# 一、程序设计与实现

## 1.1程序流程说明

首先利用libnet进行http数据包的捕获，之后对数据包进行存储和流还原，然后对数据包进行解析。首先判断该数据包是请求报文还是响应报文。

如果是请求报文，则首先判断请求网页服务器的ip是否命中设置的规则，如果命中，则进行网页封堵，并生成封堵日志，否则不做任何操作。之后提取出请求报文的host和URI字段，拼接成访问网站的URL,之后再去匹配URL规则，如果命中，则进行网页封堵，并生成封堵日志，否则不做任何操作。网页的封堵是调用libnids的nids\_killtcp()函数关闭客户端与服务器的tcp连接来实现的。

如果是响应报文，则首先提取出响应报文Content-Type字段，如果该字段为text/\*类型，进行记录，并继续。如果不是，则直接结束该数据包的解析。之后提取出响应报文Transfer-Encoding字段，如果该字段是chunked则进行chunk数据包的处理，之后再进行数据的gzip（现在网站都采用gzip压缩）解压。在本程序中chunk数据的提取和利用zlib库对数据包进行解压是关键点也是难点，在后面的文章会进行详细说明。

## 1.2 http数据包捕获及数据流还原流程

使用libnet对网卡80端口的数据包进行捕获，考虑到libnet捕包速度与协议分析速度不匹配的问题，我们需要设计一个数据包缓冲区存入libnet捕获的数据包，该缓冲区可以用循环队列进行是实现。之后，从缓冲区取出数据包进行数据流还原，计算数据包四元组信息的哈希值，并以此哈希值为结点向后延伸数据链，将后来到达的四元组信息相同的数据包都存入这条哈希链上。其中哈希表的头结点存储数据包的四元组信息以及解析数据包是所用到的各个字段，字段信息如下。由于网页数据并不是以一个数据包就传到客户端的，第一个传过来的响应数据包内含有响应头信息，而之后的数据包只包含数据并不含有响应头信息，所以包判断传过来的数据是否是text/html文件，是否是chunk传输等等，所以在对第一个数据包进行解析之后，就需要将解析出来的字段信息存入哈希表头结点中，以便后来的数据包进行使用。

typedef struct hash\_node //哈希表结点

{

struct tuple4 tupl4; //四元组

unsigned short is\_chunked;//标记是否为chunk传输，1是，0否

unsigned short is\_compresss; //标记是否为压缩，0代表不压缩，1代表gzip，2代表deflate

struct hash\_node \*next; //处理冲突用的指针

unsigned short is\_text;//标记是否是text/html文件，1是，0否

char size[65535];//用于存放chunksize

char size\_data[65535\*2];//用于存放chunkdata

int size\_q;//记录size数组的长度

int size\_p;//记录size\_data数组的长度

int count;//记录chunk中\r\n的个数

int length;//用于记录chunkdata的大小

TreeNode \*\*state\_node; //流式匹配状态指针

int is\_monitor;//标记内容关键字是否命中，1是，0否

z\_stream strm;//zlib解压数据包状态

}hash\_node;

## 1.3 http协议分析总流程

可以根据tcp\_stream结构体（libnids定义的结构体）的变量client.count\_new是否为真判断libnids捕获的报文时请求报文还是响应报文。当client.count\_new为真时为响应报文，进行响应报文解析，否则为请求报文，进行请求报文解析。



## 1.4 请求报文解析



在http\_protocol\_callback回调函数中进行ip匹配，该回调函数主要是利用libnet进行http数据包的捕获。ip匹配、网页封堵以及日志文件信息记录的具体实现如下：

for(int i=0;i<ip\_num;i++)//对提前设置好的IP规则进行遍历

{

//ip\_input为服务器ip地址，如果命中进行网页封堵

if(strcmp(ip\_model[i],ip\_input)==0)

{

nids\_killtcp(tcp\_connection);//libnids关闭tcp连接，实现网页的封堵。

printf("ip shot位置：%d\n",i+1);

//生成封堵日志

FILE \*file = fopen("/log/ip\_log.txt", "a");

if(file==NULL)

{

printf("open error\n");

break;

}

fputs("访问服务器ip:",file);

fputs(ip\_input,file);

fputs("\ttime:",file);

fputs(getCurrentTime(),file);

fputs("\n",file);

fclose(file);

return;

}

}

ip封堵日志的字段主要包括访问服务器的ip,当前时间。

url的拼接、url匹配、网页封堵以及封堵日志记录的实现如下。

if(strstr(temp,"GET"))//temp为响应头信息，该信息为明文传输，没有经过压缩

{

printf("请求行为：");

printf("%s\n",temp);

sscanf(temp,"%s %s %s",str1,str2,str3);

printf("使用的命令为：%s\n",str1);

printf("获得的资源为：%s\n",str2);

printf("HTTP协议类型为：%s\n",str3);

}

if(strstr(temp,"Host"))

{

printf("访问的主机为（Host）:%s\n",temp+strlen("Host: "));

int ii;

for(ii=0;temp[ii]!='\r';ii++)

{

url[ii]=temp[(int)strlen("Host: ")+ii];

}

/\* printf("url1%s\n",url);

printf("urlen1=%d\n",(int)strlen(url));\*/

int len=(int)strlen(url)-1;

for(ii=0;ii<strlen(str2);ii++)

{

url[len+ii]=str2[ii];

}

/\* printf("str2=%d\n",(int)strlen(str2));

printf("urlen2=%d\n",(int)strlen(url));\*/

printf("url:%s\n",url);

//利用ac自动机进行url匹配

int n=Search\_acTrie(root,state\_node,url,strlen(url),pattern)；

///////////////url封堵

if(n)

{

nids\_killtcp(tcp\_http\_connection);

FILE \*file = fopen("/log/url\_log.txt", "a");

if(file==NULL)

{

printf("open error\n");

break;

}

char local\_ip[16];

char server\_ip[16];

char match\_url[2048];//some resources' url are too long

memcpy(match\_url,url\_pattern[n-1],strlen(url\_pattern[n-1]));

match\_url[strlen(pattern[n-1])]='\0';

strcpy(local\_ip,inet\_ntoa(\*((struct in\_addr \*) &(tcp\_http\_connection->addr.saddr))));

strcpy(server\_ip,inet\_ntoa(\*((struct in\_addr \*) &(tcp\_http\_connection->addr.daddr))));

//url封堵日志记录

fputs("本地ip:",file);

fputs(local\_ip,file);

fputs("\t本地端口:",file);

fprintf(file,"%d",tcp\_http\_connection->addr.source);

fputs("\t\t访问服务器ip:",file);

fputs(server\_ip,file);

fputs("\t\t访问服务器端口:",file);

fprintf(file,"%d",tcp\_http\_connection->addr.dest);

fputs("\ttime:",file);

fputs(getCurrentTime(),file);

fputs("\t访问网站url:",file);

fputs(url,file);

fputs("\t命中url字符串:",file);

fputs(match\_url,file);

fputs("\n",file);

fclose(file);

return;

}

printf("url封堵n=%d\n", n); }

url的字符串匹配主要是利用ac自动机实现的。

## 1.5 响应报文解析



关键字的匹配也是利用ac自动机实现的。该部分chunk数据块的处理以及gzip数据包的解压缩将在下节进行介绍。

# 二、关键技术说明

## 2.1 chunk数据块的处理

chunk数据块的格式：chunksize:数据块的大小\r\ndata:数据\r\n。根据chunk数据块的格式，在处理chunk数据时首先需要读取chunksize的大小，然后再去读取后面的数据。在读取完chunksize之后，由于它以16进制的方式存储，所以需要将chunksize转成10进制数据。在读取chunkdata的时候，需要注意并不是一个数据包包含一个chunk块，有可能包含多个chunk块，还有时候一个chunk块分为两个数据包传送过来。所以在读取完chunksize之后，需要将chunksize存入哈希表头中，另外再去读chunksize长度的数据，直到读完这么长的数据结束。如果一个chunk块分为多个数据包传送，则需要将已经读取的数据大小存入哈希表头中，在另外一个数据包到来时，在读取剩余长度的数据。另外，在读完数据长度和数据之后，需要跳过\r\n的读取。在数据包末尾以\r\n\r\n结束，所以在读取到两个\r\n之后，代表数据传输完成。具体代码如下：

for(m=0;m<j;m++) {

if(p->count==0)//记录长度

{

if (entity\_content[m] != '\r' && entity\_content[m + 1] != '\n') {

p->size[p->size\_q++] = entity\_content[m];

}

else

{

p->count++;

m++;

continue;

}

}

else//记录数据

{

p->length=char2int(p->size);

if(p->size\_p<p->length)

{

p->size\_data[p->size\_p++]=entity\_content[m];

}

else

{

p->size\_data[p->size\_p] = '\0';

m++;

p->count=0;

printf("size=%s\n",p->size);

printf("q=%d\n",p->size\_q);

printf("p=%d\n", p->size\_p);

p->size\_q=0;

}

}

上述代码中，p代表哈希表表头节点，entity\_content数组存储的报文数据。p->count=0时代表此时读取chunksize,在读完chunksize之后执行+1操作去读取chunkdata，在读完chunkdata之后，执行-1操作，接下来继续读取chunksize。p->size[]数组存储的是chunksize。p->size\_data[]存储的chunkdata。char2int（）函数实现了将16进制的chunksize转换成10进制的数据。

## 2.2 gzip数据的流式解压缩

所谓流式解压缩是指在处理完一个chunk数据块之后就使用zlib库解压gzip数据，而不是处理所有chunk数据块，然后拼接完数据块之后再去解压缩gzip数据，这种非流式解压缩数据的操作太过繁琐。使用zlib解压gzip数据的代码如下。

char \*UnCompress(char \*src, int len, char \*\*unz\_data, long int \*unz\_len, unsigned short is\_compress)

{

z\_stream strm;//声明解压流

char \*dest = NULL;

unsigned have;

unsigned char out[OUTBUFFLEN];//解压后的报文

int totalsize = 0;

//allocate inflate state

strm.zalloc = Z\_NULL;

strm.zfree = Z\_NULL;

strm.opaque = Z\_NULL;

strm.avail\_in = len;//待解压数据的长度

strm.next\_in = (Byte\*)src;//待解压数据

ret = inflateInit2(&strm, 47); //zlib初始化

//run inflate() on input until output buffer not full

do

{

strm.avail\_out = OUTBUFFLEN;

strm.next\_out = out;

ret = inflate(&strm, Z\_NO\_FLUSH); //zlib数据解压

if(ret!=Z\_OK)

{

printf("Z\_OK ERROR\n");

}

if(ret!=Z\_STREAM\_END)

{

printf("Z\_STREAM\_END ERROR\n");

}

if (ret != Z\_OK && ret != Z\_STREAM\_END)

{

printf("\ninflate ret = %d\n", ret);

//return NULL;

}

have = OUTBUFFLEN - strm.avail\_out;

printf("解压后的数据：\n");

for(int i=0;i<have;i++)

{

printf("%c",out[i]);

}

printf("\n");

totalsize += have;

dest = (char\*)realloc(dest,totalsize);

if(dest == NULL)

{

perror("realloc");

return NULL;

}

memcpy(dest + totalsize - have,out,have);

dest[totalsize-1]='\0';

} while (strm.avail\_out == 0);

//(void)inflateEnd(&strm);

\*unz\_data = dest;

\*unz\_len = totalsize;

return dest;

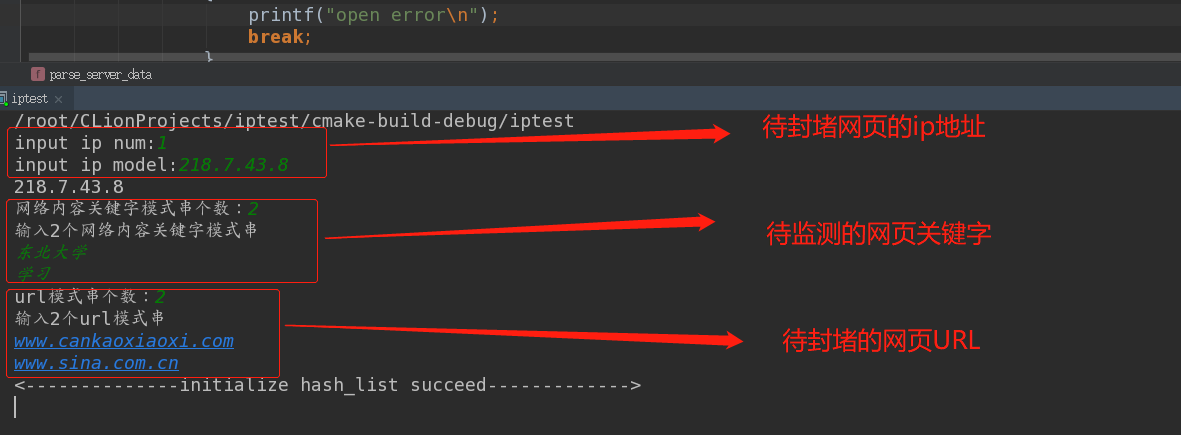
}

zlib使用方法:声明解压流z\_stream strm;初始化strm.zalloc，strm.zfree，strm.opaque = Z\_NULL;strm.avail\_in = len;//待解压数据的长度，strm.next\_in = (Byte\*)src;//待解压数据，然后对zlib进行初始化inflateInit2(&strm, 47);之后循环遍历缓冲区调用解压函数inflate()去解压数据。

# 三、程序运行测试

本程序在linux环境下进行开发和使用，操作系统为centos6，编译软件clion，gcc。

## 3.1对ip规则、关键字规则、URL规则进行设置



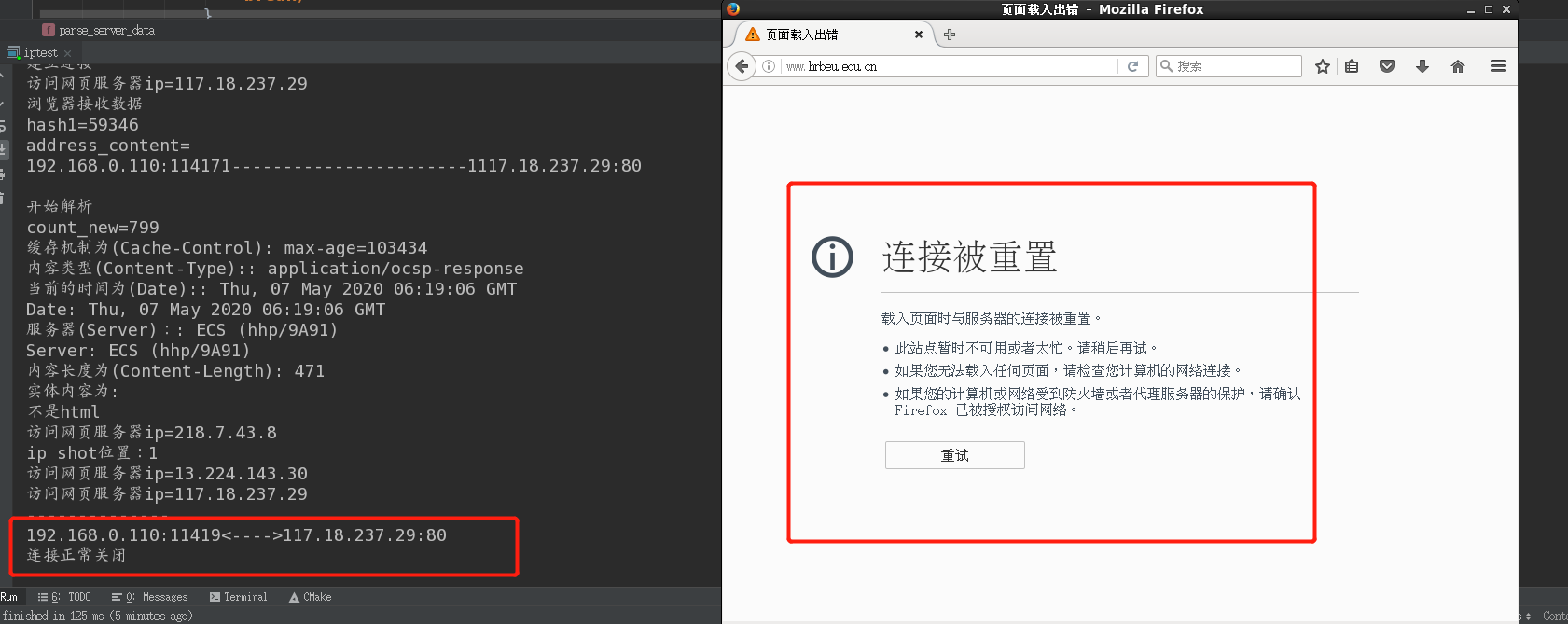
218.7.43.8为哈尔滨工程大学官网IP

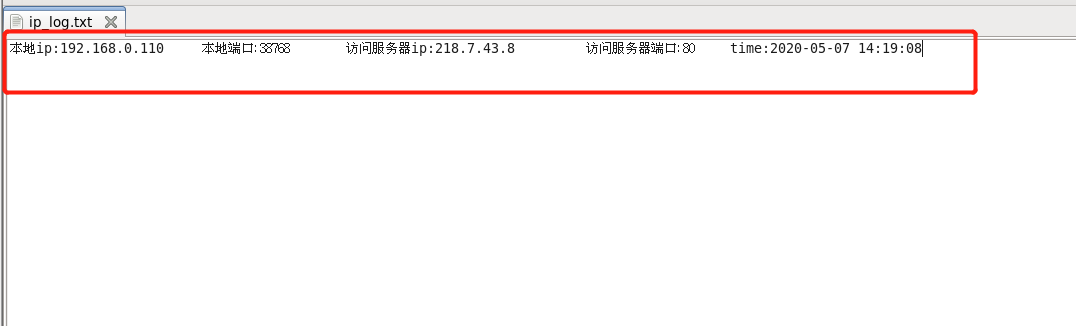
## 3.2测试ip封堵

在不启动程序的情况下访问哈尔滨工程大学主页，发现正常访问。



启动程序，在浏览器访问哈尔滨工程大学官网主页。命中规则，发现不能访问网页。查看生成的日志，发现日志正常生成。





## 3.3测试url封堵

在不启动程序的情况下访问参考消息官网、新浪官网，发现正常访问。





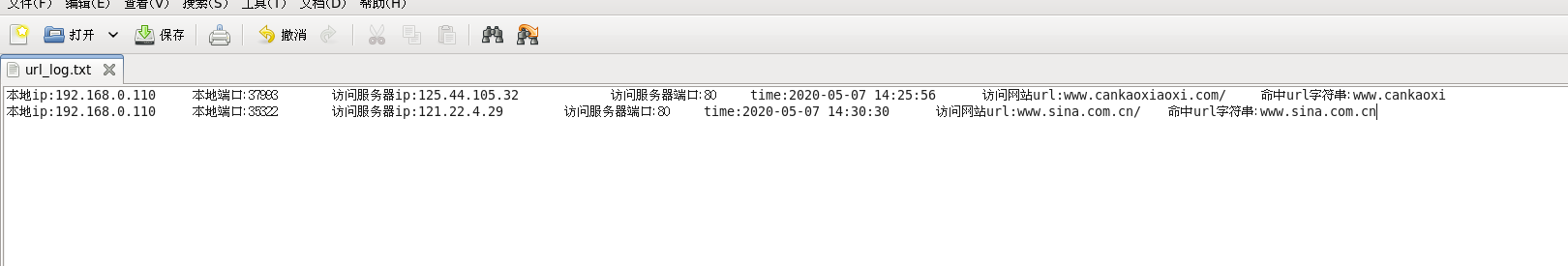
启动程序，访问[www.cankaoxiaoxi.com](http://www.cankaoxiaoxi.com)。



访问[www.sina.com.cn](http://www.sina.com.cn)

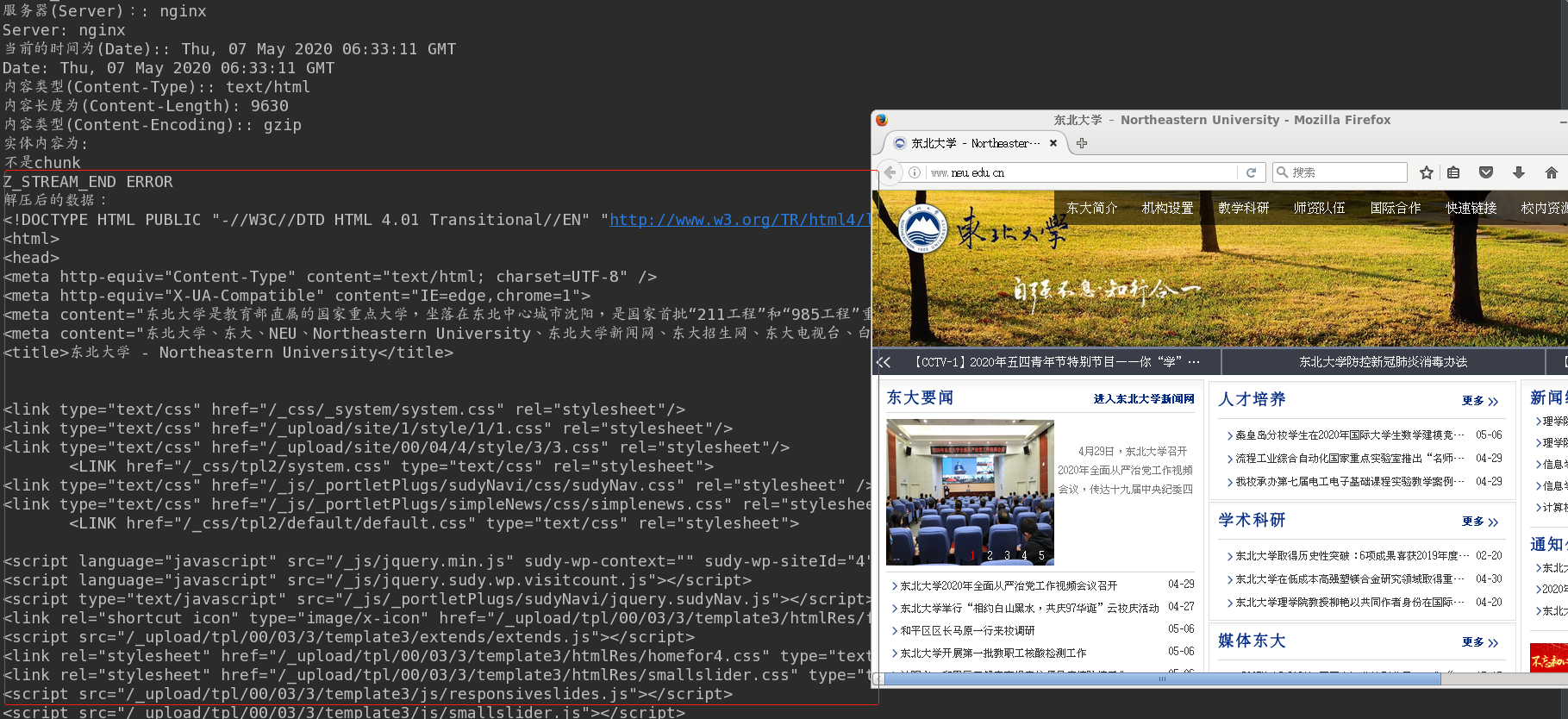


url拼接成功，并成功匹配规则，之后连接成功被reset，对网页封堵成功。查看生成的日志文件。



## 3.4测试关键字监测

访问东北大学官网。





可以看出，报文解析成功，并成功匹配出关键字。由于解析的报文过长，在这就截取部分数据。

查看生成的日志文件。

