xd media

Realizado por Mario Davó

Índice

Introducción

El proyecto consiste en hacer nuestra propia versión de "X", anteriormente conocida como Twitter, donde utilizando Golang y encriptando la información entre cliente y servidor según los métodos descritos en clase, los usuarios clientes podrán leer posts y mantener conversaciones en privado.

Puesta en marcha

Para llevar a cabo la práctica, esta se ha llevado a cabo en Linux con nuestros portátiles personales.

Software necesario

Base de datos

- Docker
- Docker Compose

Backend, Logs, Cliente

- Go

Ejecución

Necesitarás mínimo cuatro terminales.

Para iniciar la base de datos, sitúate en `/db`

```
docker-compose up -d
docker exec -i db psql -U admin -d xdm< ./seed.sql # Seedear la base de datos con datos de ejemplo

# Utilitarios
docker exec -i db psql -U admin -d xdm < ./clear.sql # Limpiar todas las tablas
docker exec -it db psql -U admin -d xdm # Acceder al contenedor
```

Existe un Makefile en el directorio fuente, con él se puede poner en marcha el servidor de logs y el backend. Estos son los comandos del Makefile explicados. Ejecutar desde `/`

```
make logger     # Compila el sistema de logs
make logger_run # Compila y ejecuta el sistema de logs
make server     # Compila el servidor backend
make server_run # Compila y ejecuta el servidor backend
```

Para ejecutar el cliente ir al directorio `/client`. Hay tres clientes:

- `/no_auth`
- `/auth`
- `/chat`

Situarse en cada carpeta y ejecutar

```
go run main.go
```

Funcionalidades

Mecanismo de autentificación (gestión de contraseñas, identidades y sesión)

Se ha implementado la seguridad mediante TLS.

```
func run() error {
  var err error
  addr := fmt.Sprintf(":%d", SERVER_PORT)

  db, err = connectDB()
  if err != nil {
    Error(err.Error())
    Error("Unsuccessful database connection attempt")
    os.Exit(1)
  }
  defer db.Close()

  mux := http.NewServeMux()
  addRoutes(mux)
  handler := corsMiddleware(mux)

  cfg := &tls.Config{
    MinVersion:               tls.VersionTLS13,
    CurvePreferences:         []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP256},
    PreferServerCipherSuites: true,
    InsecureSkipVerify:       true,
    CipherSuites: []uint16{
      tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
      tls.TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
      tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
      tls.TLS_RSA_WITH_AES_256_CBC_SHA,
    },
  }
  srv := &http.Server{
    Addr:         addr,
    Handler:      handler,
    TLSConfig:    cfg,
    TLSNextProto: make(map[string]func(*http.Server, *tls.Conn, http.Handler), 0),
  }
  Info("listening on addr %s", addr)
  Fatal(srv.ListenAndServeTLS("crypto/server.crt", "crypto/server.key").Error())
  return nil
}

func main() {
  if err := run(); err != nil {
    Fatal(err.Error())
  }
}
```

Usando la implementación de la libraría estándar (`HandleFunc`) ligamos las rutas a las funciones que las manejan, y a aquellas que requieran autenticación se envuelven a través de la función `authMiddleware()`.

```
func addRoutes(mux *http.ServeMux) {
  mux.HandleFunc("POST /signup", signupHandler)
  mux.HandleFunc("POST /signin", signinHandler)
  mux.HandleFunc("GET /users/me", authMiddleware(userMeHandler))
  mux.HandleFunc("GET /users", getUsersHandler)
  mux.HandleFunc("POST /users/{user_id}/follow", authMiddleware(userFollowHandler))
  mux.HandleFunc("DELETE /users/{user_id}/follow", authMiddleware(userUnfollowHandler))
  mux.HandleFunc("POST /posts", authMiddleware(createPostHandler))
  mux.HandleFunc("GET /posts", getPostsHandler)
  mux.HandleFunc("GET /users/{user_id}", getUserHandler)
  mux.HandleFunc("GET /posts/{post_id}", getPostHandler)
  mux.HandleFunc("POST /posts/{post_id}/like", authMiddleware(likePostHandler))
  mux.HandleFunc("DELETE /posts/{post_id}/like", authMiddleware(deleteLikePostHandler))
  mux.HandleFunc("GET /users/{user_id}/like", authMiddleware(userLikesHandler))
  mux.HandleFunc("GET /ws", handleWebSocket)
}
```

Para la autenticación empleamos tokens JWT.

Transporte de red seguro entre cliente y servidor

Se ha implementado transporte seguro TLS entre cliente y servidor, además de HTTPS para la conexión con el cliente.

Almacenamiento seguro

cifrado en descansoSe ha empleado una extensión para PostgreSQL *pg_tde GitHub*.

Sistema de gestión de contenido general

(público)El usuario puede ver posts de otros usuarios. Se puede probar con el cliente de ejemplo no_auth

Sistema de comunicación privado (cifrado) entre usuarios

Se ha implementado un chat donde cada usuario posee su clave pública y clave privada. Cada usuario posee su clave púb./priv. El servidor mantiene la sincronización de claves públicas entre usuarios, cuando un usuario envía un texto, se cifra con la clave pertinente y se redirige a el destinatario.

```
for clientID, publicKey := range connectedClients {
    if clientID == userId {
      continue
    }
    encryptedContent, err := encryptMessage([]byte(content), publicKey)
    if err != nil {
      fmt.Printf("Error encrypting message for client %s: %v\n", clientID, err)
      continue
    }
    signature, err := signMessage([]byte(content), privateKey)
    if err != nil {
      fmt.Printf("Error signing message for client %s: %v\n", clientID, err)
      continue
    }
    encryptedContentBase64 := base64.StdEncoding.EncodeToString(encryptedContent)
    signatureBase64 := base64.StdEncoding.EncodeToString(signature)

    message := Message{
      Type:             "message",
      Sender:           userId,
      Recipient:        clientID,
      EncryptedContent: encryptedContentBase64,
      Signature:        signatureBase64,
    }

    err = conn.WriteJSON(message)
    if err != nil {
      fmt.Printf("Error sending message to client %s: %v\n", clientID, err)
    }
}
```

(Opcional) Sistema de registro de eventos (logging), para mejorar la trazabilidad (remoto)

El backend hace logs tanto a su terminal como a un servidor remoto dedicado a logging.
Las funciones para hacer logging en el servidor:

```
func Info(format string, v ...interface{}) {
  payload := fmt.Sprintf(format, v...)
  InfoLog.Printf(payload)
  sendLog(payload)
}

func Warning(format string, v ...interface{}) {
  payload := fmt.Sprintf(format, v...)
  WarningLog.Printf(payload)
  sendLog(payload)
}

func Error(format string, v ...interface{}) {
```

```
  payload := fmt.Sprintf(format, v...)
  ErrorLog.Printf(payload)
  sendLog(payload)
}

func Fatal(format string, v ...interface{}) {
  payload := fmt.Sprintf(format, v...)
  FatalLog.Printf(payload)
  sendLog(payload)
  os.Exit(1)
}
```

En el servidor de logging se recibe cada conexión y se escribe en un fichero.

```
func handleConnection(conn net.Conn) {
  defer conn.Close()
  buf, err := io.ReadAll(conn)
  if err != nil {
    log.Println("Failed to read from connection:", err)
    return
  }

  str := string(buf)
  if _, err := file.WriteString(str + "\n"); err != nil {
    log.Println("Failed to write to log file:", err)
  }
}
```

Cronología aproximada

Primero se realizó el servidor de logging. Posteriormente se creó la base de datos básica para poder ir trabajando y luego empezó el trabajo sobre el servidor. Una vez hubo una base sólida se empezó con la implementación de los clientes.

Ambos nos apoyamos mutuamente para llevar a cabo las tareas, cogíamos lo que más le gustaba a cada uno, a Ignacio el diseño y la experiencia del usuario, mientras que Mario fue el aportador principal al apartado de back-end, apoyado periódicamente por Ignacio.

Expectativas de seguridad y privacidad

Límites de seguridad (en qué condiciones el proyecto es seguro y en cuáles no)

Condiciones seguras

- Concurrencia y Gestión de Memoria

Las goroutines reducen el riesgo de problemas con la gestión de memoria

- Manejo de errores

Manejo explícito de errores

- Lenguaje Tipado Estáticamente

Reduce errores de tipo y posibles vulnerabilidades a ser explotadas por atacantes

- Bibliotecas Estándar Seguras

Operaciones criptográficas, manejo de solicitudes HTTP, implementación de funcionalidades seguras sin depender de bibliotecas externas.

Conclusión

¿ Qué se ha realizado?

Fuimos capaces de llevar a cabo todos los apartados básicos del proyecto más algún que otro apartado opcional como el de logging.

Aspectos positivos y relevantes del proyecto

Lo más entretenido que he realizado ha sido el chat con cifrado de punto a punto. Al principio iba a funcionar entre parejas de usuarios pero acabó siendo un chat grupal/broadcasting.