

UNIVERSIDADE DE BRASÍLIA
TELEINFORMÁTICA E REDES 2
TURMA A

RELATÓRIO - TRABALHO FINAL
IMPLEMENTAÇÃO DE UM INSPETOR
HTTP
BASEADO EM SERVIDOR PROXY

1º/2018
Rafael Chehab - 15/0045123
Davi Freitas - 15/0033010

Introdução

O seguinte trabalho tem como principal objetivo a aplicação dos conhecimentos relacionados à arquitetura de camadas de redes de computadores para a construção de um inspetor de tráfego do protocolo web HTTP, por meio de um servidor Proxy. Além disso, procura-se aplicar os conhecimentos da camada de aplicação e do protocolo HTTP para executar outras duas funcionalidades: o **spider** é uma funcionalidade que permite a listagem de todas URLs subjacentes a uma dada URL; e o **cliente recursivo**, que como um aplicativo *wget*, permite o *dump* de todo o conteúdo (a partir de uma dada url) localmente no sistema cliente.

Resumo Teórico

Proxy Server Web

A aplicação Web consiste numa arquitetura cliente-servidor que possui um hospedeiro sempre em funcionamento para atender requisições de outros inúmeros hospedeiros. Esse hospedeiro que está sempre funcionando para responder a requisições é chamado de servidor. Os hospedeiros que realizam requisições ao servidor são chamados de clientes.

Assim, um servidor Proxy é uma entidade de rede que atua como um intermediário entre o servidor e seus clientes (fig. 1). Assim, o proxy pode atender a requisições dos clientes em nome do servidor Web de origem, podendo ser utilizado com três objetivos principais:

1. Compartilhamento de uma conexão com a internet com apenas um IP disponível (com o proxy conectado à internet e os clientes ligados a ele fazendo requisições por ele);
2. Implementação de uma cache *web*, com o proxy armazenando páginas, arquivos e outros objetos acessados mais recente e/ou frequentemente. Isso pode beneficiar tanto os clientes, por meio de um tempo de resposta menor, como também o servidor, pela diminuição de requisições (e, consequentemente, de tráfego) para ele.
3. Listagem de sites, palavras e outros objetos para serem "bloqueados" pelo proxy, funcionando como uma espécie de filtro. Essa funcionalidade pode ser utilizada para controle parental, medidas de segurança, entre outros.

Essas funcionalidades oferecidas por proxys permitiram a popularização dessas entidades na aplicação Web. Sua facilidade de uso, uma vez que não é necessário um sistema dedicado à



Figura 1: Diagrama representando o funcionamento de um servidor proxy

realização dessas funções (mas sim apenas de um programa especializado), também contribuiu para a popularização dos proxys - principalmente os proxys web.

TCP

O TCP (Transmission Control Protocol) é um protocolo da camada de transporte que define como estabelecer and gerenciar uma conexão para que programas de aplicações possam trocar dados. Por ser uma **arquitetura de rede**, ele fornece um conjunto específico de serviços para o desenvolvedor da aplicação, com uma arquitetura fixa.

Assim, o modelo de serviço TCP possui uma orientação à conexão - troca de informações de controle cliente-servidor antes do começo do fluxo de dados entre os dois sistemas - e fornece um serviço confiável de transporte. Dessa forma, os sistemas finais da rede não precisam se preocupar no envio de dados em ordem e sem erros, uma vez que o TCP já fornece esse serviço.

Protocolo HTTP

O HTTP é um protocolo de camada de aplicação, definindo como as mensagens entre dois **processos** de uma aplicação devem se comunicar estando em sistemas diferentes.

O HTTP é um **protocolo** de camada de aplicação da Web, caracterizando-se como apenas uma parte da aplicação Web. Por meio de trocas de mensagens HTTP (requisições de objetos por parte do cliente e respostas por parte do servidor), dois processos de aplicação conseguem se comunicar em sistemas finais diferentes.

Esses objetos requisitados pelos clientes são caracterizados por um arquivo-base HTML e outros objetos referenciados por esse arquivo, como imagens, formulários, entre outros. O conjunto desses objetos compõem uma página Web.

Uma mensagem de requisição HTTP é aquela enviada do cliente para o servidor, e é escrita em ASCII (o corpo da requisição pode também ser escrito em binário). Como pode ser visto

na fig. 2, uma requisição HTTP é composto por uma linha de requisição, linhas de cabeçalho e o corpo da entidade.

- Uma linha de cabeçalho tem seu formato fixo, sempre na primeira linha da requisição. Ela é composta pelo método da requisição (que indica que tipo de resposta o cliente irá querer do servidor), uma URL (que indica o objeto que o cliente quer, no caso do método GET) e a versão do protocolo.
- As linhas de cabeçalho possuem tamanho variável de acordo com a requisição, apresentando informações valiosas para o processamento tanto do proxy quanto do servidor para o fornecimento de uma resposta adequada.
- Por último, o corpo da entidade na requisição é utilizado para métodos do tipo POST, servindo como auxiliar no preenchimento de um formulário, por exemplo.

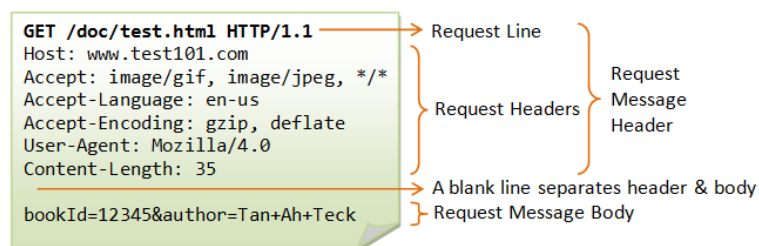


Figura 2: Exemplo de requisição HTTP

Um exemplo de uma resposta HTTP - por parte do servidor/proxy - pode ser observada na fig. 3. Uma resposta HTTP é composta por uma linha de estado, um conjunto de linhas de cabeçalho e o corpo da entidade.

- Uma linha de estado é sempre fixa na primeira linha da resposta, tal como a linha de requisição funciona na requisição. Seus campos são a versão do protocolo, um código de estado - que indica ao cliente o resultado da requisição enviada por ele - e uma mensagem de estado para acompanhar o código de estado.
- As linhas de cabeçalho da resposta são análogas às vistas na requisição, possuindo um tamanho variável.
- O corpo da entidade na resposta contém o objeto pedido pelo cliente.



Figura 3: Exemplo de resposta HTTP

Programação com Sockets

Um socket pode ser considerado como a interface entre a camada de aplicação e a camada de transporte, dentro de um sistema final. O socket permite especificar o processo num sistema final por meio de um identificador, chamado número de porta. Assim, a programação com sockets permite a conexão de dois sistemas finais diferentes a nível de aplicação, podendo ser configurada diretamente por software. O processo de comunicação por sockets funciona como visto na fig. 4. Pode-se ver, por essa figura, que o servidor precisa de alguns passos a mais para o estabelecimento de uma conexão com um cliente. Serão abordados brevemente o que cada um desses métodos é responsável no estabelecimento dessa conexão:

- `socket()` - Método responsável pela criação de um socket, retornando um inteiro com o identificador do socket (tanto para o cliente quanto para o servidor);
- `setsockopt()` - Opcional, ajuda na manipulação de opções para o socket referido pelo identificador obtido por `socket()`;
- `bind()` - O método `bind()` faz uma ligação entre o socket e o endereço e número de porta especificados.
- `listen()` - Coloca o servidor no modo de escuta, de forma a esperar a chegada do cliente para conexão;
- `accept()` - Utilizada pelo servidor para extrair a requisição da primeira conexão na lista de clientes pendentes;
- `connect()` - Utilizada pelo cliente para se conectar ao socket servidor
- `send/recv()` - Métodos utilizados para a troca de dados, como o próprio nome já infere

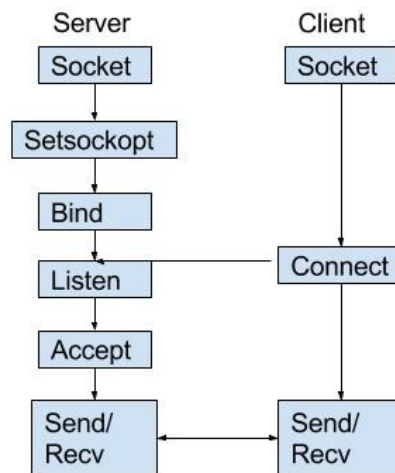


Figura 4: Esquema da conexão na programação de sockets

Arquitetura e Documentação

Spider e Cliente Recursivo

As funcionalidades do spider e do cliente recursivo (dump), possuem várias similaridades, o que levou a decisão de projeto de unir a implementação básica desses 2 serviços, porém disponibilizando uma interface externa diferente para cada um.

A implementação desses serviços foi dividida em 3 partes:

1. Os arquivos *src/spider.cpp* e *include/spider.h* contêm uma série de funções básicas para manipular URLs, enviar um request HTTP mínimo e receber responses HTTP, além de uma manipulação simples das mensagens.
2. Os arquivos *src/html_parser.cpp* e *include/html_parser.h* contêm funções para obter referências hypertext a partir de uma página HTML obtida pelo *spider.cpp*.
3. Os arquivos *src/spider_web.cpp* e *include/spider_web.h* contêm funções para criar uma árvore de URLs e para a partir de uma URL inicial construir a árvore de URLs subjacentes, permitindo o download das URLs encontradas. Além disso, esses arquivos possuem as interfaces externas *spider()* e *dump()*.

spider.cpp

A parte do *spider.cpp* possui como funções principais: criar uma abstração do protocolo

HTTP e da comunicação utilizando a rede e facilitar a separação de uma url base e do caminho. Suas 2 funções comumente utilizadas pelas outras partes são:

- *spider_get_data(char *url)*

Essa função cria uma mensagem de requisição mínima HTTP. Em seguida, separa obtém a url base, o caminho e a porta e utiliza os serviços do DNS para descobrir o endereço do servidor e estabelecer uma conexão TCP com ele.

Estabelecida a conexão, envia-se a requisição previamente criada e recebe-se a resposta. Para finalizar, retira-se o cabeçalho HTTP e retorna-se somente a parte de dados.

Utilizando essa função consegue obter os arquivos requisitados pelo cliente recursivo, individualmente.

- *spider_separate_url_path(char *url, size_t *url_size, size_t *path_size)*

Essa função recebe uma url que inclui a url base e o caminho e a separa, alterando também a variável que indica o tamanho delas.

Durante as várias etapas dos spider e do cliente recursivo são necessárias constantemente manipulações de urls, ora com a url completa, ora com a url separada do caminho. Assim, essa função facilita essa alternância.

html_parser.cpp

Após a obtenção do arquivo requisitado através de *spider_get_data(char *url)*, caso esse arquivo seja um HTML, é necessário uma verificação das referências contidas nele para uso na geração de URLs subjacentes e na recursão do dump.

O parser do HTML contém uma lista dos atributos, de certas tags, que possuem como valor uma URL. Utilizando essa lista, ele analisa todas as tags e todos os atributos delas. Sempre que o par tag, atributo estiver na lista, foi encontrada uma referência, a qual é dividida em url base e caminho e colocada em um vetor, o qual é retornado ao final da função.

Além disso, essa parte possui uma interface, a qual inclui a obtenção do arquivo requisitado.

spider_web.cpp

Essa parte contém a estrutura de dados utilizada para fazer a recursão nas URLs e para manter a árvore de URLs subjacentes.

A estrutura de dados utilizada é um grafo, uma vez que toda árvore é um grafo e, por isso, essa representação contém tudo necessário para a criação da árvore e pode conter mais informações, como por exemplo duas páginas que se referenciam mutuamente.

Cada nó do grafo possui o seu caminho a partir da url base e ponteiros para seus vizinhos. Além disso, criou-se uma estrutura de controle chamada web que contém todos os nós do grafo, além de um nó central e nós de borda.

O nó central é o nó a partir do qual iniciou-se o spider ou o cliente recursivo. Os nós de borda são nós que foram referenciados por algum nó obtido, porém ainda não foram baixados, ou seja, ainda é necessário buscar por referências nos nós de borda.

Para baixar o dump, utiliza-se uma opção na manipulação da web, a qual faz com que no *html_parser.cpp*, após a obtenção do response porém antes do parse do HTML, salve-se o arquivo.

Todos os arquivos são salvos na pasta */src/files*. Em seguida, há um diretório com o nome da url base e dentro desse diretório seguem-se os caminhos.

Para imprimir o spider, passa-se por todos os nós da árvore dividindo os caminhos com o delimitador */* para criar uma estrutura de diretório com a organização de uma árvore. Em seguida, imprime-se essa árvore pela política dfs (depth first search).

Inspetor com Proxy

O servidor proxy implementado nesse projeto funciona conforme a fig. 1, funcionando como intermediário entre o cliente e o servidor. Apesar de não realizar caching ou filtragem de conteúdo, esse servidor proxy realiza a inspeção de cabeçalho, permitindo a alteração de campos de cabeçalho ao interceptar uma requisição. Para o lado da resposta que vem do servidor, o servidor transmite diretamente ao cliente.

As funcionalidades do proxy foram divididas nos seguintes arquivos:

ProxyParser.h/cpp

Essa biblioteca contém as funções que fazem o *parsing* da string de requisição obtida do browser para estruturas divididas em campos, assim como também faz o contrário *unparsing*. O parsing da requisição pode ser encontrado no método *int RequestFields_parse(RequestFields *parse, const char *buf, int buflen)*.

O método *RequestFields_parse* transforma a string de requisição do browser numa struct chamada *RequestFields*, o que permite realizar a inspeção do cabeçalho:

RequestFields

- *char *method* - método da requisição;
- *char *protocol* - protocolo (deve ser HTTP, mas funciona com requisições HTTPS);

- *char *host* - Contém o host/domínio do servidor de interesse;
- *char *port* - contém porta do sistema fina, apenas quando for necessário (não será utilizado sempre)
- *char *path* - Contém o caminho a partir do domínio para a busca do objeto de interesses;
- *char *version*;
- *char *buf* e *size_t buflen* - esses elementos permitem o armazenamento da linha de requisição em forma de string, para facilidades no *unparsing da estrutura*;
- *HeaderFields *headers* - A estrutura *HeaderFields* armazena uma linha de cabeçalho, com um conjunto (chave, valor). Como as linhas de cabeçalho possuem números variáveis, armazena-se na requisição as linhas de cabeçalho em forma de lista.

ProxyServer.h/cpp

A classe *ProxyServer*, contida no arquivo *ProxyServer.h/cpp*, é o principal trecho de código para o funcionamento do servidor proxy desejado. Como dito anteriormente, ele permite a inspeção de requisições vindas do navegador - com possibilidade para alterações de campos das linhas de cabeçalho - e reencaminha as respostas com os objetos vindas do servidor. Essa classe possui os seguintes métodos:

- *ProxyServer(int port)* - O construtor da classe chama o método *CreateServerSocket*, fornecendo o número de porta para a ligação do proxy a seu endereço e porta pelo *bind()*;
- *void CreateServerSocket(int port)* - Esse método realiza as etapas de *socket()*, *setsoc-kopt()*, *bind()* e começa a escutar à espera de clientes por meio do *listen()*. O protocolo utilizado na construção do *socket* é IPv4 com TCP; o **bind** é feito com o próprio IP do sistema + a porta passada por parâmetro de entrada - que na verdade foi originada como um dos argumentos de linha de comando do programa. O número de *backlog* no *listen*, que indica o número de requests máximos na lista de espera do browser para o proxy;
- *void ProxyServer::ProxyRequest(int client_fd, int inspection)* - *ProxyRequest* é o método mais importante no funcionamento do servidor proxy. Uma vez que o proxy funciona tanto como cliente (para o servidor remoto) quanto como servidor (para o browser), é necessário que o proxy possua tanto o comportamento para ouvir quanto para escrever.

Assim, esse método chama o parse para a estrutura `RequestFields` após o fim dos *recv* da requisição atual. Após fazer a inspeção e eventuais alterações nas linhas de cabeçalho com *InterceptRequest(RequestFields *req)*, a classe utiliza o método *RequestToString* para transformar de volta a requisição em uma string e chama *CreateRemoteSocket*, *SendRequestRemote* e *ProxyBackClient*;

- *int CreateRemoteSocket(char* remote_addr, char* port)* - faz a criação do identificador pro socket remoto e se conecta a ele;
- *void SendRequestRemote(const char *req_string, int remote_socket, int buff_length)* - Faz o envio ao servidor remoto;
- *void ProxyBackClient(int client_fd, int remote_socket)* - Faz o envio em *chunks* da resposta do servidor para o browser (cliente).

Dificuldades na Implementação

Encontraram-se, no projeto, algumas dificuldades de implementação:

- No Proxy Server, encontrou-se alguns problemas de requisições encerrarem após o seu término, encerrando inesperadamente, assim, o funcionamento do servidor do Proxy. Para resolver isso, decidiu-se por fazer um *fork()*, a fim de criar um processo filho para cada requisição. Dessa forma, o processo pai estará sempre em execução para atender aos pedidos do browser. Essa implementação de multithreading acrescentou alguns problemas de concorrência no input e output do console, mas acredita-se que essas eventualidades foram resolvidas (pelo menos nos casos testados)
- Ocorreu-se um erro de decisão de projeto ao tentar integrar a interface gráfica ao projeto feito apenas ao fim do ciclo de desenvolvimento, o que impediu que se alcançasse o nível de integração entre a GUI e o processo worker desejado. O framework de interface gráfica utilizado foi o QT, que possui bibliotecas próprias para vários tipos e recursos naturais do C++. Isso contribuiu para colaborar no fracasso da integração com o projeto (Figura 5). Para ainda poder proporcionar uma interatividade mínima com o usuário, optou-se por implementar uma interface em linha de comando.

Funcionamento e imagens

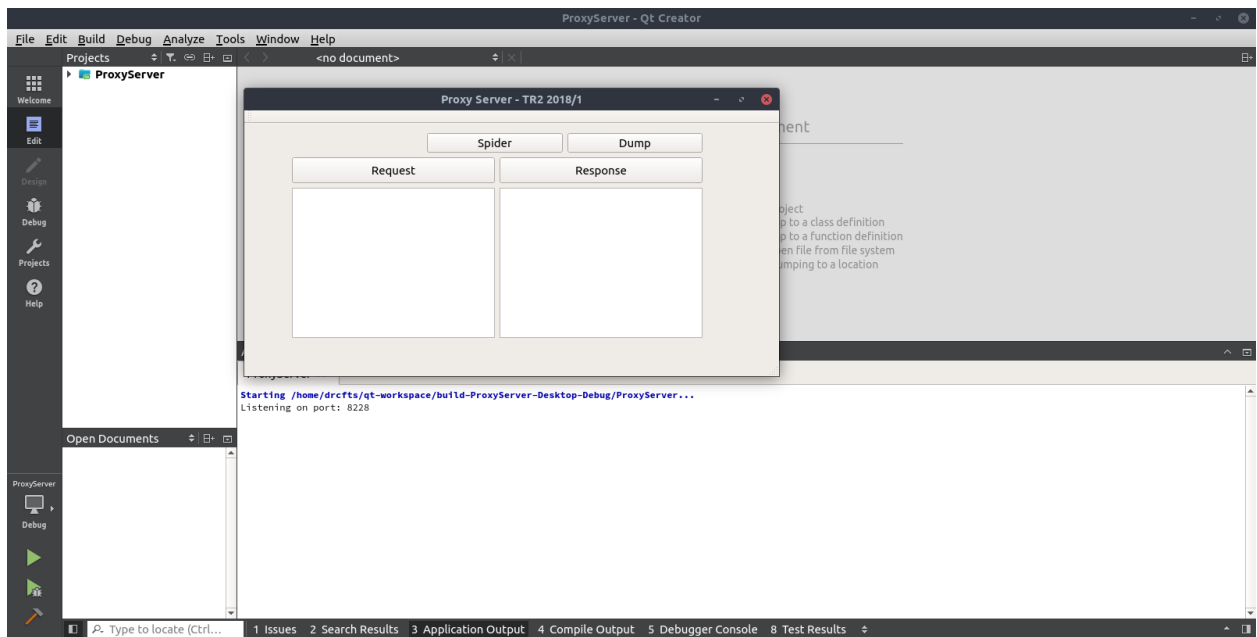


Figura 5: Tentativa de interface gráfica com o framework QT

Instruções de compilação

Para compilar o programa, basta ir ao diretório `src` e digitar no terminal:

```
$ make
```

Após isso, pode-se executar o programa de duas formas:

```
$ ./aracne -p <port>
```

```
$ ./aracne
```

O primeiro modo conecta o proxy à porta local `<port>`, enquanto o segundo conecta na porta padrão 8228. Para poder enviar requisições do browser ao programa, deve-se configurar o browser para utilizar o browser no endereço 127.0.0.1 e porta `<port>` (ou 8228, no segundo modo de rodar).

Imagens de compilação/execução

Pode-se observar as figuras de compilação e de execução do programa das figuras 6 a 9.


```
drcfts@drcfts-pc:~/Documents/Git-Projects/ProxyServer/src$ make
g++ -c -o obj/ProxyParser.o ProxyParser.cpp -Wall -g -I../include
g++ -c -o obj/ProxyServer.o ProxyServer.cpp -Wall -g -I../include
g++ -c -o obj/main.o main.cpp -Wall -g -I../include
g++ -c -o obj/spider.o spider.cpp -Wall -g -I../include
g++ -c -o obj/spider_web.o spider_web.cpp -Wall -g -I../include
g++ -c -o obj/html_parser.o html_parser.cpp -Wall -g -I../include
g++ -o aracne obj/ProxyParser.o obj/ProxyServer.o obj/main.o obj/spider.o obj/spider_w
b.o obj/html_parser.o -Wall -g -I../include -lm
drcfts@drcfts-pc:~/Documents/Git-Projects/ProxyServer/src$ ./aracne -p 5300
Listening on port: 5300
```

Figura 6: Resultado do comando make

NOTÍCIAS

COMUNICADO

O GOVERNO DO ESTADO DA BAHIA INFORMA QUE, RESPEITANDO A LEGISLAÇÃO ELEITORAL, ESTARÁ SUSPensa, DURANTE O PERÍODO DE TRÊS MESES, A PUBLICAÇÃO DE QUALQUER CONTEÚDO NOTICIOSO NOS PERIFs DAS REDES SOCIAIS E NOS SITES OFICIAIS DAS SECRETARIAS, ÓRGÃOS E ENTIDADES QUE COMPÕEM ESTA ADMINISTRAÇÃO.



Governo entrega nova ciclovia com 12 km de extensão na Paralela

[Read maps.googleapis.com](#)

[Galeria Multimídia](#)

[Diário Oficial](#) [Transparência](#)

[Fale Conosco](#) [Ações do Gove](#)

```
drcfts@drcfts-pc:~/Documents/Git-Projects/ProxyServer/src$ make
g++ -c -o obj/ProxyServer.o ProxyServer.cpp -Wall -g -I../include
g++ -o aracne obj/ProxyParser.o obj/ProxyServer.o obj/main.o obj/spider.o obj/spider_w
b.o obj/html_parser.o -Wall -g -I../include -lm
drcfts@drcfts-pc:~/Documents/Git-Projects/ProxyServer/src$ ./aracne -p 5400
***** Request Intercepted *****

GET / HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.ba.gov.br/
DNT: 1
Upgrade-Insecure-Requests: 1
Host: www.legislabahia.ba.gov.br
Connection: close

Do you want to edit the request? (Y/N)
y
What field do you want to edit
Accept-Language
What value do you want?
en-GB
```

Figura 7: Resultado da inspeção do cabeçalho

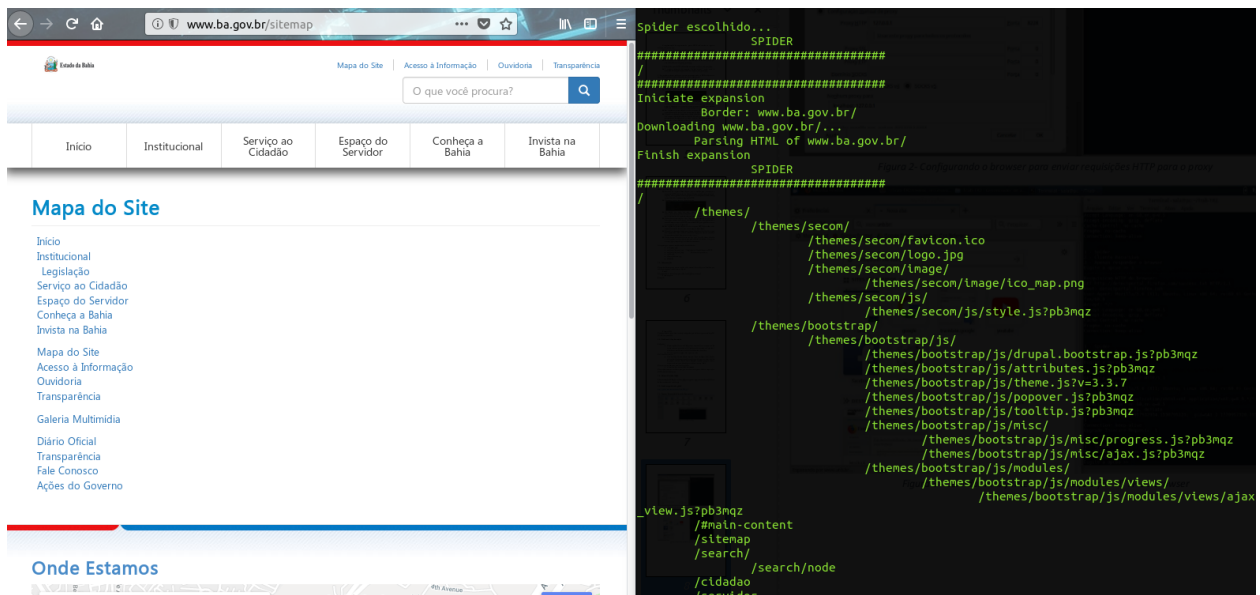


Figura 8: Resultado da execução do spider

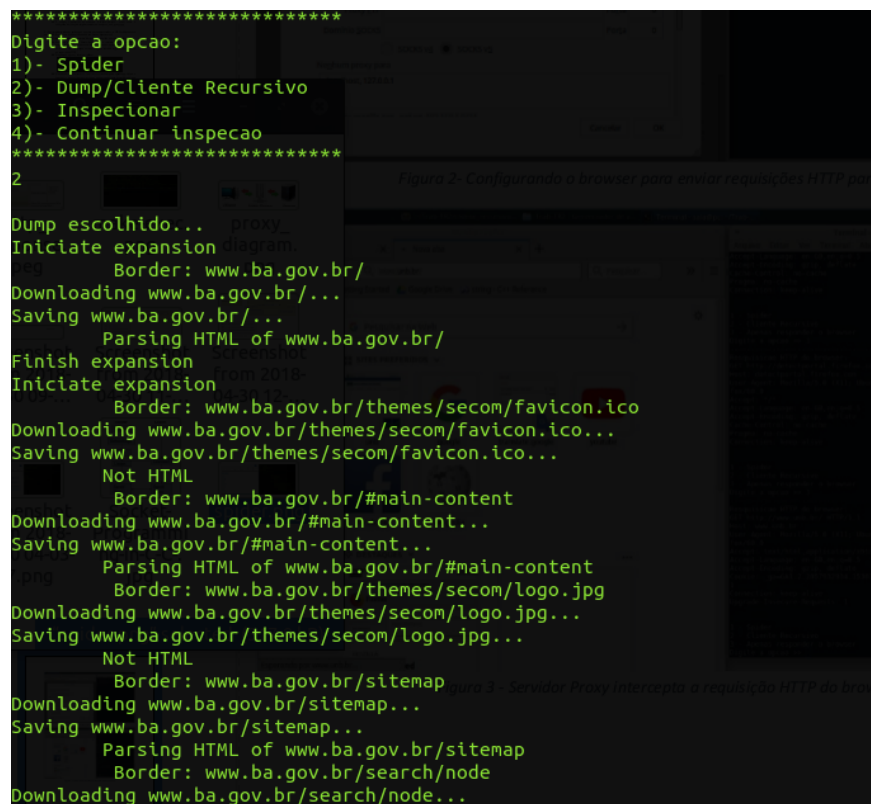


Figura 9: Resultado da execução da funcionalidade do cliente recursivo