Aracne

Cesar Augusto de Carvalho
Departamento de Ciência da Computação
Universidade de Brasília
Matrícula: 15/0121814

Rafael Amaral Soares
Departamento de Engenharia Elétrica
Universidade de Brasília
Matrícula : 15/0145331

***Resumo*—Este documento tem por objetivo descrever a funcionalidade do trabalho Aracne desenvolvido na disciplina de Teleinformática e Redes 2. Em tal trabalho, foi empregado a teoria desenvolvida na displina à respeito do protocolo HTTP e programação com sockets em C/C++. O uso de proxy é uma prática comum em ISPs (Internet Service Providers) e diminuem considerávelmente o tráfego na rede e o tempo de espera do usuário. Porém ataques como man-in-the-middle podem ser feitos nessa topologia.**

***Index Terms*—HTTP, Proxy, Server, Inspetor, Spider, Dump.**

I. Introdução

Este projeto consiste em aplicar conhecimentos obtidos na disciplina de Teleinformática e Redes 2 para a implementação de um Proxy que intercepta requisições HTTP de um navegador web. Além do Proxy, também são implementadas as funções de geração de árvore hipertextual contendo as referências encontradas em uma página web e o dump de uma página, semelhante ao aplicativo *wget*.

II. Apresentação teórica

A. Proxy Server Web

Um servidor *proxy web* serve para atender requisições de clientes se passando por eles para resgatar recursos de outros servidores e, ao mesmo tempo, se passando por estes para responder à seus clientes. O uso do proxy melhora o tráfego na rede e ao mesmo tempo diminui o tempo de espera quando há a presença de *cache*. É interessante notar que nem todo o conteúdo da internet precisa estar presente num proxy (imagine a catástrofe que seria se algo do tipo fosse necessário) que conrém cache. O modo de operação é simples: o cliente se conecta ao proxy, que por sua vez analisa o conteúdo da requisição (manda a resposta imadiatamente caso possua cache) e possívelmente altera alguns campos. O proxy então abre uma conexão com o servidor de destino, resgata o conteúdo e transmite de volta ao cliente de acordo com sua política.

Nesse tipo de comunicação, é interessante notar que estamos literalmente colocando uma máquina entre um cliente e o servidor. Para o caso de um proxy transparante, não damos essa noção ao cliente e as requisições podem sofrer algum tipo de alteração sem que o cliente saiba. Nesse caso temos um conhecido ataque chamado *Man in the Middle*. Nele o atacante se passa como servidor alterando o conteúdo original abrindo espaço para roubo de informações sensíveis. Há técnicas para evitar tais situações que não serão abordadas aqui.

B. TCP

O TCP (*Transmission Control Protocol*) é um dos protocolos da camada de transporte da Internet. Ele atua sobre o protocolo IP (*Internet Protocol*), que na prática não garante nada (nem que os segmentos enviados chegarão ao seu destino, por exemplo), o chamado serviço de entrega de melhor esforço. Portanto, atua como ampliador dos serviços de entrega fornecidos pelo IP entre dois sistemas finais.

O protoclo TCP garante vários recursos adicionais à aplicação, o principal deles a transferência confiável de dados. Outros dos serviços providos são controle de congestionamento e verificação de erros.

O TCP é classificado como orientado à conexão, porque um processo de aplicação precisa se ”apresentar- enviar segmentos um ao outro para estabelecer parâmetros da transferência - antes de poder iniciar uma conexão.

Outra característica deste protocolo é conexão *Full-Duplex*, ou seja, uma vez estabelecida a conexão entre dois hospedeiros A e B, os dados podem ser transferidos tanto de A para B, quanto de B para A ao mesmo tempo. A conexão também é ponto-a-ponto, o que significa que a mesma tem sempre apenas um remetente e apenas um destinatário.

C. Protocolo HTTP

O HTTP/1.1 (*Hypertext Transfer Protocol*) é um protocolo padrão definido no RFC 7231 e difine a sintaxe das mensagens trocadas entre processos distintos. É um protocolo da camada de aplicação e para tal precisa do protocolo TCP para comunicação sobre um canal não confiável. O significado das mensagens trocadas é simples. Um exemplo de uma requisição GET ao site http://www.unb.br/ é mostrado abaixo:

```
GET http://www.unb.br/ HTTP/1.1
Host: www.unb.br
Connection: Keep-Alive
User-Agent: Chrome/70.0.3538.110
Accept: text/html
Accept-Encoding: gzip, deflate
```

No exemplo acima temos um GET não-condicional requisitando a página raiz do hospedeiro *www.unb.br*

utilizando a versão 1.1 do protocolo HTTP. Na segunda linha, temos o nome do hospedeiro (esse campo é necessário durante a análise da requisição feita pelo proxy, pois não se sabe à princípio onde o site está hospedado. Ele pode ser usado para fazer uma *query* DNS). O campo *Connection* diz ao hospedeiro para manter a conexão aberta (não mandar um FIN do protocolo TCP) após a transmissão do arquivo. Em seguida, temos o campo *User-Agent* que identifica o "agente que faz o trabalho"para o usuário (no momento que disparei a requisição estava usando o Chrome). Abaixo de *User-Agent* temos *Accept* que é o tipo de arquivo aceitável. Nesse caso, a requisição está esperando como resposta um html e depois temos o formato de codificação aceitável em *Accept-Encoding* que é o *gzip ou deflate*.
O servidor, por sua vez manda uma resposta:

HTTP/1.1 200 OK
Connection: Keep-Alive
Transfer-Encoding: chunked
Date: Mon, 10 Dec 2018 17:58:34 GMT
Content-Type: text/html; charset=utf-8
Server: nginx
Last-Modified: Mon, 10 Dec 2018 17:58:34 GMT

(data...data...data...)

Do exemplo acima podemos ver que a requisição foi bem sucedida (interpretado pela versão do protocolo HTTP/1.1 seguido de um 200 OK). O campo *Connection* diz que a conexão será mantida após a transmissão do arquivo. *Transfer-Encoding* diz que a stream de dados será dividida numa série de chunks. *Date* informa a hora do servidor na qual a resposta está sendo mandada de volta. *Content-Type* contém o formato do arquivo e sua codificação (ela será usada quando os bytes são lidos do socket pela aplicação). *Server* informa o nome do servidor que gerou a resposta e por último *Last-Modified* informa a data da última modificação daqueles dados. Em seguida temos uma linha em branco que marca o final do cabeçalho e o início da stream de dados.

O exemplo mostrado é simples, mas demonstra razoávelmente o que o HTTP faz. Há outros métodos nesse protocolo como o PUT, POST, DELETE, HEAD, mas não serão abordados aqui.

III. Arquitetura

O projeto foi realizado com a linguagem C++ 11 no sistema operacional Ubuntu. Os principais módulos e suas funcionalidades são descritos a seguir.

A. Request

A ideia principal de Request é realizar a requisição recebendo como parâmetro a própria requisição na forma de string. Ela cria um socket para isso, se conecta ao servidor, transmite a string recebida como parâmetro, recebe a requisição e retorna na forma de um vetor de *unsigned char* (o motivo de ser unsigned char é que estava ocorrendo alguns problemas na transmissão de imagens). A própria função se encarrega de buscar o endereço IP de destino dado o campo Host no cabeçalho HTTP. Não passe a requisição sem esse campo, pois certamente a requisição não será bem feita sem ele.

B. Receive

O módulo Receive tem por função principal criar um socket para escuta de requisições vindas do browser (proxy e inspector). Através dele é possível receber a requisição, tratá-la no caso do inspector, buscar no servidor de origem e mandá-la de volta ao browser.

C. Proxy

O proxy tem como objetivo abrir um socket, dependendo do valor da porta passada na execução do programa, receber as requisições vindas do browser, enviá-las imadiatamente ao servidor de origem abrindo outro socket, receber a requisição e mandá-la de volta ao browser. Um porém desse proxy é que ele está esperando uma reposta sempre do servidor de origem. Caso ele não mande, o proxy ficará escutando até que a conexão seja fechada e não algo do tipo 404 NOT FOUND. Nesse caso, uma string vazia será mandada de volta ao browser. Além disso, apenas métodos GET são possíveis de serem feitos.

D. Inspetor

Nosso Inspector possui uma funcionalidade semelhante ao do proxy, porém podemos editar as requisições vindas do browser antes de enviá-la ao servidor de origem assim como sua reposta. A requisição vem do browser através do mecanismo explicado acima, é editada de acordo com o gosto do usuário (lembre de deixar o campo Host para que o módulo Request funcione corretamente), passada para o servidor de origem que retorna um conteúdo que também é editado pelo usuário e então repassado ao browser (é a ideia semelhante do ataque *Man in the Middle*).

E. Spider

Recebe uma URL e analisa todas as referências subjacentes a partir da mesma. O usuário tem a opção de digitar quantos níveis de referências deseja buscar. O tempo de execução aumenta muito rápido de acordo com o número de níveis. Ao fim da execução é gerada uma árvore hipertextual contendo todas as referências encontradas e seus respectivos níveis.

F. Dump

Tem como entrada a saída do Spider e realiza o download para a máquina local de todos os arquivos dos níveis especificados. Em cada HTML é necessário realizar a correção das referências (src, href e url) para que ela possa ser visualizado como se estivesse na máquina. Algumas referências são dinâmicas. Infelizmente essas não puderam ser tratadas e corrigidas.

IV. Funcionalidades implementadas e uso

Para rodar o programa, basta utilizar o comando *make* no diretório do *Makefile*. Em seguida, rodar o executável *aracne* que foi gerado, podendo passar a porta que será utilizada. Por exemplo:

./aracne -p 8000

Caso não seja passada nenhuma porta, a porta padrão utilizada é 8228.

Ao inicializar o programa, é exibida a tela da Figura 1.

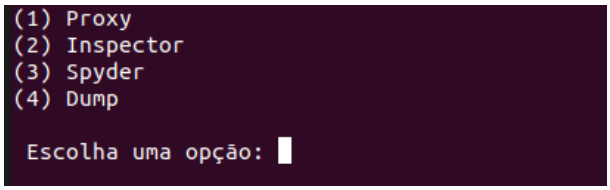

Figura 1. Tela inicial do programa.

O usuário então escolhe qual serviço deseja executar.

A. Proxy

Para utilização do proxy, é recomendado o navegador *Mozilla Firefox*, por possuir maior facilidade para configuração.

Basta ir em "Preferências → Configurações de conexão → Configuração manual do proxy" e preencher como na Figura 2.

Figura 2. Configuração do proxy no Mozilla Firefox.

Em nosso programa, ao escolher a opção do Proxy, o mesmo já estará interceptando as requisições de maneira transparente ao usuário, carregando o site normalmente como se não houvesse o proxy intermediário. A desvantagem é que pode ser que demore um pouco mais que o comum para carregar todo o site, mas é algo que passaria desapecebido da maioria dos usuários.

B. Inspector

O inspector é utilizado da mesma maneira que o *proxy*, porém é exibido para o usuário a cada requisição o texto com possibilidade de edição. Para isso, foi utilizado o editor *nano* através do terminal em que o programa roda. Um exemplo de *request* é mostrado na figura 3 e um de *response* na figura 4.

Figura 3. Exemplo de *request* com possibilidade de edição.

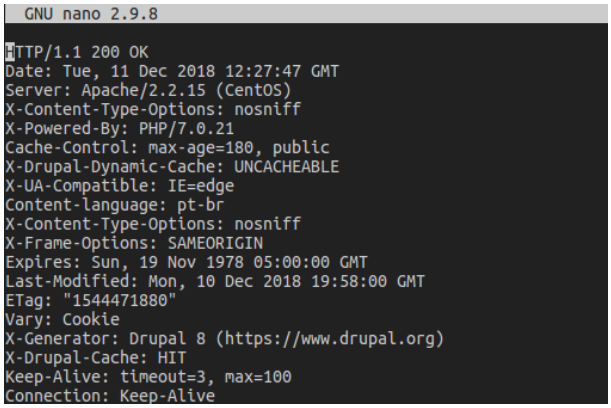

Figura 4. Exemplo de *response* com possibilidade de edição.

C. Spider

Ao executar o programa escolhendo a opção 3, do spider, temos uma tela como a mostrada na Figura 5. O usuário deve digitar qual domínio deseja buscar e em seguida escolher quantos níveis a busca terá. Após a execução, é gerada uma árvore, que tem sua estrutura exibida no terminal, conforme mostrado na Figura 6, e também é salva em um arquivo local chamado *arvore_hipertextual.txt*, para melhor visualização.

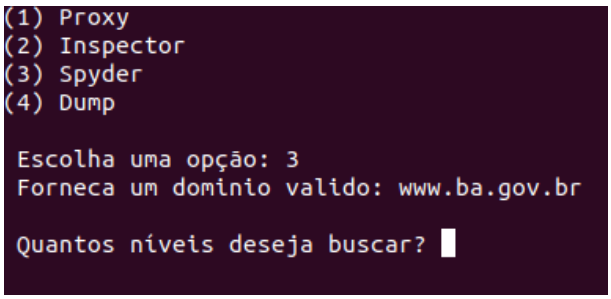

Figura 5. Tela do spider.

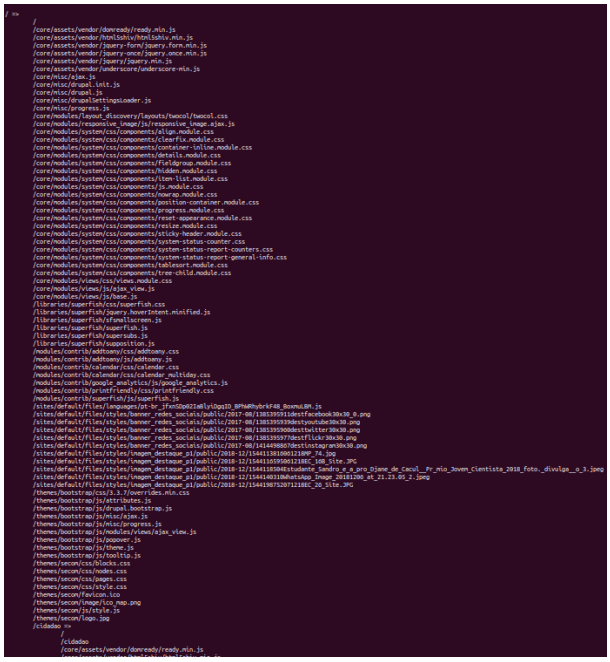

Figura 6. Parte da árvore hipertextual para o site www.ba.gov.br.

D. Dump

Na execução da função Dump, o Spider é feito antes da transferência dos arquivos. Após a execução, os arquivos são baixados e as referências dentro dos htmls são corrigidas para que o conteúdo possa ser visto como se estivesse na máquina local. A figura 7 ilustra o funcionamento de Dump para o site *www.ba.gov.br*.

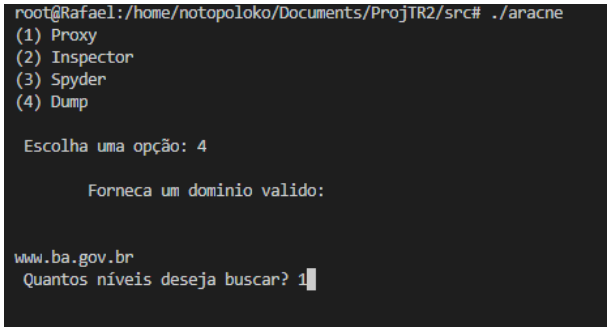

Figura 7. Tela do dump.

Em seguida são feitas requisições GET no servidor de origem. Cada .html é colocado na pasta raiz e o restante nas respectivas pastas como mostrado na figura 8.

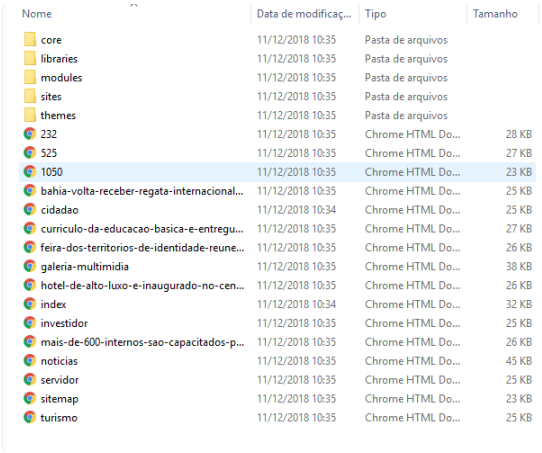

Figura 8. Repositório após a execução do dump.

Em cada html, as referências são corrigidas. Quando a página raiz (resolvemos colocar o nome de index.html) é aberta no browser, vemos que algumas refências não foram resolvidas. Isso ocorre porque algumas aparecem dinamicamente e nesse caso não podemos inferí-las. A figura 9 mostra o resultado de dump aberto no browser. Repare a URL na barra de endereço. Ela realmente aponta para a máquina local.

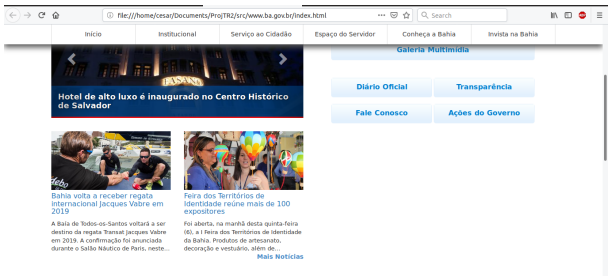

Figura 9. Página index.html mostrada após a execução do dump.

É bem próximo ao que seria caso não estivesse na máquina local.

V. Documentação do código

A documentação foi feita através do software Doxygen, gerando um PDF e um HTML com as funções do código, descrição, parâmetros e retornos. O PDF foi anexado ao fim deste documento, nas páginas seguintes, e o HTML pode ser encontrado no diretório do código-fonte.

Código disponível em: https://github.com/notopoloko/ProjTR2.

Referências

[1] KUROSE, J. F. e ROSS, K. - Redes de Computadores e a Internet - uma abordagem top-down - 6ª Ed., Pearson, 2013.

Projeto de TR2 - Aracne

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 dump.cpp File Reference

```
#include "connection.hpp"
```

Functions

- void makeDump (string baseURL)
- int dump (set< string > requests, string baseURL)
- void generateMap (map< string, string > &mapRefs, set< string > &requests, string baseURL)
- string cutHead (string serverRequest)
- void fixRefs (string &serverResponse, map< string, string > &mapRefs)

2.1.1 Function Documentation

2.1.1.1 cutHead()

```
string cutHead (
            string serverRequest )
```

Função que corta o cabeçalho e retorna a stream de dados da resposta do servidor.

Parameters

serverRequest	String que contém a resposta vinda do servidor.

Returns

string: Uma string com os dados.

Definition at line 127 of file dump.cpp.

2.1.1.2 dump()

```
int dump (
            set< string > requests,
            string baseURL )
```

Função reliza de fato o resgate das requisições passadas como parâmetro

Parameters

requests	Set com requisições que deve ser feitas em baseURL.
baseURL	Endereço base onde as requisições serão feitas. Ex: www.ba.gov.br.

Returns

int: Um inteiro indicando sucesso das requisições feitas. Ele pode ser ignorado.

Definition at line 15 of file dump.cpp.

2.1.1.3 fixRefs()

```
void fixRefs (
            string & serverResponse,
            map< string, string > & mapRefs )
```

Função que corrige as referências no HTML apontando para o local correto.

Parameters

&serverResponse	Referência à uma string que contém a reposta do servidor (deve ser um HTML).
&mapRefs	Referência à um Map que contêm as referências corretas.

Definition at line 143 of file dump.cpp.

2.1.1.4 generateMap()

```
void generateMap (
            map< string, string > & mapRefs,
            set< string > & requests,
            string baseURL )
```

Função que realiza o mapeamento das referências para as referências corretas que serão colocadas no html depois de baixado.

Parameters

&mapRefs	Referência para um Map com requisições que devem ser mapeadas.
&requests	Referência para um Set que contêm as requisições que devem ser feitas em baseURL.
baseURL	Endereço base onde as requisições serão feitas. Ex: www.unb.br.

Definition at line 90 of file dump.cpp.

2.1.1.5 makeDump()

```
void makeDump (
            string baseURL )
```

Função base de Dump. Ela reliza um spider seguido de um dump. O resultado é colocado na pasta raiz dentro de uma pasta com o nome passado como parâmtro

Parameters

baseURL	URL base do domínio desejado.

Definition at line 5 of file dump.cpp.

2.2 inspector.cpp File Reference

```
#include "connection.hpp"
```

Macros

- #define MAXRCVLEN 2000

Functions

- int inspector (int PORTNUM)
- std::vector< unsigned char > readBinaryFile (string filename)
- std::string readTextFile (string path)
- bool writeFile (string path, vector< unsigned char > dados)

Variables

- struct freeMemoryList fml

2.2.1 Macro Definition Documentation

2.2.1.1 MAXRCVLEN

```
#define MAXRCVLEN 2000
```

Definition at line 3 of file inspector.cpp.

2.2.2 Function Documentation

2.2.2.1 inspector()

```
int inspector (
            int PORTNUM )
```

Função que raliza inspeção da requisição vinda do browser e da resposta do servidor

Parameters

PORTNUM	Número da porta que será aberta para receber requisições vindas do browser.

Returns

int: Um inteiro indicando sucesso das requisições feitas. Ele pode ser ignorado.

Definition at line 8 of file inspector.cpp.

2.2.2.2 readBinaryFile()

```
std::vector<unsigned char> readBinaryFile (
            string filename )
```

Função que faz a leitura do binária do arquivo passada como parâmetro

Parameters

filename	Uma string que apontando o arquivo à ser lido

Returns

std::vector<unsigned char>: Um vetor de unsigned char com o bytes lidos do arquivo.

Definition at line 80 of file inspector.cpp.

2.2.2.3 readTextFile()

```
std::string readTextFile (
            string path )
```

Função que faz a leitura em modo de texto do arquivo passada como parâmetro.

Parameters

path	Uma string que apontando o arquivo à ser lido.

Returns

std::string: Um string contendo o conteúdo do arquivo.

Definition at line 113 of file inspector.cpp.

2.2.2.4 writeFile()

```
bool writeFile (
            string path,
            vector< unsigned char > dados )
```

Função que faz a escrita dos dados passados como parâmetro em path.

Parameters

path	Uma string que apontando o arquivo à ser escrito.
dados	Um vector de unsigned char com os dados à serem escritos no arquivo.

Returns

bool: Um booleano indicando sucesso na escrito arquivo.

Definition at line 127 of file inspector.cpp.

2.2.3 Variable Documentation

2.2.3.1 fml

```
struct freeMemoryList fml
```

Definition at line 5 of file inspector.cpp.

2.3 main.cpp File Reference

```
#include "connection.hpp"
```

Functions

- int main (int argc, char ∗argv[ ])

2.3.1 Function Documentation

2.3.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Função principal do projeto e que liga todos os módulos de acordo com a opção escolhida pelo usuário

Parameters

argc	
argv	

Returns

int

Definition at line 5 of file main.cpp.

2.4 proxy.cpp File Reference

```
#include "connection.hpp"
```

Macros

- #define MAXRCVLEN 2000

Functions

- int proxy (int PORTNUM)

2.4.1 Macro Definition Documentation

2.4.1.1 MAXRCVLEN

```
#define MAXRCVLEN 2000
```

Definition at line 3 of file proxy.cpp.

2.4.2 Function Documentation

2.4.2.1 proxy()

```
int proxy (
            int PORTNUM )
```

Função do proxy que recebe as requisições vindas do browser e passa imediatamente para o socket ligado ao servidor sem a possibilidade de editar, assim como sua resposta.

Parameters

PORTNUM	Um inteiro dizendo qual porta será aberta para receber requisições do browser.

Returns

int: Um inteiro indicando sucesso na execução do proxy.

Definition at line 7 of file proxy.cpp.

2.5 receive.cpp File Reference

```
#include "connection.hpp"
```

Functions

- int createNewSocket (uint16_t portNum, uint16_t parallelConnections)
- void freeMemory ()

2.5.1 Function Documentation

2.5.1.1 createNewSocket()

```
int createNewSocket (
            uint16_t portNum,
            uint16_t parallelConnections )
```

Função que cria um socket para escutar as requisições vindas do browser.

Parameters

portNum	Porta que será aberta para receber requisições.
parallelConnections	Número de conexões em paralelo que podem ser recebidas do browser

Returns

int: Um inteiro com o descriptor do socket.

Definition at line 3 of file receive.cpp.

2.5.1.2 freeMemory()

```
void freeMemory ( )
```

Função que libera a memória após a execução do programa evitando memory leak.

Definition at line 41 of file receive.cpp.

2.6 request.cpp File Reference

```
#include "connection.hpp"
```

Functions

- vector< unsigned char > makeRequest (std::string msg_string)
- std::string getHostValue (std::string msg_string)

2.6.1 Function Documentation

2.6.1.1 getHostValue()

```
std::string getHostValue (
            std::string msg_string )
```

Função que resgata o valor do campo Host para fazer um query DNS em busca do IP de origem. O campo Host precisa ser passado na requisição HTTP.

Parameters

msg_string	Uma string que será inpecionada para extrair o valor de Host.

Returns

std::string: Uma string que contém o valor do campo.

Definition at line 51 of file request.cpp.

2.6.1.2 makeRequest()

```
vector<unsigned char> makeRequest (
            std::string msg_string )
```

Função que realiza a requisição no servidor de origem.

Parameters

msg_string	Uma string que será passada ao socket para enviar ao servidor.

Returns

vector <unsigned char>="">: Um vector de unsigned char com a resposta recebida.

Definition at line 5 of file request.cpp.

2.7 spyder.cpp File Reference

```
#include "connection.hpp"
#include "spyder.hpp"
```

Functions

- set< string > spyder (string baseURL)
- void buildReference (set< string > &result, string response, string baseURL)
- bool isHTML (string url, string baseURL)
- bool isReallyHTML (string url, string baseURL)
- set< string > searchChildren (string url, string baseURL)
- Tree generateTree (string baseURL, int levels)

2.7.1 Function Documentation

2.7.1.1 buildReference()

```
void buildReference (
            set< string > & result,
            string response,
            string baseURL )
```

<Insere em result os arquivos/diretórios encontrados em response>

Parameters

result	Endereço do set onde serão inseridas as referências.
response	resposta obtida do request.
baseURL	URL base do domínio desejado.

Definition at line 26 of file spyder.cpp.

2.7.1.2 generateTree()

```
Tree generateTree (
            string baseURL,
            int levels )
```

Gera a árvore hipertextual.

Parameters

baseURL	URL base do domínio desejado.
levels	Número de níveis máximos desejados para a árvore..

Returns

Tree: Árvore gerada.

Definition at line 204 of file spyder.cpp.

2.7.1.3 isHTML()

```
bool isHTML (
             string url,
             string baseURL )
```

Inspeciona um cabeçalho para saber se um caminho é HTML para saber se deve ser inspecionado.

Parameters

url	Url da referência.
baseURL	URL base do domínio desejado.

Returns

bool: Indica se é ou não HTML.

Definition at line 129 of file spyder.cpp.

2.7.1.4 isReallyHTML()

```
bool isReallyHTML (
             string url,
             string baseURL )
```

<Verifica se o cabeçalho da url já foi inspecionado para retornar a informação se a url é ou não um HTML.>

Parameters

url	Url da referência.
baseURL	URL base do domínio desejado.

Returns

bool: Indica se é ou não HTML.

Definition at line 157 of file spyder.cpp.

2.7.1.5 searchChildren()

```
set<string> searchChildren (
            string url,
            string baseURL )
```

<Busca pelas referências um nível exatamente abaixo da url.>

Parameters

url	Url que será buscada.
baseURL	URL base do domínio desejado.

Returns

set<string>: Conjunto de referências encontradas na url.

Definition at line 177 of file spyder.cpp.

2.7.1.6 spyder()

```
set<string> spyder (
            string baseURL )
```

Função principal do spyder.

Parameters

baseURL	URL base do domínio desejado.

Returns

set<string>: Set com os nomes das referências encontradas.

Definition at line 6 of file spyder.cpp.

Index