrama Handshake TCP

A **confiabilidade** do TCP, garante que o receptor recerá dados não corrompidos, sem *gaps*, sem duplicação e em sequência.

O serviço de controle de fluxo serve para eliminar a possibilidade de *overflow* no *buffer* do receptor.

2.2 HTTP

O HTTP(*Hypertext Transfer Protocol,*) é um protocolo da camada de aplicação, que serve como base para o funcionamento da web.

O HTTP é implementado em dois programas: um cliente e um servidor. O cliente e o servidor funcionam em sistemas finais distintos e se comunicam através de mensagens HTTP. O protocolo HTTP é responsável então por definir como clientes requisitam páginas de web para o servidor e como o servidor transfere páginas web para clientes, como demonstrado na Figura 2. A comunicação é feita através do TCP, descrito na seção 2.1.

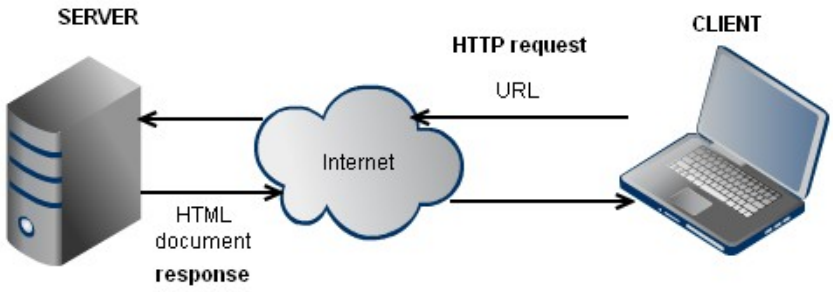

Figura 2: Diagrama comunicação HTTP

2.3 Proxy Server Web

Um servidor Proxy, também chamado de *web cache*, é uma entidade que atua como mediadora entre o cliente e o servidor em um processo de comunicação utilizando o HTTP.

Para funcionanmento do proxy, browser pode ser configurado de forma que todas as requisições HTTP sejam primeiro redirecionadas para o servidor proxy, antes de ir para o servidor final. O proxy então pode atuar tanto como cliente quanto servidor, recebendo e enviando requisições HTTP.

As duas principais funcionalidades de um proxy são: reduzir o tempo de resposta para uma requisição de um clientel e reduzir o tráfego de rede em um link de acesso de uma instituição.

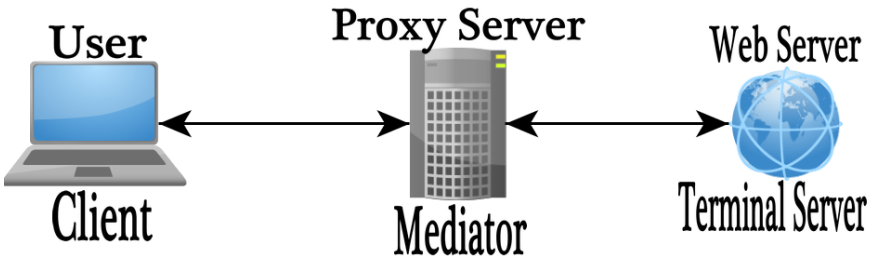

Figura 3: Proxy em uma comunicação HTTP

3 Arquitetura do sistema

A arquitetura do sistema foi dividida em quatro módulos: aracne.cpp, dumper.cpp, spider.cpp e proxy.cpp.

O módulo principal (aracne.cpp) é o módulo responsável pela função main, ou seja, é o módulo que rege o funcionamento geral do programa como um todo.

Proxy.cpp é o módulo encarregado pelo gerenciamento das funcinalidades de comunicação e criação de sockets, para o envio e recebimento de pacotes entre o cliente e o servidor, bem como é responsável pela implementação da funcionalidade de cache do servidor proxy.

Spider.cpp é o módulo que gerencia a funcionalidade spider, que consiste em- a partir de um número de níveis especificado pelo usuário- gerar uma árvore a partir de um código HTML, segunido qualquer referência que esse site tenha para outra página dentro dele mesmo.

O Dump.cpp tem funcionalidade muito parecida com o spider, no sentido de que esse módulo necessita de uma árvore de referências para poder funcionar. Seu funcionamento consiste em baixar todo o conteúdo indicado pelas referências armazenadas na árvore, gerando uma espécie de cópia offline do site requerido pelo cliente.

4 Código

A função main é responsável pela execução do programa principal. A criação e conexão dos sockets responsáveis pela comunicação entre o navegador e o proxy local é feita dentro dessa

função, chamando o método responsável por isso que está presente no módulo Proxy.cpp. Após isso, é iniciado o loop que mantém o programa sempre funcionando, gerando, cada vez, um menu no terminal para o usuário escolher o que deseja fazer tanto com o pedido enviado pelo navegador quanto pela resposta recebida do servidor.

Dentro desse loop principal está presente a lógica responsável por captar a requisição enviada pelo navegador, que é então passada por um parser que separa a informação mais relevante, como nome do host, url requirido e versão http desejada. Essa mesma requisição é salva em um arquivo txt. Esse parse é feito por meio de um dos métodos da classe proxy, chamado parseHttp().

Após criado o arquivo com a requisição original, o usuário pode escolher o que fazer com o mesmo por meio de uma escolha em um menu. Caso a opção selecionada seja editar a requisição, a função "system"é chamada para abrir o arquivo no qual a requisição original foi salva por meio do editor de texto "vim", salvando as alterações e enviando a requisição para o servidor por meio do método sendHttpRequest.

O método sendHttpRequest() é responsável por passar uma requisição para o servidor. A função incia criando um socket para a conexão entre o servidor e o proxy, por meio da função socket(). O endereço do servidor é obtido por meio do uso da função gethostbyname(name), onde name é obtido por meio de um dos atributos do objeto httpParsed. A requisição é então enviada por meio da função write(). A resposta do servidor é captada por meio de um loop while que tem como parâmetro de parada o retorno da função recv(). Essa resposta é armazenada em um arquivo de texto na pasta "responses".

Um segundo menu é apresentado para o usuário, do mesmo jeito que foi feito anteriormente. O usuário pode escolher entre mandar a resposta original para o navegador; editar a resposta e depois enviar a resposta editada para o navegador; realizar o spider ou o dump.

A classe spider recebe uma resposta de servidor e, dentro dessa resposta, procura no html da página qualquer referência a outras páginas dentro do mesmo domínio do site original e, após um tratamento de string, salva todos esses caminhos em uma pasta "spider"dentro de um arquivo de texto, que é usado para gerar mais requisições caso a árvore desejada tenha mais de um nível. Para cada nível que a árvore tem, os caminhos são colocados abaixo de seu link pai, de maneira indentada para que fique visível sua hierarquia.

A classe dumper

5 Funcionamento

Antes de executar o programa, o usuário deve configurar o navegador de sua preferência para fazer uma conexão proxy com o localhost na porta desejada. O programa, por padrão, seleciona a porta 8228, mas pode ser alterada quando o programa for executado. Com essa etapa feita, é necessário exucatar o comando "make", compilando todos os arquivos presentes na pasta do projeto (apenas os que passaram por mudanças desde a última compilação). Feito isso, o programa automaticamente criará os sockets necessários para a conexão entre o navegador e o localhost, aguardando apenas que o usuário faça uma requisição (por meio do navegador) de alguma página web com protocolo HTTP. Essa requisição é, então, salva em um arquivo txt nomeado de acordo com o site requisitado pelo navegador, substituindo quaisquer caracteres '/' pelo caractere '_' por motivos de lógica interna do programa.

O usuário receberá uma lista de opções referentes ao que pode ser feito com a requisição gerada pelo navegador, podendo escolher entre editar a requisição e depois enviá-la ou simplesmente enviá-la sem alteração. Caso escolha editá-la, o programa automaticamente abrirá o editor de texto "Vim"para permitir a edição do arquivo txt que contém a requisição gerada pelo navegador.

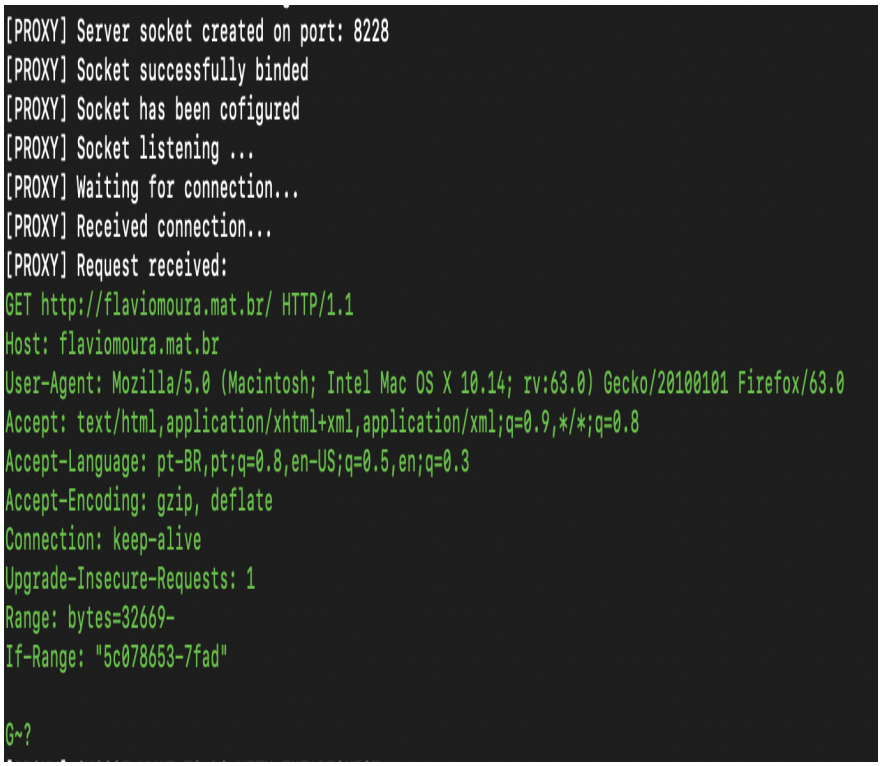

Figura 4: Requisição inteceptada pelo proxy

Após o usuário ter escolhido o que fazer com a requisição, esta é passada para o servidor, como se tivesse originado diretamente do navegador utilizado. Esse processo é possível por meio da criação de sockets para realizar a comunicação entre o proxy e o servidor desejado. O servidor retorna, então, os dados referentes à solicitação enviada. A resposta do servidor é armazenada em um arquivo txt, seguindo a lógica de nomeação implementada para salvar as requisições. Com as respostas do servidor devidamente salvas em arquivos, é possível implementar a função de cache, consistindo em, toda vez que uma requisição for feita, verificar se já existe uma resposta correspondente salva em um arquivo de texto e, caso haja, o conteúdo é simplesmente copiado deste para o usuário (ou navegador, dependendo da opção selecionada para o tratamento dessa resposta), caso contrário a requisição deve ser feita diretamente com o servidor. O menu de opções para tratamento da resposta é muito parecido com o menu da etapa anterior, consistindo na edição da resposta recebida seguida pelo envio da mesma; envio imediato da resposta; spider e dump.

O spider é um mecanismo capaz de pegar o html presente na resposta do servidor e vasculhá-lo por qualquer link (por meio do atributo href) que leve para outra página do mesmo domínio que foi especificado na requisição. Todos esses links são salvos em uma árvore (também em um arquivo de texto) e então são mostrados na tela do usuário.

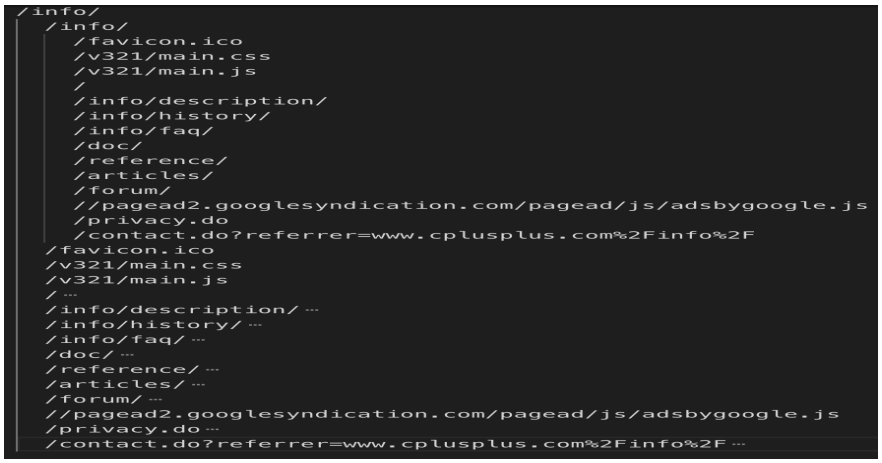

Figura 5: Resultado Spider

O dump é um processo complementar ao spider, consistindo em baixar o html presente em todos os links gerados a partir da árvore de caminhos gerada pelo spider. Esse processo é feito de maneira muito similar ao envio de requisições para o servidor, uma vez que os url's obtidos a partir do txt, gerado pelo spider, serão colocados em uma string, formando um header de requisição http.

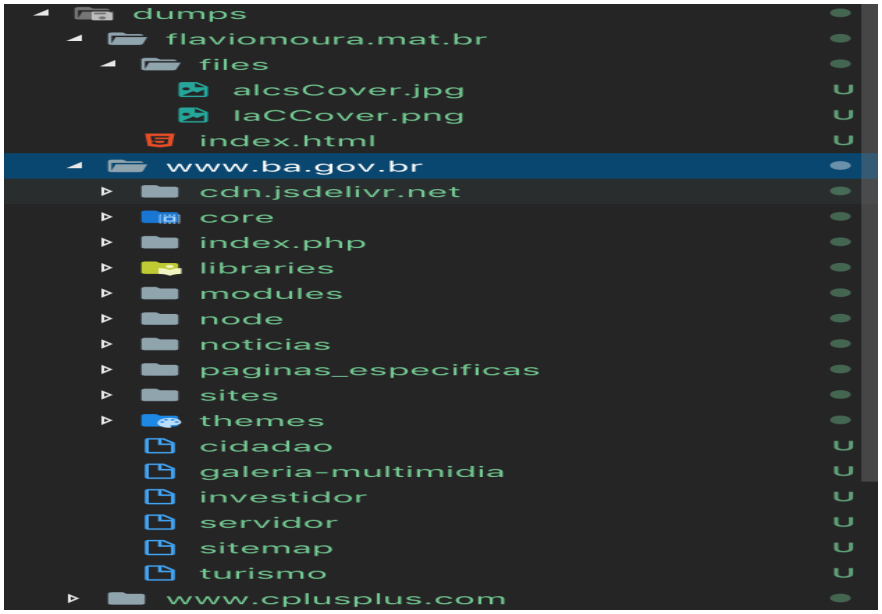

Figura 6: Resultado Dumper

Referências

[1] Kurose, James F., and Keith W. Ross. Computer networking: a top-down approach. Vol. 4. Boston: Addison Wesley, 2009.

[2] https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/

[3] https://tableless.com.br/criando-seu-proprio-servidor-http-do-zero-ou-quase-parte-i/