Relatório de redes - Inspetor HTTP + website dump

Gabriel Taumaturgo - 14/0140522
Andrey Emmanuel Matrosov Mazépas - 16/0112362

July 2019

1 Introdução

Este trabalho é uma aplicação dos conhecimentos adquiridos em sala em um software contido capaz de agir como uma proxy, uma aplicação que age como intermediário entre um ou mais hosts e servers, realizando requisições e repassando repsostas. Além da funcionalidade principal também foram parcialmente implementadas duas ferramentas capazes de realizar uma pesquisa em largura das urls de um certo dominio e realizar o download desses documentos em uma pasta especificada.

2 Ferramentas Utilizadas

Seguindo as especificações do projeto, o software foi desenvolvido em C++, utilizando a sockets API padrão e Qt para a interface gráfica e execução por threads dentro do sistema operacional Linux. Além das bibliotecas padrão da linguagem de programação, todas as outras funcionalidades e auxiliadores foram desenvolvidas especificamente para este trabalho. Foi acordado entre os desenvolvedores a utilização do Qt Creator como IDE/framework e a utilização do serviço de versionamento GitHub, o repositório contendo o código fonte é https://github.com/Andreymazepas/ArachnID O navegador utilizado para testar as funcionalidades foi o Mozilla Firefox versão 67.0.4

3 Estrutura Geral do Programa

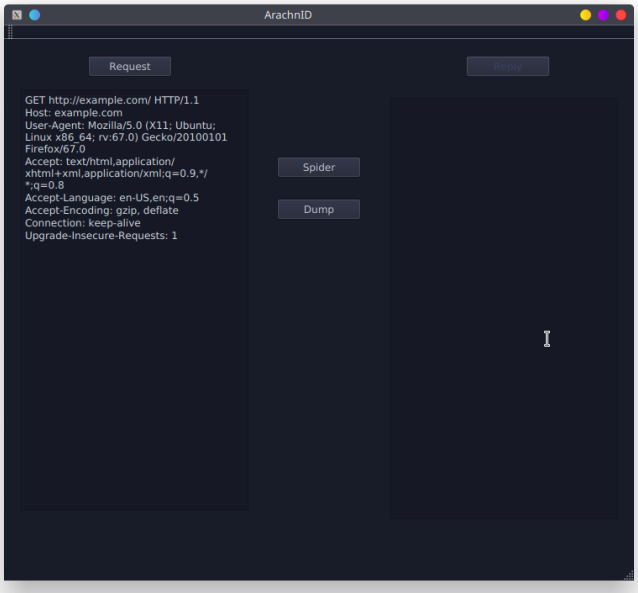
3.1 Arquitetura e Comunicação

A arquitetura do código não segue nenhum padrão de projeto especifico mas foi almejado o uso de boas práticas de C++ e Qt, utilizando a interface de signals e slots para comunicação entre elementos da interface gráfica e funções. Signals e slots são uma maneira de gerar comunicação entre classes.

Funciona assim: classes podem emitir um signal, por exemplo, a thread que recebe as requisições avisa a interface gráfica que uma nova requisição chegou, enquanto a classe da interface gráfica conectou um slot(uma função de callback) para ser executado quando aquele signal fosse emitido.

3.2 ProxyServer

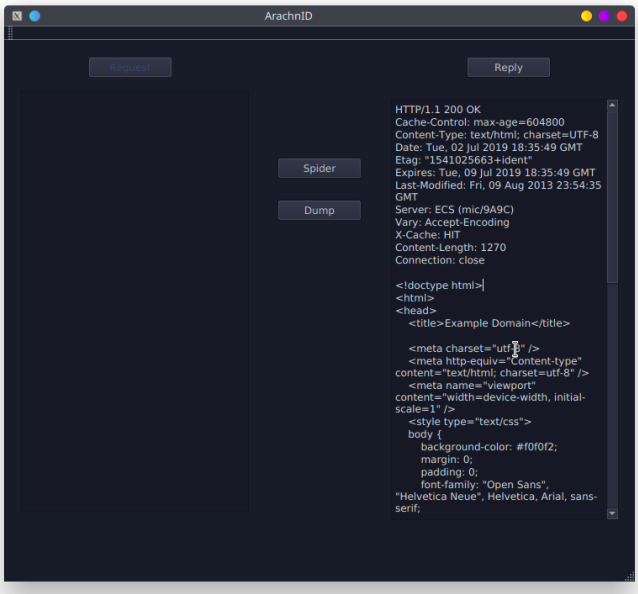
Para a funcionalidade principal de realizar inspeção do tráfego foi criada uma classe ProxyServer contendo em alto nível as funções que compõem o loop de escuta de requisições. Quando uma requisição chega, mandamos ela para que o usuário possa editar na classe MainWindow, emitindo um signal. Quando isso acontece a thread que executa o servidor se termina.



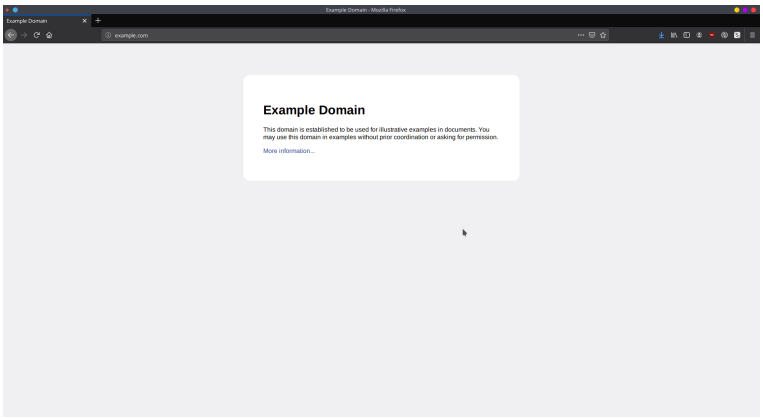
A execução volta quando a MainWindow manda um signal dizendo que o usuário terminou de editar o request e deseja o enviar para a internet. Isso desencadeia uma função de callback conectada a aquele sinal, então na prática, apesar da execução ter terminado é como se o programa estivesse bloqueado nesse evento.

Além disso, é nessa rotina que os headers de envio e resposta são parseados. O header de envio é parseado para que possamos detectar requests do tipo POST e CONNECT e recusá-los com um 501 NOT IMPLEMENTED, já que o programa só suporta requisições GET. Além disso, após parsear os headers, alguns campos são sobreescrevidos para simplicidade de implementação, como por exemplo: "connection", "accept-encoding", "range" e "if-range". O motivo

disso é para que o processamente da resposta, então por exemplo, a conexão sempre é fechada(nunca keep-alive), sempre são recebidos texto sem codificação alguma e nunca em chunks. Feita essa sobrescrita de campos, o pacote é enviado a internet e o processo fica bloqueado aguardando a resposta.

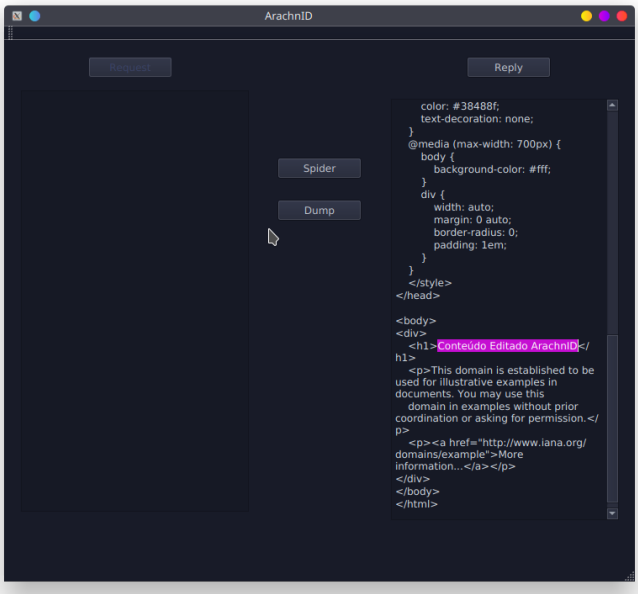


(a) Response original

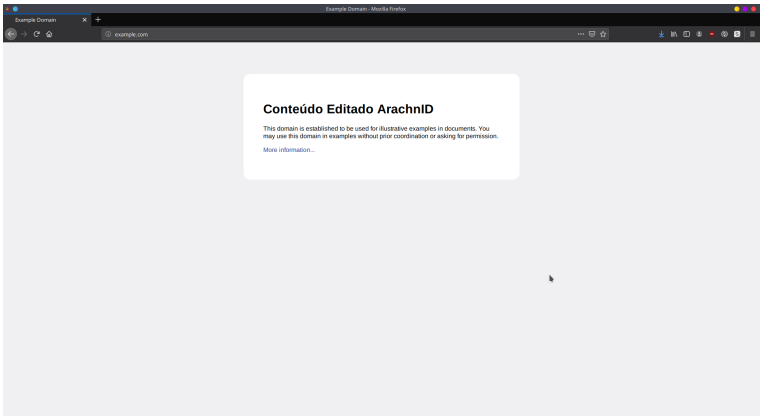


(b) Recebimento no Browser

Quando a resposta chega, apenas o header é lido, e em seguida parseado, para extrair os campos "content-type" e "content-length". "content-length" é usado para sabermos quantos bytes de payload esperar, e content type é usado para



(a) Response alterada



(b) Resultado no Browser

decidir se será enviado para a MainWindow o payload para ser editado também, já que payloads que não são textos não fazem muito sentido serem editados, como imagens ou PDFs. Então, a reposta que veio da internet (com payload ou não) é enviada para a MainWindow através de um signal. Assim como quando o request chegou do browser, após enviar o signal o fluxo de execução termina, e só volta quando a MainWindow emitir um signal dizendo que o usuário quer entregar a resposta ao browser. A resposta é então reconstituida se necessário

(ela é quebrada caso a mídia não seja texto para que apenas o header vá para edição do usuário) e a resposta é entregue ao browser, e as conexões fechadas, e o fluxo de execução volta para o inicio e fica bloqueado no accept esperando uma nova conexão.

3.3 Socket Utils

Há também uma classe SocketUtils que serve para reaproveitar e modularizar funções relacionadas a socket usada diversas vezes nos módulos do programa, entre as principais funções pode-se citar uma função que lê bytes até que leia uma sequência específica em sequência, por exemplo "\r \n \r \n", para que pudéssemos detectar o fim de um header e também a função de ler um número exato de bytes para um buffer, a implmentação dessa função era principalmente relevante nos casos de baixar arquivos grandes (imagens e pdfs), porque na função read, apesar de ser blocante até que algum byte chegue para ser lido, não há garantia que quando o primeiro byte chega todos os bytes estão lá, e isso acontecia com arquivos grandes (maiores que 16kb).

3.4 MainWindow

Essa é a classe da janela principal do programa. É uma classe "reativa", no sentido que sua tarefa é renderizer a UI e o código só é executado respondendo a acontecimentos como botões apertados ou signals emitidos por outras classes. Então entre as funcionalidades exercidas além da renderização estão: Criação inicial do processo que escuta as requisições(ProxyServer), abertura das janelas do spider e dump, assim como gerenciar quais botões ficam ativos(por exemplo, se não há request para ser enviado para a internet o botão que faz com que isso aconteça é desabilitado) e enviar signals para o ProxyServer.

3.5 HTTP Helper

Para facilitar a organização do código, funções auxiliares em relação a http utilizadas por outras classes foram definidas aqui. São apenas três funções que são utilizadas em todas as classes realizando requisições. A primeira é o simplify http header que altera alguns campos do request http, como o campo "connection" para "close" e removendo campos referentes a fragmentação como o "range" devido as limitações da proxy. Como retorno, ele transforma a request original em um map(string, string) para facilitar manipulação em outras classes. As outras duas funções tratam do analisador e construtor de headers, o build html header cria um header em plainText a partir da estrutura criada na função anterior e possibilita adicionar mais conteúdo. Já o leitor faz o processo inverso, transformando um header em plainText nas estruturas previamente mencionadas.

3.6 SpiderWindow

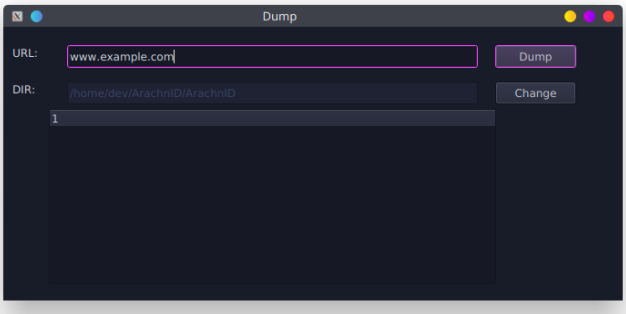
SpiderWindow é a classe da janela que o usuário interage para executar o spider, assim como a MainWindow, composta apenas de uma caixa de texto para exibir o resultado e um Input para a url aonde a exploração extensiva começará. Então é criada uma thread separada que executará esse processo e retornará o grafo formado pelos links. A partir dessa parte é que não foi tudo implementado, a ideia era retornar o grafo, isto é uma lista de listas de adjacência com os vértices sendo as urls exploradas, e isso ser exibido num campo de texto da janela do spider.

3.7 Spider

A classe spider é responsável por executar a exploração exaustiva do link. A funcionalidade não foi terminada, a suspeita é que havia um erro com o parsing para obter os links, de forma que isso fazia com que a exploração extensiva não terminasse mesmo filtrando os links que vão para fora daquele domínio, no entanto na apresentação do trabalho foi mostrado que se pusermos um número máximo de vértices explorados, conseguimos imprimir listas de adjacência consistentes, na maioria dos primeiros vértices, até que em algum momento um link 'errado' é enfileirado e os subsequentes também ficam errados.

A implementação foi feita realizando busca em largura nos links do site e só enfileirando os links do mesmo domínio, mantendo os links já visitados para evitar loop infinito. Para cada link, é gerado um header, ele é enviado ao host, e espera-se a resposta, em seguida ela é parseada e descobrimos os links do mesmo domínio e enfileiramos todos os ainda não enfileirados antes, até que a fila fique vazia.

3.8 DumpWindow



DumpWindow é a classe responsável pela renderização da UI da funcionalidade de Dump, apesar da funcionalidade não estar inteiramente implementada,

a janela já possui os slots e eventos necessários para a implementação. Uma caixa de texto que iria receber o output do processo de Dump e dois campos de Input, um para a entrada do site a ser realizado o Dump e outro para o diretório de armazenamento da função, sendo esse último acompanhado de um botão que abre o seletor de pastas do próprio sistema operacional para escolha. Por uma questão de tempo e problemas utilizando o seletor de pastas e arquivos, não foi possível implementar a funcionalidade, que consistiria basicamente do spider modificado para pegar todos os arquivos de uma certa url e seguir para as urls do mesmo domínio que estão linkadas.

4 Recursos

Sites usados para pesquisa:
`https://archive.codeplex.com/?p=tcpproxy`
`http://www.linuxhowtos.org/C_C++/socket.htm`
`https://www.binarytides.com/python-packet-sniffer-code-linux/`
`https://www.geeksforgeeks.org/socket-programming-cc/`
`http://www.dicas-l.com.br/arquivo/programando_socket_em_c++_sem_segredo.php`

Sites usados para teste:
`http://www.example.com`
`http://www.example.org`
`http://howto-pages.org/`