Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 4.1

по курсу «Компьютерные системы и сети»

«SSH сервер и клиент»

Студент группы ИУ9-32Б Федуков А. А.

Преподаватель Посевин Д. П.

29 октября 2024 г.

Цель работы

Целью данной работы является разработка SSH-сервера и клиента на языке GO.

Часть 1. Разработка SSH-сервера

Задача: Реализовать ssh сервер на языке GO с применением указанных пакетов и запустить его на localhost. Проверка работы должна проводиться путем использования программы ssh в ОС Linux/Unix или PuTTY в ОС Windows. Должны работать следующие функции:

- авторизация клиента на ssh сервере;
- создание директории на удаленном сервере;
- удаление директории на удаленном сервере;
- вывод содержимого директории;
- перемещение файлов из одной директории в другую;
- удаление файла по имени;
- вызов внешних приложений, например ping.

Реализация

Я создал программу, которая слушает порт 2288 server.go и при попытки соединения по протоколу ssh проверяет логин и пароль authenticate.go на нахождение в clients.json, а после предоставляет возможность выполнить команды реализованные в commands.go.

Код

Листинг 1: Файл clients.json

```
[
    {
        "login": "Alice",
        "password": "aboba"
    },
    {
        "login": "user",
        "password": "123"
    }
]
```

Листинг 2: Файл authenticate.go

```
package main

import (
  "encoding/json"
  "fmt"
  "os"
)

type User struct {
  Login    string `json:"login"`
  Password string `json:"password"`
}

func writeJSON(filePath string, data interface{}) error {
  jsonData, err := json.MarshalIndent(data, "", "    ")
  if err != nil {
    return err
  }

  return os.WriteFile(filePath, jsonData, 0644)
}

func readJSON(filePath string, data interface{}) error {
  jsonData, err := os.ReadFile(filePath)
  if err != nil {
    return err
  }

  return json.Unmarshal(jsonData, data)
```

```
30 }
31
32 func checkUserPasswd(lgn string, psswd string) bool {
33   var users []User
34   if err := readJSON(WORKING_FOLDER+USER_LOGIN_FILE, &users); err != nil
       {
35     fmt.Println("Error reading JSON:", err)
36     return false
37   }
38
39   for _, user := range users {
40     if user.Login == lgn && user.Password == psswd {
41       return true
42     }
43   }
44   return false
45
46 }
```

Листинг 3: Файл server.go

```
1  package main
2
3  import (
4    "fmt"
5    "io"
6    "log"
7    "strings"
8
9    "github.com/gliderlabs/ssh"
10   terminal "golang.org/x/term"
11 )
12
13 // ssh -p 2222 127.0.0.1 -o StrictHostKeyChecking=no
14
15 var is_Authenticated = false
16 var userName = ""
17 var userLogin = ""
18
19 func updatePromt(term *terminal.Terminal) {
20   term.SetPrompt(fmt.Sprintf("%s@%s:%s > ", userLogin, userName,
       WORKING_FOLDER))
21 }
22
23 func sessionHandler(s ssh.Session) {
24   defer s.Close()
25   if s.RawCommand() != "" {
```

```
26      io.WriteString(s, "raw commands are not supported")
27      return
28    }
29
30    // создаем терминал
31    term := terminal.NewTerminal(s, fmt.Sprintf("/%s/ > ", s.User()))
32
33    // добавляем обработку pty-request
34    pty, winCh, isPty := s.Pty()
35    if isPty {
36      _ = pty
37      go func() {
38        // реагируем на изменение размеров терминала
39        for chInfo := range winCh {
40          _ = term.SetSize(chInfo.Width, chInfo.Height)
41        }
42      }()
43    }
44
45    for {
46      if !is_Authenticated {
47        term.SetPrompt("")
48
49        io.WriteString(term, "Enter Login: ")
50        lgn, err := term.ReadLine()
51        if err == io.EOF {
52          _, _ = io.WriteString(s, "EOF.\n")
53          break
54        }
55        userLogin = lgn
56        userName = s.User()
57
58        psswd, err := term.ReadPassword("Enter Password: ")
59        if err == io.EOF {
60          _, _ = io.WriteString(s, "EOF.\n")
61          break
62        }
63
64        if checkUserPasswd(lgn, psswd) {
65          is_Authenticated = true
66          updatePromt(term)
67          io.WriteString(term, "Authenticated\n")
68
69        } else {
70          io.WriteString(term, "Wrong login or password\n")
71        }
```

```
72      } else {
73        line, err := term.ReadLine()
74        if err == io.EOF {
75          _, _ = io.WriteString(s, "EOF.\n")
76          break
77        }
78        fmt.Println("Recieved line:", line)
79        cmd := strings.Split(line, " ")
80        handleCommand(term, cmd[0], cmd[1:]...)
81
82      }
83
84    }
85  }
86
87  func startServer() {
88    ssh.Handle(sessionHandler)
89
90    log.Fatal(ssh.ListenAndServe(SSH_PORT, nil))
91  }
```

Листинг 4: Файл commands.go

```
1  package main
2
3  import (
4    "fmt"
5    "os"
6    "os/exec"
7    "path/filepath"
8
9    terminal "golang.org/x/term"
10 )
11
12 func handleCommand(term *terminal.Terminal, command string, args ...
       string) {
13   path := WORKING_FOLDER
14   if len(args) != 0 {
15     path = args[0]
16   }
17
18   switch command {
19   case "pwd":
20     term.Write([]byte(WORKING_FOLDER + "\n"))
21   case "exit":
22     is_Authenticated = false
23   case "cd":
```

```
24      fmt.Println(args)
25      if args[0] == ".." {
26        WORKING_FOLDER = filepath.Dir(WORKING_FOLDER[:len(WORKING_FOLDER)
        -1]) + "/"
27      } else {
28        dest := args[0]
29        if filepath.IsLocal(dest) {
30          dest = WORKING_FOLDER + dest
31        }
32
33        if dest[len(dest)-1] != '/' {
34          dest += "/"
35        }
36
37        WORKING_FOLDER = dest
38      }
39      updatePromt(term)
40
41    case "ls":
42      term.Write([]byte(WORKING_FOLDER + "\n"))
43      intend := "  --> "
44      files, err := os.ReadDir(path)
45      if err != nil {
46        term.Write([]byte("Error reading directory: " + err.Error()))
47      } else {
48        for _, file := range files {
49          term.Write([]byte(intend + file.Name() + "\n"))
50        }
51      }
52
53    case "mkdir":
54      if err := os.Mkdir(path, 0755); err != nil {
55        term.Write([]byte("Error creating directory: " + err.Error()))
56      } else {
57        term.Write([]byte("Directory created\n"))
58      }
59
60    case "rmdir":
61      if err := os.Remove(path); err != nil {
62        term.Write([]byte("Error deleting directory: " + err.Error()))
63      } else {
64        term.Write([]byte("Directory deleted\n"))
65      }
66
67    case "mv":
68      fmt.Println(args[0], args[1])
```

```
69      if err := os.Rename(args[0], args[1]); err != nil {
70        term.Write([]byte("Error moving file: " + err.Error()))
71      } else {
72        term.Write([]byte("File moved\n"))
73      }
74
75    case "rm":
76      if err := os.Remove(path); err != nil {
77        term.Write([]byte("Error deleting file: " + err.Error()))
78      } else {
79        term.Write([]byte("File deleted\n"))
80      }
81
82    case "exec":
83      cmd := exec.Command(args[0], args[1:]...)
84      output, err := cmd.CombinedOutput()
85      if err != nil {
86        term.Write([]byte("Error running command: " + err.Error()))
87      }
88      term.Write(output)
89
90    default:
91      term.Write([]byte("Unknown command: " + command + "\n"))
92
93    }
94  }
```

Листинг 5: Файл main.go

```
1  package main
2
3  import (
4    "fmt"
5    "net"
6    "os"
7  )
8
9  // var WORKING_FOLDER = "/home/chinalap/Документы/ComputerNetworks/lab4
       .1/src/"
10 var WORKING_FOLDER string
11 var USER_LOGIN_FILE = "clients.json"
12 var SSH_PORT = ":2288"
13
14 func main() {
15   WORKING_FOLDER, _ = os.Getwd()
16    if WORKING_FOLDER[len(WORKING_FOLDER)-1] != '/' {
17      WORKING_FOLDER += "/"
```

```
18    }
19
20    s := "127.0.0.1"
21    s, _ = getLocalIP()
22    fmt.Println("Server started")
23    fmt.Printf("Command to connect:\nssh -p %s %s -o StrictHostKeyChecking
       =no\n", SSH_PORT[1:], s)
24    startServer()
25  }
26
27  func getLocalIP() (string, error) {
28    interfaces, err := net.Interfaces()
29    if err != nil {
30      return "", err
31    }
32
33    for _, iface := range interfaces {
34      if iface.Flags&net.FlagUp == 0 || iface.Flags&net.FlagLoopback != 0
       {
35        continue // Ignore down interfaces and loopback interface
36      }
37
38      addrs, err := iface.Addrs()
39      if err != nil {
40        return "", err
41      }
42
43      for _, addr := range addrs {
44        var ip net.IP
45
46        switch v := addr.(type) {
47        case *net.IPNet:
48          ip = v.IP
49        case *net.IPAddr:
50          ip = v.IP
51        }
52
53        // Filter for IPv4 addresses only
54        if ip != nil && ip.To4() != nil {
55          return ip.String(), nil
56        }
57      }
58    }
59    return "", fmt.Errorf("no IP address found")
60  }
```

Часть 2. Разработка SSH-клиента

Задача 1: Реализовать ssh-клиент и запустить его на localhost.

Задача 2: Протестировать соединение Go SSH-клиента к серверу реализованному в предыдущей задаче, а также к произвольному ssh серверу.

Требования: SSH-клиент должен поддерживать следующие функции:

- авторизация клиента на SSH-сервере;
- создание директории на удаленном SSH-сервере;
- удаление директории на удаленном SSH-сервере;
- вывод содержимого директории;
- перемещение файлов из одной директории в другую;
- удаление файла по имени;
- вызов внешних приложений, например ping.

Реализация

Я создал программу, которая обеспечивает соединение до SSH сервера и эмулирует терминал main.go.

Код

Листинг 6: Файл main.go

```
1  package main
2
3  import (
4    "fmt"
5    "io"
6    "log"
7    "os"
8    "os/signal"
9    "syscall"
10
11   "github.com/creack/pty"
```

```
12    "golang.org/x/crypto/ssh"
13    "golang.org/x/term"
14  )
15
16  // SSHConfig holds the SSH connection configuration
17  type SSHConfig struct {
18    User     string
19    Host     string
20    Port     int
21    Password string
22  }
23
24  // NewSSHClient initializes an SSH client and connects to the server
25  func NewSSHClient(config SSHConfig) (*ssh.Client, error) {
26    sshConfig := &ssh.ClientConfig{
27      User: config.User,
28      Auth: []ssh.AuthMethod{
29        ssh.Password(config.Password),
30      },
31      HostKeyCallback: ssh.InsecureIgnoreHostKey(), // For simplicity,
      ignore host key verification
32    }
33
34    address := fmt.Sprintf("%s:%d", config.Host, config.Port)
35    client, err := ssh.Dial("tcp", address, sshConfig)
36    if err != nil {
37      return nil, fmt.Errorf("failed to connect to SSH server: %w", err)
38    }
39
40    return client, nil
41  }
42
43  // StartInteractiveShell starts an interactive shell session on the SSH
      server
44  func StartInteractiveShell(client *ssh.Client) error {
45    session, err := client.NewSession()
46    if err != nil {
47      return fmt.Errorf("failed to create SSH session: %w", err)
48    }
49    defer session.Close()
50
51    // Get the current terminal state for restoration after the session
      ends
52    fd := int(os.Stdin.Fd())
53    oldState, err := term.MakeRaw(fd)
54    if err != nil {
```

```
55      return fmt.Errorf("failed to set terminal raw mode: %w", err)
56    }
57    defer term.Restore(fd, oldState) // Restore terminal state on exit
58
59    // Set up standard input, output, and error streams
60    session.Stdout = os.Stdout
61    session.Stderr = os.Stderr
62    session.Stdin = os.Stdin
63
64    // Request a pseudo-terminal
65    rows, cols, err := pty.Getsize(os.Stdin)
66    if err != nil {
67      return fmt.Errorf("failed to get window size: %w", err)
68    }
69    if err := session.RequestPty("xterm-256color", rows, cols, ssh.
       TerminalModes{
70      ssh.ECHO:          1,
71      ssh.TTY_OP_ISPEED: 14400,
72      ssh.TTY_OP_OSPEED: 14400,
73    }); err != nil {
74      return fmt.Errorf("failed to request pseudo-terminal: %w", err)
75    }
76
77    // Handle window size changes
78    go func() {
79      ch := make(chan os.Signal, 1)
80      signal.Notify(ch, syscall.SIGWINCH)
81      for range ch {
82        rows, cols, _ := pty.Getsize(os.Stdin)
83        session.WindowChange(rows, cols)
84      }
85    }()
86    signal.Notify(make(chan os.Signal, 1), syscall.SIGWINCH) // Catch
       SIGWINCH
87
88    // Start the interactive shell
89    if err := session.Shell(); err != nil {
90      return fmt.Errorf("failed to start shell: %w", err)
91    }
92
93    // Wait for the session to end
94    if err := session.Wait(); err != nil && err != io.EOF {
95      return fmt.Errorf("session error: %w", err)
96    }
97
98    return nil
```

```
99 }
100
101 func main() {
102   // sshConfig := SSHConfig{
103   //   User:        "root",
104   //   Host:        "185.102.139.169",
105   //   Port:        22,
106   //   Password:    "w3Bt8hjge8oV",
107   // }
108
109   sshConfig := SSHConfig{
110     User:     "user",
111     Host:     "185.102.139.169",
112     Port:     2288,
113     Password: "123",
114   }
115
116   client, err := NewSSHClient(sshConfig)
117   if err != nil {
118     log.Fatalf("Error creating SSH client: %v", err)
119   }
120   defer client.Close()
121
122   // Start the interactive shell
123   if err := StartInteractiveShell(client); err != nil {
124     log.Fatalf("Error starting interactive shell: %v", err)
125   }
126 }
```

Вывод программы

После запуска серверной части и подключение к ней через клиентскую, все необходимые функции работали.

Вывод

В этот лабораторной я реализовал свои SSH клиент и сервер. Теперь я умею создавать удаленные консоли на языке GO на основе протокола SSH.