CS3031 Proxy Server Assignment Golang

by Dylan LEWIS

Submission date: 19-Feb-2018 04:15PM (UTC+0000)
Submission ID: 918160776
File name: Telecomms_3.pdf (158.65K)
Word count: 2770
Character count: 12356

CS3031 Assignment

Dylan Lewis, Student no 15317659

February 2017

Contents

1 Introduction

The goal of this assignment was to create a working proxy server which fetches items from the internet for a client and then sends the response to the client. The proxy server should also allow for the caching of pages as well as access control. This proxy server was implemented in the language Golang. In order to run the server install golang then enter on the command line "go run server.go cache.go".

2 HTTP/S Requests

For the first part of the proxy the server must be able to handle HTTP and HTTPS requests from the client. In order to do this the program listens on port 8080 for any incoming packets. The Golang Server struct handles parsing a HTTP and HTTPS packet into a Golang Request struct. The program then evaluates whether the request is a HTTP or HTTPS request. If it's a HTTP request then the request is passed to the HTTP handler function if it's HTTPS it's passed to the HTTPS handler function. The HTTP handler function determines the destination of the request. It then sends the packet to the destination and waits for a response, when the response arrives it reformats the headers to make it usable by the client. The body of the response and the newly formatted header are then sent off to the client using the Golang ResponseWriter struct. Thanks to the ListenAndServe function on a request a new thread will be spun up to handle it.

Next the HTTPS packets must be handled. In order determine whether the connection is a HTTP or HTTPS connection the program looks at the method field in the HTTP packet. If this method field has the value "CONNECT" then it is looking to upgrade the connection to a HTTPS or Websocket connection. To handle this the HTTPS handler function allows for tunneling. To implement tunneling the proxy server establishes a tcp connection between itself and the client, then establishes a tcp connection between the proxy server and the destination.

Once the tcp connections are established all packets from the client connection are forwarded to the destination and all packets from the destination are forwarded to the client. This connection allows for both HTTPS and Websockets. The connection is kept open until forwarding is done, it's done when all packets from both ends have been transmitted or until the connection is closed on either end.

3 Blocking

The next part of the assignment was implementing a blocklist. First I decided that the blocklist would be an array of strings and on each request I would compare and see if the url is a member of the blocklist by using the Compare function in the Golang string library. If the url is in the blocklist then "Blocked by Proxy" is printed in the client's browser unless it's HTTPS in which case the browser will simply fail to connect. I decided that the blocklist would be persistent so it is saved to a file. I decided to save it to a json file called "blocked.json" . At the start of the program the urls from this file are loaded into an array. After every addition to the blocklist the in-memory version is updated first then the json file is updated this is because the in-memory blocklist needs to be the most upto date for url checking. The new blocklist is then written back to the file, however the program ensures that the new blocklist and the on disk have only a difference of one url.

4 Console

The console part of the assignment was a way of adding and removing urls from the blocklist. For the console I have it as a HTML page which is served when the client makes a HTTP request for "://console". The console is handled by the Console function if it is just the console being requested then the "base.html" file is served to the user. The console HTML page has 2 forms on it one for adding urls to the blocklist and another for removing urls from the blocklist. The url to be added or removed can be entered into the appropriate form's text box and once done submitted as a post request. This request is then handled by the Console function again if it's a remove request it removes the url from the blocked list and then writes new blocked list to file. If it's an add to blocklist request it adds the url to the blocklist and displays the new blocklist.
Finally the console management page has a link to "console/blocked". Upon a request to that url the current state of the blocklist is displayed to the user.
If the person running the server wishes to view the requests at any given moment they can view the requests coming in on the terminal which is running the proxy server program as the program prints out the request type HTTP/HTTPS and the url of the request.

5 Cache

The cache is used to store HTTP responses as it not possible to resend the same HTTPS response. The cache maps a request's url to a byte array, the byte array is a response to the request. When a http request is sent to the proxy server it checks if the request is present in the cache if the request is not in the cache then the request is sent to the destination and the response is saved in the cache as a byte array the terminal then prints out "Miss". If the request is in the cache then the byte array is loaded from the cache and we check if the response has expired by checking the "Expires" field. If the response has expired then the request is sent to the destination and the new response is saved in the cache. However if the response is present in the cache and has not expired then the byte array in the cache is converted into a HTTP response and sent to the client. The aim of this is to reduce the overall time required to load data into

the browser
Overall the cache caused the loading times of pages that were being refreshed to be reduced dramatically

6 Code

6.1 Cache.go

```go
package main
import(
  "bytes"
  "bufio"
  "fmt"
  "net/http"
  "net/http/httputil"
  "sync"
  "time"
)
const dumpBody = true

type Cache struct {

  // Use sync.Map as it a thread safe Dictionary
  // Maps url to byte array
  Elems sync.Map
}

func CreateCache() *Cache {

  return &Cache{}
}

// Check if cache contains the response to the http request
// If in cache byte array and true are returned if not in cache
// nil and false are returned
func Hit(req *http.Request,cache *Cache) ([]byte,bool) {

  elems := cache.Elems
  val, ok := elems.Load(req.URL.String())
  if ok {
    return val.([]byte),true
  }
  return nil,false
}

func Expired(req *http.Request,cache *Cache) bool{

  elems := cache.Elems
  val, ok := elems.Load(req.URL.String())
  r := bufio.NewReader(bytes.NewReader(val.([]byte)))
  resp, _ := http.ReadResponse(r, req)

  arr, val_key := resp.Header["Expires"]
  // Evaluate whether it has expires field and it is present in the
     cache
  if ok && val_key {
    time_val, _ := time.Parse(time.RFC1123, arr[0])
    if int64(time_val.Sub(time.Now())) < 0 {
      return true
    }
  }
  return false

}


func Insert(req *http.Request, resp * http.Response, cache *Cache){


  body, err  := httputil.DumpResponse(resp, dumpBody)
  if err != nil {
    fmt.Println("Unable to add to cache")
  } else {
    req_str := req.URL.String()
    cache.Elems.Store(req_str, body)
  }
}
```

6.2 Server.go

```
package main


import (
  "bufio"
  "bytes"
  "encoding/json"
  "fmt"
  "io"
  "io/ioutil"
  "log"
  "net"
  "net/http"
  "os"
  "strings"
  "time"
)

const JsonFile = "./blocked.json"
const ConsoleAdr = "://console"


type Blocked struct{

  URL string `json:url`

}

var blockedUrls []Blocked
var cachedUrls *Cache


```

```
// converts a Blocked struct to a string
func (b Blocked) toString() string {
  bytes, err := json.Marshal(b)
  if err != nil {
    fmt.Println(err.Error())
    os.Exit(1)
  }

  return string(bytes)

}


//loads the blocked urls from the json file
func LoadBlocked() []Blocked {
  file, err := ioutil.ReadFile(JsonFile)
  if err != nil {
    fmt.Printf("File error: %v\n", err)
    os.Exit(1)
  }
  var result []Blocked
  json.Unmarshal(file, &result)
  return result
}


// Write the blocklist back to the file
func WriteBlocked() {

  result := []byte("[")
  length := len(blockedUrls)
  // Convert the urls into a json format
  for index,item := range blockedUrls{
    bytes, err := json.Marshal(item)
    if err != nil {
      fmt.Println(err.Error())
      os.Exit(1)
    }
    result = append(result, bytes...)
    if index < length - 1 {
          result = append(result, []byte(",")...)
    }

  }
  result = append(result, []byte("]")...)
  ioutil.WriteFile("./blocked.json",result, 0644)
}


// This is repsonsible for dealing with HTTP headers
// Checks destination then sends the request to its destination
func HttpHeader(w http.ResponseWriter, req *http.Request){
  client := &http.Client{}
  fmt.Printf("HTTP request received for %s\n", req.URL.String())
  req.RequestURI = ""
  // If request is for console call Console handler
  if strings.Contains(req.URL.String(), ConsoleAdr) {
    Console(w, req)

  } else if IsBlocked(req.URL.String()) {
    fmt.Fprintf(w, "%s", "Blocked by proxy")

  } else {
    var resp *http.Response
    // check if in cache
    val, hit := Hit(req, cachedUrls)
    if !hit || Expired(req, cachedUrls) {
      // If not in cache or expired insert packet into cache
      resp , _ = client.Do(req)
      fmt.Println("Miss")
      Insert(req, resp, cachedUrls)
    } else {
      fmt.Println("Hit")
      r := bufio.NewReader(bytes.NewReader(val))
      resp, _ = http.ReadResponse(r, req)

    }
    a := w.Header()
    b := resp.Header
    FormatHeader(a, b)
    w.WriteHeader(resp.StatusCode)
    // Copy body to ResponseWriter
    io.Copy(w, resp.Body)
  }

}

// Checks if the url is in the blocked list
func IsBlocked(url string) bool  {
  for _,blocked := range blockedUrls {
    if strings.Contains(url, blocked.URL) {
      return true
    }

  }
  return false
}

// This handles the console page which adds and removes
// urls from the blocklist
func Console(w http.ResponseWriter, req *http.Request){
  if strings.Contains(req.URL.String(), "/blocked") {
    // create string of blocked urls which will be displayed to
    user
    blockString := ""
    for _, item := range blockedUrls {
      blockString = fmt.Sprintf("%s <li>%s</li>", blockString, item
    .URL)
    }
    fmt.Fprintf(w,"<html> <h1>Block list</h1> <body><ul> %s </ul>
    </body> </html>", blockString)
  } else if strings.Contains(req.URL.String(), "/remove") &&  req.
    Method == "POST"  {
    // This iterates through the blocklist and removes the
    specified url from the list
    url := req.PostFormValue("remove_url")
    for index, item := range blockedUrls {
      if strings.Compare(url, item.URL) == 0 {
        blockedUrls[len(blockedUrls)-1], blockedUrls[index] =
    blockedUrls[index], blockedUrls[len(blockedUrls)-1]
        blockedUrls = blockedUrls[:len(blockedUrls)-1]
        break
      }
    }
    defer WriteBlocked()
    fmt.Fprintf(w,"successfully removed %s from the blocked list",
    url)

  } else if req.Method == "POST" {
    // Get the specified url to be added to the blocked list
    url := req.PostFormValue("url")
    newUrl := Blocked{ URL: url,}
    blockedUrls = append(blockedUrls, newUrl)
    defer WriteBlocked()
    // Print new blocked list for user
    blockString := ""
    for _, item := range blockedUrls {
      blockString = fmt.Sprintf("%s <li>%s</li>", blockString, item
    .URL)

    }
    fmt.Fprintf(w,"<html> <h1>New blocked list</h1><body><p>%s
    added</p><p>List currently has the following urls</p><ul> %s </
    ul></body></html>", url, blockString)


  } else {

    http.ServeFile(w, req, "base.html")
  }
}


// Formats header so it can be used by client
func FormatHeader(dest , src http.Header ){

  for k, vs := range src {
    for _, v := range vs {
      dest.Add(k, v)
    }
  }
  dest.Del("Proxy-Connection")
  dest.Del("Proxy-Authenticate")
  dest.Del("Proxy-Authorization")
  dest.Del("Connection")
}

// This forwards the bytes from one tcp connection to another
func CopyTo(dest, src net.Conn){
  defer src.Close()
  io.Copy(dest, src)
```

```

}

func HttpsHeader(w http.ResponseWriter, req *http.Request){
  if !IsBlocked(req.URL.String()) {
    // Establish tcp connection with destination
    dest_conn, err := net.Dial("tcp", req.URL.Host)
    hjk, works := w.(http.Hijacker)
    if !works {
      http.Error(w, "Hijacking not supported", http.
    StatusInternalServerError)
      return
    }
    client_conn, _, err := hjk.Hijack()
    if err != nil {
      http.Error(w, err.Error(), http.StatusServiceUnavailable)
    }
    // Prints https request
    fmt.Printf("HTTPS request received for %s\n", req.URL.String())
    // accepts the https upgrade
    client_conn.Write([]byte("HTTP/1.0 200 OK\r\n\r\n"))

    // Now all thats left is to forward the https requests and
    bytes
    // from the client to the destination and the responses back to
     the
    // client
    go CopyTo(dest_conn, client_conn)
    go CopyTo(client_conn, dest_conn)
  }
}

// Handler function for server which determines which function
// to use based on request method
func Handler(w http.ResponseWriter, r *http.Request) {
  start := time.Now()
  if r.Method != http.MethodConnect {
    HttpHeader(w, r)
  } else {
    HttpsHeader(w,r)

  }
  fmt.Println("Time taken to serve is " + time.Since(start).String
    ())
}


func main(){
  blockedUrls = LoadBlocked()
  cachedUrls =  CreateCache()
  server := http.Server{
    Addr: ":8080",
    Handler: http.HandlerFunc(Handler),
  }
  err := server.ListenAndServe()
  if err != nil {
    log.Fatalln("Error: %v", err)
  }
}
```

6.3 base.html

```
<html>
<h1>Management Console
</h1>

<body>
  <p>
    This is the management console this allows you to view block
  </p>
    <a href="/blocked">View Blocked URLs</a>
  <p>To add a url to the blocklist enter it here and submit<p>
  <form action="/" method="post">
    <input type="text" name="url" value="">
  <br>
  <input type="submit" value="Submit">
  </form>

  <p>To remove a url enter it here and submit<p>
  <form action="/remove" method="post">
    <input type="text" name="remove_url" value="">
  <br>
  <input type="submit" value="Submit">
  </form>
</body>
</html>
```

6.4 blocked.json

```
[{"URL":"test.com"},{"URL":"text.com"}]
```