CS3031 Web Proxy Assignment

James Tait - 16321184

I used the Go programming language to build this web proxy, using only the built-in packages. The main function simply reads the console for input to blacklist URLs in one ‘goroutine’ (thread), and creates a server to listen for requests on port 8080. When a request is received, a new thread is created and subsequently runs the HTTP handler.

To block URLs or hosts, you type `/b <hostname>` into the console where you are running the proxy from. To unblock, use `/u <hostname>` instead. There is a global slice of strings that the hostname is added to, and any incoming requests are checked against all the blacklist entries. If the requested URL is found in the blacklist, then a `HTTP/1.0 403 Forbidden` error response is returned to the client.

For HTTP requests, the request is first checked against the list of blacklisted URLs. The request’s proxy headers are removed, and then the method of the request is checked. If it is a `CONNECT` request, the HTTPS handler is called. Otherwise, the request is forwarded on to the server and then the response is just relayed to the client.

For HTTPS requests, the proxy hijacks the connection the client is making to the server and makes a separate connection to the server itself. Then it concurrently copies data from both the client to the server and from the server to the client. Once they stop sending data, they close their half of the connection.

```
package main

import (
        "bufio"
        "fmt"
        "io"
        "log"
        "net"
        "net/http"
        "os"
        "regexp"
        "strings"
)

var hasPort = regexp.MustCompile(`:\d+$`)
var buffer [256]string
var blockedHosts []string = buffer[0:0]

func removeProxyHeaders(r *http.Request) {
        r.RequestURI = ""
        r.Header.Del("Proxy-Connection")
        r.Header.Del("Proxy-Authenticate")
        r.Header.Del("Proxy-Authorization")
        r.Header.Del("Connection")
}
```

```
//copies data from src to dst and then closes relevant halves of both connections
func copyAndClose(dst, src *net.TCPConn, host string) {
        if _, err := io.Copy(dst, src); err != nil {
                fmt.Printf("Error copying to client: %s", err)
        }
        dst.CloseWrite()
        src.CloseRead()
        if host != "" {
                fmt.Printf("Connection to %s Closed\n", host)
        }
}

func handleHTTP(w http.ResponseWriter, req *http.Request) {
        fmt.Printf("Rquest received: %s %s\n", req.Method, req.URL)

        //Check the host is not blacklisted
        for _, h := range blockedHosts {
                if strings.Contains(req.URL.Host, h) {
                        fmt.Printf("%s is currently blacklisted, refusing request.\n", h)
                        w.Write([]byte("HTTP/1.0 403 FORBIDDEN\r\n\r\n"))
                        req.Body.Close()
                        return
                }
        }

        removeProxyHeaders(req)
        //switch to https handler if a CONNECT request is received
        if req.Method == "CONNECT" {
                handleHTTPS(w, req)
        } else {
                c := &http.Client{}
                res, err := c.Do(req) //forward request
                if err != nil {
                        log.Fatal(err)
                }

                fmt.Printf("Response received: %s\n\n", res.Status)
                res.Write(w) //relay response to client

                //Go requires the bodies to be closed
                req.Body.Close()
                res.Body.Close()
        }
}

func handleHTTPS(w http.ResponseWriter, req *http.Request) {
        //set the scheme to avoid "unsupported protocol scheme" errors
        req.URL.Scheme = "https"

        hij, ok := w.(http.Hijacker)
        if !ok {
                panic("httpserver does not support hijacking")
        }
        proxyClient, _, err := hij.Hijack() //hijack connection
        if err != nil {
                panic("Cannot hijack connection " + err.Error())
        }
        host := req.URL.Host
        if !hasPort.MatchString(host) {
```

```
                host += ":80"
        }

        //conect to server
        targetSiteCon, err := net.Dial("tcp", host)
        if err != nil {
                log.Println(err.Error())
        }

        fmt.Printf("Accepting CONNECT to %s\n", host)
        proxyClient.Write([]byte("HTTP/1.0 200 OK\r\n\r\n"))
        //get the two tcp connections and stream the data between them
        targetTCP, targetOK := targetSiteCon.(*net.TCPConn)
        proxyClientTCP, clientOK := proxyClient.(*net.TCPConn)
        if targetOK && clientOK {
                go copyAndClose(targetTCP, proxyClientTCP, "")
                go copyAndClose(proxyClientTCP, targetTCP, host)
        }
}

func readConsoleInput() {
        scanner := bufio.NewScanner(os.Stdin)
        for scanner.Scan() {
                input := scanner.Text()
                host := input[3:]
                if input[0:3] == "/b " {
                        //command to block URL or host etc
                        blockedHosts = append(blockedHosts, host)
                        fmt.Printf("Blocked %s\n", host)
                } else if input[0:3] == "/u " {
                        //command to unblock URL or host etc
                        for i, h := range blockedHosts {
                                if strings.Contains(host, h) {
                                        blockedHosts = append(blockedHosts[:i],
blockedHosts[i+1:]...)
                                        break
                                }
                        }
                        fmt.Printf("Unblocked %s\n", host)
                }
        }
}

func main() {
        httpHandler := http.HandlerFunc(handleHTTP)
        fmt.Printf("Proxy activated!\n\n")

        go readConsoleInput()
        //listen for and serve requests on port 8080
        //and use 'httpHandler' to handle them
        http.ListenAndServe(":8080", httpHandler)
}
```