Programming Assignment #9

Introduction to Computer Networks

The Assignment

`PA9.go` is light. We will, simply, extend the `PA8.go` to a secure Web file server. This is straightforward. One just needs to know the APIs to establish secure sockets, i.e., the **TLS sockets**.

1. Setting up the Public/Private Key

Before trying the examples out, download the public and private key file from http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/PA9/server.cer and http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/PA9/server.key. Put these two files in your PA9 directory (where you'll place your `PA9.go`).

2. Secure HTTP using `tls.Listen()`

This example will be more relevant to those who took the native string approach for PA7 and PA8. Start a file `sec-string-Response.go` and type up the following.

```
package main

import "fmt"
import "bufio"
import "net/http"
import "crypto/tls"

func check(e error) {
    if e != nil {
        panic(e)
    }
}
```

```
func main() {
    cert, _ := tls.LoadX509KeyPair("server.cer", "server.key")
    config := tls.Config{Certificates: []tls.Certificate{cert}}

    fmt.Println("Launching server...")
    ln, _ := tls.Listen("tcp", ":<your port#>", &config)
    defer ln.Close()
    conn, _ := ln.Accept()
    defer conn.Close()

    reader := bufio.NewReader(conn)
    req, _ := http.ReadRequest(reader)
    fmt.Printf("Method: %s\n", req.Method)

    fmt.Fprintf(conn, "HTTP/1.1 404 Not Found\r\n")
    fmt.Fprintf(conn, "Date: ...\r\n")
    fmt.Fprintf(conn, "\r\n")
    fmt.Fprintf(conn, "File not found\r\n")
    fmt.Fprintf(conn, "\r\n")
}
```

Replace `<your port#>` with the port number assigned to your team. Start the `sec-string-Response.go` first. Then, `curl -k` an arbitrary Web object.

```
$ curl -k https://127.0.0.1:<your port#>/
File not found

$
```

Code walk-through:

- tls.Listen(), a new API defined in the `crypto/tls` package, is equivalent of net.Listen() to start secure communication.

- The line `tls.Config{Certificates: []tls.Certificate{cert}}` configures the TLS socket. The minimum requirement is to specify the `Certificates` field (in the `tls.Config` structure). The `Certificates` field is defined as an array of certificate chains. Each certificate chain is of `tls.Certificate` type. The array of certificate chains is therefore `[]tls.Certificate`. In the example, the array's got only 1 element, containing only 1 certificate chain – `cert`.
- `tls.LoadX509KeyPair()` takes two parameters, the public and private key file. The public key file is used to generate the certificate in PEM form following the X509 standard. That certificate in PEM form is kept in `cert` and used next to configure the TLS socket in `tls.Config`.
- In the example, `server.cer` is server's public key file. `cert` is therefore the server certificate. The certificate will be sent to the client when the client attempts to connect. The client can then verify the certificate through a certificate authority (CA). When the CA approves the certificate, the server public key is used by the client to encrypt the subsequent handshake messages, which only the server will be able to decrypt (with the server's private key), therefore maintaining the confidentiality and preventing man-in-the-middle attacks.
- Note though the public key (`server.cer`) needs to be registered to the CA to allow certificate verification from clients worldwide. The CA needs to be well administered (by an NPO usually) to be trustworthy. To sustain, a service that demands secure message exchange pays the CA to keep the server's certificate alive and valid.
- As the `server.cer` used for the example is not registered to any CA, https to the server process will fail the certificate verification. That is why we use the `-k` flag in `curl` to bypass the process.
- Alternatively, start a Firefox browser and request for https://127.0.0.1:<your port#>/. The browser will complain the access might not be secure and ask if we wish to proceed anyway. If we do proceed, Firefox records the certificate as an exception and allows accesses in the future.

3. Secure HTTP using `ListenAndServeTLS()`

For those who build on Golang’s built-in Web server, see if you can borrow from this example. Start `sec-handler-Response.go` with the code block below.

```
package main

import "fmt"
import "net/http"

func helloHandler(w http.ResponseWriter, r *http.Request) {
   fmt.Fprintln(w, "Hello, world!")
}

func main() {
   fmt.Println("Launching server...")

   hh := http.HandlerFunc(helloHandler)
   http.Handle("/hello", hh)
   fs := http.FileServer(http.Dir("."))
   http.Handle("/", http.StripPrefix("/", fs))
   http.ListenAndServeTLS(":<your port#>", "server.cer",
"server.key", nil)
}
```

Start `sec-handler-Response.go` first. Then, `curl -k` from another terminal. The output of the following `curl -k` commands should be similar to those from `handler-Response.go` in PA8.

```
$ curl -k https://127.0.0.1:<your port#>/hello
$ curl -k https://127.0.0.1:<your port#>/sec-string-Response.go
$ curl -k https://127.0.0.1:<your port#>/qwerty.htm
```

Code walk-through:

- The only API new is `http.ListenAndServeTLS()`. It is the equivalent of `http.ListenAndServe()` to start secure communication. The usage is slightly different as it takes the public and private key file as inputs as well.

4. PA9.go

Make sure your `PA9.go` is listening on the port number you are assigned to. To test your `PA9.go`, use `curl` and `curl -k` to see if the responses are different. Use at least Mozilla Firefox to request a text/html file existing and non-existing from the server's home directory.

On your laptop/desktop, download and install a traffic sniffing tool such as tcpdump (Unix-based) or Wireshark (Windows). Sniff the traffic running between your server and client. One should see the messages in plain text running the `PA8.go` server, as opposed to being encrypted running `PA9.go`.

To help you verify your implementation, polly has made the compiled byte code of her `PA9.go` (native string solution) and `PA9-http.go` (`FileServer()` wrapper approach) available here: http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/PA9/PA9 and http://homepage.ntu.edu.tw/~pollyhuang/teach/intro-cn-pa/PA9/PA9-http. Again, polly's `PA9` and `PA9-http` are configured to run on port# 11999.

5. Submit your PA9

`ssh` to the `140.112.42.161` workstation. At the team account's home directory, create a directory `PA9`. Upload your `PA9.go` to directory `PA9`. Test your `PA9.go` again on the workstation just to make sure it's working as expected.