

Distributed Programming University Project



H.D.S

1st January 2025

Student: Angelo Rosa

Screenshots

Brief

H.D.S (Hashcat-Distributed-Service) is a university project written entirely in **Go**, designed to distribute WPA handshake cracking tasks across multiple **Hashcat** clients. It serves as a practical proof of concept for distributed password-cracking workloads.

Run the Application (in Test Mode) with Docker

Since `client` within docker runs a GUI application using **Raylib**, it requires access to the host's desktop environment. A utility called `xhost` is needed (`xorg-xhost` on Arch Linux).

Run the following commands:

```
export DISPLAY=:0.0 && \  
xhost +local:docker && \  
docker compose up --build
```

Access the **Frontend (FE)** by visiting:

➡ `http://localhost:4748`

Docker Containers

The setup will spawn four containers: - **dp-database** - **dp-server** - **emulate-raspberrypi** - **dp-client**

The provided `docker-compose.yml` file already includes all necessary environment variables for a functional **test environment**. No changes are required to run the project for demonstration purposes.

Default credentials: `admin:test1234`

This account can be used on the frontend to upload and submit WPA handshakes for cracking.

While the software is primarily designed for **Linux**, GPU capabilities can potentially be shared with a containerized `client` via **WSL** on Windows. Future improvements may include native support for additional operating systems.

Project Features

1. Handshake Capturing and Uploading:

Users can capture WPA handshakes using tools like **bettercap** (or similar) and use a **daemon** to upload them to the server. Although referred to as `RaspberryPI` in the project, the daemon can run on any platform supporting **Golang**.

2. Frontend Management:

Users can access the **Frontend (FE)** to:

- View captured handshakes.
- Submit them to clients for cracking.
- Manage connected clients and daemon devices.
- Remove unnecessary handshakes.

3. Independent Clients:

Each **client** operates independently and communicates directly with the server. Users can select which client will handle specific cracking tasks.

4. Modularity:

The software is designed with modularity in mind to simplify future changes and improvements.

Architectural Notes

- While a fully **Clean Architecture** approach wasn't strictly followed, both the **Frontend (FE)** and **Backend (BE)** adopt a similar structure.

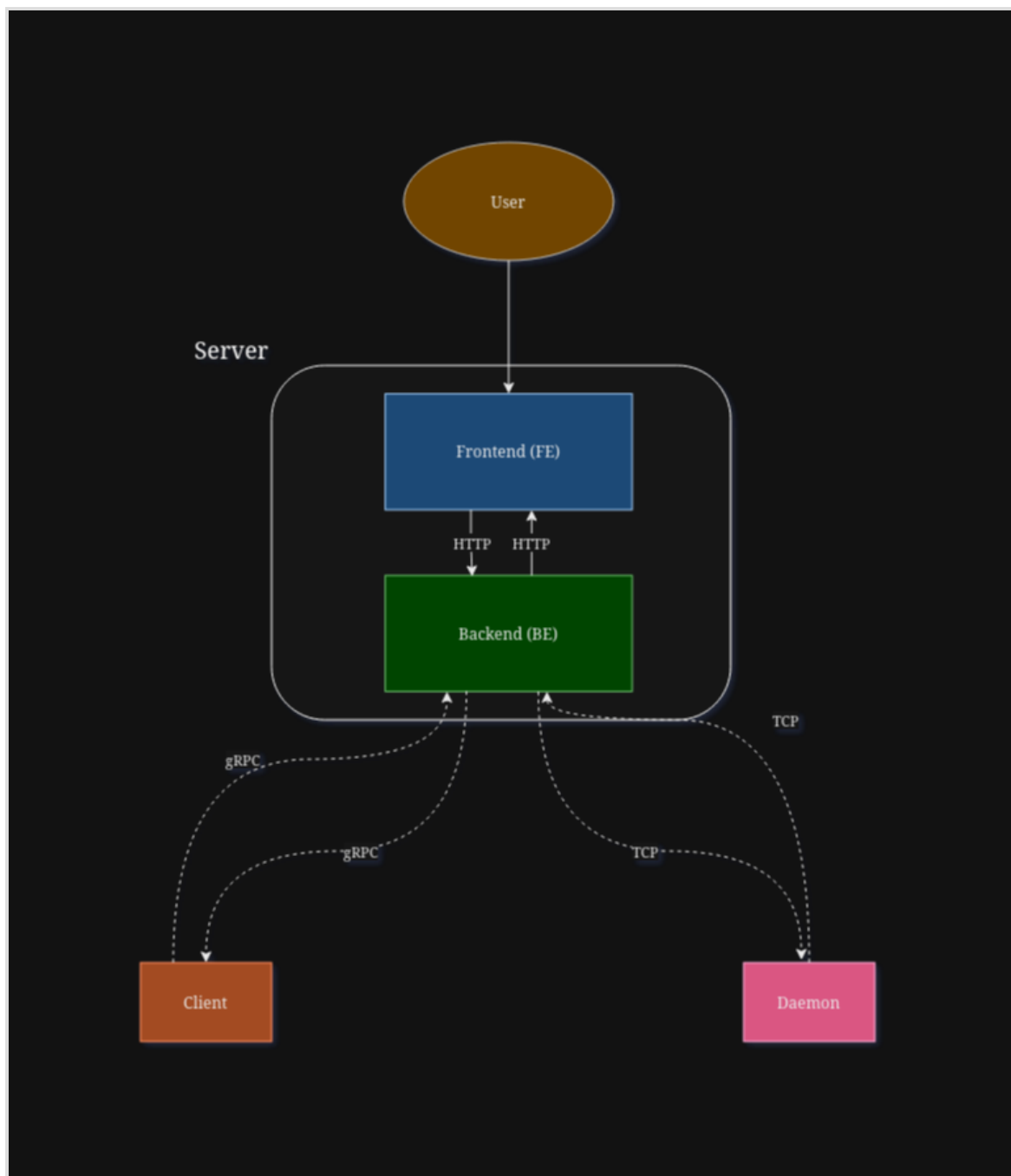
- **Entities (database models)** reside in the `server` folder and are shared between **FE** and **BE**.
 - If FE and BE are deployed on separate servers, the `entities` directory must be moved into each respective folder. Minor refactoring will be required.
- **gRPC Communication:**

Clients and the backend use **gRPC** for communication. Both must include compiled **protobuf** files:

```
cd client && make proto
```

```
cd server/backend && make proto
```

Project Scheme



As shown in the diagram, the **Backend (BE)** is isolated and can only be accessed through the **Frontend (FE)**.

Communication Flow:

- **FE ↔ BE (HTTP/REST API):**
 - **FE:** Sends HTTP requests to BE.
 - **BE:** Handles database interactions and returns data.

- **Daemon ↔ BE (TCP):**
 - After authenticating via **REST API**, the daemon communicates with BE via raw **TCP**.
- **Client ↔ BE (gRPC):**
 - A **bidirectional gRPC stream** allows clients to dynamically send logs and receive updates during **Hashcat** operations.

Directory Mapping:

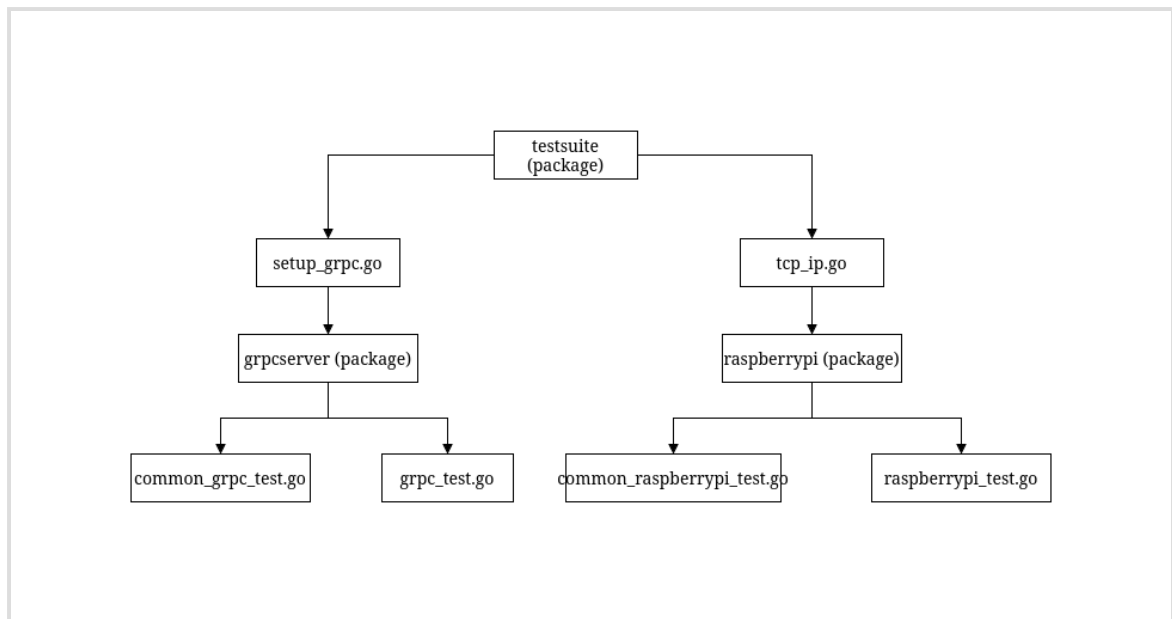
- **Client:** -> /client
 - **Daemon:** -> /raspberry-pi
 - **Server:** -> /server
 - **Backend:** -> /server/backend
 - **Frontend:** -> /server/frontend
-

Test scheme

The tests have been implemented in the backend, emulating `gRPC` and `daemon` clients to test out the methods of these 2 protocols.

When running a test, a `gRPC` server or `tcp` server, along with the rest API, is initialized. It depends on what tests you're running.

It was not that easy to achieve a good solid test scheme, but in the end, it should look something like this.



- `setup_grpc.go` -> set up a grpc mock server to communicate with. It is initialized during `SetupSuite`. The server is killed on `TeardownSuite`. A rest API server is initialized too.
- `tcp_ip.go` -> Set up a tcp mock server to communicate with. It is initialized during `SetupSuite`. The server is killed on `TeardownSuite`. A rest API server is initialized too.
- `common_grpc.go` -> calls `SetupSuite` and `TeardownSuite`, define a structure with mock data useful in tests
- `common_raspberrypi_test.go` -> calls `SetupSuite` and `TeardownSuite`, define a structure with mock data useful in tests
- `grpc_test.go` -> contains tests for grpc infrastructure
- `raspberrypi_test.go` -> contains tests for raspberrypi infrastructure

A database container must be up and running on port 3306 to run tests.

You can run tests using `cd server && make test` but env variables must be set before proceeding

```
export BACKEND_HOST="0.0.0.0"
export BACKEND_PORT="4747"
export FRONTEND_HOST="0.0.0.0"
export FRONTEND_PORT="4748"
```



```
export DB_USER="agent"
export DB_PASSWORD="SUPERSECUREUNCRAKABLEPASSWORD" # This
               should be changed (remember to change it in
               database/initialize.sql too)
export DB_HOST="localhost"
export DB_PORT="3306"
export DB_NAME="dp_hashcat"
export ALLOW_REGISTRATIONS="True" # Disable if needed
export DEBUG="True" # This will enable seeds for having
               some accounts for testing purposes. admin:test1234
               will be created
export RESET="True"
export GRPC_URL="0.0.0.0:7777"
export GRPC_TIMEOUT="10s"
export TCP_ADDRESS="0.0.0.0"
export TCP_PORT="4749"
```

Security and Future Improvements

While security auditing and privacy were not primary objectives for this project, some measures and considerations have been noted:

1. Encryption:

- Currently, **pcap files** sent by the daemon are **not encrypted**.
- A symmetric encryption key has been generated, but encryption is yet to be implemented.

2. Daemon Authentication:

- Daemon authenticates via **REST API** before establishing a **TCP** connection.
- Credentials are sent via command-line arguments, which could be stolen easily if a malicious actor have access remotely to the machine.

3. gRPC Security:

- gRPC communication currently lacks **SSL/TLS certificates** for encryption.

Security Measures Implemented:

- Basic protection against vulnerabilities like **SQL Injection** and **IDORs** has been considered.

If you have suggestions or improvements, feel free to **open a pull request**.

Project files

List of files

```
├─ client
│   ├── Dockerfile
│   ├── go.mod
│   ├── go.sum
│   ├── internal
│   │   ├── constants
│   │   │   └─ constants.go
│   │   ├── entities
│   │   │   ├── auth_request.go
│   │   │   └─ handshake.go
│   │   ├── environment
│   │   │   └─ init.go
│   │   ├── grpcclient
│   │   │   ├── communication.go
│   │   │   └─ init.go
│   │   ├── gui
│   │   │   ├── login_window.go
│   │   │   └─ process_window.go
│   │   ├── hcxtools
│   │   │   └─ hcxpcapngtool.go
│   │   ├── mygocat
│   │   │   ├── gocat.go
│   │   │   └─ task_handler.go
│   │   ├── resources
│   │   │   └─ fonts
│   │   │       ├── Roboto-BlackItalic.ttf
│   │   │       ├── Roboto-Black.ttf
│   │   │       ├── Roboto-BoldItalic.ttf
│   │   │       ├── Roboto-Bold.ttf
│   │   │       └─ Roboto-Italic.ttf
```

```
| | | └─ Roboto-LightItalic.ttf
| | | └─ Roboto-Light.ttf
| | | └─ Roboto-MediumItalic.ttf
| | | └─ Roboto-Medium.ttf
| | | └─ Roboto-Regular.ttf
| | | └─ Roboto-ThinItalic.ttf
| | | └─ Roboto-Thin.ttf
| | └─ utils
| | └─ utils.go
| └─ main.go
| └─ Makefile
| └─ wordlists
└─ database
    └─ Dockerfile
    └─ initialize.sql
    └─ my.cnf
└─ docker-compose.yaml
└─ externals
    └─ gocat
    └─ hashcat
    └─ hcxtools
└─ LICENSE
└─ proto-definitions
    └─ hds
        └─ hds.proto
        └─ hds_request.proto
        └─ hds_response.proto
└─ proto.sh
└─ raspberry-pi
    └─ Dockerfile
    └─ go.mod
    └─ go.sum
    └─ handshakes
        └─ test.pcap
    └─ internal
        └─ authapi
            └─ authenticate.go
        └─ cmd
```

```
|   |   |   └─ command_parser.go
|   |   └─ constants
|   |   |   └─ constants.go
|   |   └─ daemon
|   |   |   └─ communication.go
|   |   |   └─ environment.go
|   |   |   └─ init.go
|   |   └─ entities
|   |   |   └─ api_entities.go
|   |   |   └─ handshake.go
|   |   └─ utils
|   |   |   └─ utils.go
|   |   └─ wifi
|   |   |   └─ wifi.go
|   |   └─ wpaparser
|   |       └─ getwpa.go
|   |       └─ parser.go
|   └─ main.go
|   └─ Makefile
└─ README.md
└─ server
    └─ backend
        └─ cmd
            └─ main.go
            └─ internal
                └─ constants
                └─ constants.go
                └─ errors
                └─ errors.go
                └─ grpcserver
                └─ commands.go
                └─ common_grpc_test.go
                └─ controllers.go
                └─ grpc_test.go
                └─ init.go
                └─ options.go
                └─ infrastructure
                └─ database.go
```

```
|      |└─ raspberrypi
|      |  |└─ common_raspberrypi_test.go
|      |  |└─ components.go
|      |  |└─ init.go
|      |  |└─ raspberrypi_test.go
|      |  |└─ tcp_server.go
|      |└─ repository
|      |  |└─ repository.go
|      |└─ response
|      |  |└─ response.go
|      |└─ restapi
|      |  |└─ authenticate
|      |  |  |└─ handler_anonymous.go
|      |  |  |└─ handler_user.go
|      |  |└─ client
|      |  |  |└─ handler_user.go
|      |  |└─ handlers.go
|      |  |└─ handshake
|      |  |  |└─ handler_user.go
|      |  |└─ logout
|      |  |  |└─ handler_user.go
|      |  |└─ middlewares
|      |  |  |└─ auth_middleware.go
|      |  |  |└─ common_middleware.go
|      |  |  |└─ log_requests.go
|      |  |└─ raspberrypi
|      |  |  |└─ handler_user.go
|      |  |└─ register
|      |  |  |└─ anonymous_handler.go
|      |  |└─ routes.go
|      |└─ seed
|      |  |└─ seed_api.go
|      |└─ testsuite
|      |  |└─ auth_api.go
|      |  |└─ setup_grpc.go
|      |  |└─ tcp_ip.go
|      |└─ usecase
|      |  |└─ usecase.go
```

```
|      └─ utils
|      └─ utils.go
|      └─ validator.go
|─ Dockerfile
|─ entities
|  └─ client.go
|  └─ handshake.go
|  └─ raspberry_pi.go
|  └─ role.go
|  └─ uniform_response.go
|  └─ user.go
|─ frontend
|  └─ cmd
|     └─ custom.go
|     └─ main.go
|     └─ internal
|        └─ constants
|           └─ constants.go
|           └─ errors
|              └─ errors.go
|              └─ middlewares
|                 └─ auth_middleware.go
|                 └─ cookie_middleware.go
|                 └─ log_requests.go
|                 └─ pages
|                    └─ clients
|                       └─ clients.go
|                       └─ handshakes
|                          └─ handshake.go
|                          └─ login
|                             └─ login.go
|                             └─ logout
|                                └─ logout.go
|                                └─ pages.go
|                                └─ raspberrypi
|                                   └─ raspberrypi.go
|                                   └─ register
|                                      └─ register.go
```

```
| | | | └─ routes.go
| | | | └─ welcome
| | | |   └─ welcome.go
| | | └─ repository
| | |   └─ repository.go
| | └─ response
| |   └─ response.go
| └─ usecase
|   └─ usecase.go
└─ utils
    └─ utils.go
        └─ validator.go
└─ static
    └─ images
        └─ logo.png
    └─ scripts
        └─ bootstrap.min.js
        └─ dashboard.js
        └─ github-stats.js
        └─ jquery-3.3.1.min.js
        └─ popper.min.js
        └─ theme-toggle.js
    └─ static.go
    └─ styles
        └─ bootstrap-4.3.1.min.css
        └─ custom.css
        └─ main.css
└─ views
    └─ clients.html
    └─ handshake.html
    └─ login.html
    └─ raspberrypi.html
    └─ register.html
    └─ views.go
    └─ welcome.html
└─ go.mod
└─ go.sum
└─ main.go
```

└─ Makefile

79 directories, 143 files

Daemon

The Raspberry Pi component is designed to **send network captures performed by bettercap to the server**. Its functionality is straightforward: you run the `bettercap` daemon on the Raspberry Pi, and it automatically transmits captured handshakes to the server whenever your local Wi-Fi SSID is detected nearby.

This feature is **disabled** if the `TEST` environment variable is set to `False`. Ensure that your Raspberry Pi has at least **two network interfaces**, with one interface always connected to a stable network.

Tested on Raspberry Pi 5 Model B Rev. 2

Update the `wlan1` interface to match your network interface. Make sure your Wi-Fi card supports **monitor mode** and **packet injection**.

What Does the Daemon Do?

The daemon performs the following tasks:

1. Acts as a **TCP/IP client** to establish raw network connections.
2. Scans all `.PCAP` files located in the `~/handshakes` directory (typically where `bettercap` saves handshakes).
3. Utilizes the `gopacket` library to read `.PCAP` file layers, extracting **BSSID** and **SSID** information, and verifying if a **valid 4-way handshake** exists.

4. If a valid handshake is detected, the daemon **encodes the file in Base64** and sends it to the server.
 5. Waits for a predefined **delay period** before repeating the process.
-

Run Bettercap

This configuration sets up `bettercap` to capture handshakes and save them in the correct directory.

```
sudo bettercap -iface wlan1 -eval 'set
    wifi.handshakes.aggregate false; set
    wifi.handshakes.file ~/handshakes; wifi.recon on;
    set wifi.show.sort clients desc; set ticker.commands
    "wifi.deauth *; clear; wifi.show"; set ticker.period
    60; ticker on';
```

The daemon uses the `HOME` directory as its base. Since `bettercap` requires `sudo`, you must run the daemon as `root`.

Other Useful Bettercap Commands

These commands can help you fine-tune your `bettercap` setup:

```
sudo bettercap -iface wlan1
wifi.recon BBSID
wifi.recon on
wifi.recon.channel N; # N is the channel to recon
```

Compile and Run the Daemon

Make sure the following requirements are met before building and running the daemon:

The daemon requires `libpcap0.8-dev` to be installed on your system.

The file `/etc/machine-id` must exist on your machine.

Follow these steps to compile and run the daemon:

```
cd raspberry-pi
go mod verify
go mod tidy
go build main.go
sudo ./main
```

Client

Clients communicate with the server using `gRPC`. This enables clients to identify whether they are the intended receiver for a specific cracking task.

The communication channel also supports a **bidirectional stream**, allowing the server and client to exchange messages in real time during `hashcat` execution.

What Does the Client Do?

A client performs the following tasks:

1. **Waits for tasks** from the server.
2. Upon receiving a task, it **acknowledges the server**.
3. The server then **removes the task from the pending queue** and updates its status. Meanwhile, the client saves the **base64-encoded hash file** into a temporary directory.
4. Once saved as a **.PCAP file**, the client converts it into a **hash format compatible with hashcat**.
5. The client uses `hcxtools` for the conversion. This library supports multiple operations on `.PCAP` files and beyond.
6. After conversion, **hashcat begins execution**, applying user-defined or default options.
7. **Logs and status updates** generated by `hashcat` are sent asynchronously to the server.

8. If `hashcat` successfully cracks the password, the **result is sent back to the server**.
 9. The client then resets itself and **waits for the next task**.
-

Gocat

The client uses the `gocat` dependency to execute `hashcat` from within Go. Since `hashcat` is written in **C**, a **porting layer** was required to bridge the two environments.

Hcxttools

For our use case, we rely specifically on `hcxpcapngtool` from the `hcxttools` suite.

This tool doesn't natively support building as a shared library. To work around this limitation and enable its integration with Go, we **modified its entry point** using `sed` :

```
sed -i 's/int main(int argc, char \*argv\[ \]) /int
      convert_pcap(int argc, char \*argv\[ \])/'
      hcxpcapngtool.c
```

This command replaces the standard `main` function signature with `convert_pcap`. We then compile it into a shared library:

```
cc -fPIC -shared -o /app/client/libhcxpcapngtool.so /app/
      hcxttools/hcxpcapngtool.c -lz -lssl -lcrypto -
      DVERSION_TAG=\"6.3.5\" -DVERSION_YEAR=\"2024\"
```

This shared library can now be directly imported and used in Go:

File: `client/internal/hcxttools/hcxpcapngtool.go`

```
/*
#cgo LDFLAGS: -L../.. -lhcxpcapngtool
#include <stdlib.h>
```

```

// Declare the convert_pcap function from the shared library
int convert_pcap(int argc, char *argv[]);
*/
import "C"
import (
    "fmt"
    "unsafe"
)

func ConvertPCAPToHashcatFormat(inputFile, outputFile string)
    error {
    // Prepare arguments for the convert_pcap function
    args := []string{"", inputFile, "-o", outputFile}
    argc := C.int(len(args))
    argv := make([]*C.char, len(args))

    // Convert Go string slices to C strings
    for i, arg := range args {
        argv[i] = C.CString(arg)
        defer C.free(unsafe.Pointer(argv[i]))
    }

    // Call the convert_pcap function from the shared
    library
    ret := C.convert_pcap(argc, &argv[0])
    if ret != 0 {
        return fmt.Errorf("hcxpcapngtool conversion failed
            with code %d", ret)
    }

    return nil
}

```

While this solution works for our current requirements, future improvements could include **porting the library fully to Go**. However, this is considered **out of scope** for the current project.

Compile and Run

The following dependencies are required before proceeding

```
apt update -y && \
  apt install -y --no-install-recommends \
  libminizip-dev \
  ocl-icd-libopencl1 \
  opencl-headers \
  pocl-opencl-icd \
  build-essential \
  wget \
  git \
  dumb-init \
  ca-certificates \
  libz-dev \
  libssl-dev \
  dbus \
  # Graphic libraries for raylib
  libgl1-mesa-dev libxi-dev libxcursor-dev libxrandr-dev
  libxinerama-dev libwayland-dev libxkbcommon-dev
```

The file `/etc/machine-id` must exist on your machine.

Follow these steps to compile and run the client, run it from project root dir

```
git submodule init
git submodule update
git pull --recurse-submodule
```

Then

```
BASE=${PWD}

# Environment Variables
export HASHCAT_SRC_PATH="${BASE}/client/hashcat"
```

```
export CGO_CFLAGS="-I$HASHCAT_SRC_PATH/OpenCL -  
I$HASHCAT_SRC_PATH/deps/LZMA-SDK/C -I$HASHCAT_SRC_PATH/deps/  
zlib -I$HASHCAT_SRC_PATH/deps/zlib/contrib -  
I$HASHCAT_SRC_PATH/deps/OpenCL-Headers $CGO_CFLAGS"  
  
# Proto:  
cd client  
make proto  
  
cd ${BASE}/externals/hashcat  
git checkout v6.1.1  
  
cd ${BASE}  
  
# Directories  
mkdir -p "${BASE}/client/hashcat" "${BASE}/client/gocat" "${  
{BASE}/client/hcxttools"  
  
cp -r externals/hashcat/* "${BASE}/client/hashcat/"  
cd "${BASE}/client/hashcat"  
sudo make install SHARED=1 ENABLE_BRAIN=0  
sudo cp deps/LZMA-SDK/C/LzmaDec.h /usr/local/include/  
hashcat/  
sudo cp deps/LZMA-SDK/C/7zTypes.h /usr/local/include/  
hashcat/  
sudo cp deps/LZMA-SDK/C/Lzma2Dec.h /usr/local/include/  
hashcat/  
sudo cp -r OpenCL/inc_types.h /usr/local/include/hashcat/  
sudo cp -r deps/zlib/contrib /usr/local/include/hashcat  
  
sudo ln -sf /usr/local/lib/libhashcat.so.6.1.1 /usr/local/  
lib/libhashcat.so  
sudo ln -sf /usr/local/lib/libhashcat.so.6.1.1 /usr/lib/  
libhashcat.so.6.1.1  
  
cd ${BASE}
```

```
cp -r externals/gocat/* "${BASE}/client/gocat/"
cd "${BASE}/client/gocat"
go test -c
sudo cp gocat.test /usr/local/share/hashcat
sudo cp -r testdata /usr/local/share/hashcat
/usr/local/share/hashcat/gocat.test

cd ${BASE}

sudo chown -R ${USER}:${USER} "${BASE}/client"
sudo chown -R ${USER}:${USER} /usr/local/share/hashcat

ln -sf /usr/local/share/hashcat/hashcat.hcstat2 "${BASE}/client/hashcat.hcstat2"
ln -sf /usr/local/share/hashcat/hashcat.hctune "${BASE}/client/hashcat.hctune"
ln -sf /usr/local/share/hashcat/OpenCL "${BASE}/client/OpenCL"
ln -sf /usr/local/share/hashcat/kernels "${BASE}/client/kernels"
ln -sf /usr/local/share/hashcat/modules "${BASE}/client/modules"

cp -r externals/hcxttools/* "${BASE}/client/hcxttools/"
sed -i 's/int main(int argc, char \*argv\[ \]) /int
convert_pcap(int argc, char \*argv\[ \]) /' "${BASE}/client/
hcxttools/hcxpcapngtool.c"
cc -fPIC -shared -o "${BASE}/client/libhcxpcapngtool.so" "${
BASE}/client/hcxttools/hcxpcapngtool.c" -lz -lssl -lcrypto -
DVERSION_TAG=\"6.3.5\" -DVERSION_YEAR=\"2024\"

cd "${BASE}/client"
go mod verify
go mod tidy
go build main.go
```


Run:

```
export GRPC_URL=localhost:7777
export GRPC_TIMEOUT=10s
export DISPLAY=${DISPLAY} # Pass display variable from
terminal
export LD_LIBRARY_PATH=client/:$LD_LIBRARY_PATH
./main
```

Produces files tree

```
├─ hashcat
├─ gocat      <-- From /externals/gocat
├─ hcxtools   <-- From /externals/hcxtools
├─ client
│   └─ main
│   └─ hashcat.hcstat2 (symlink)
│   └─ hashcat.hctune (symlink)
│   └─ OpenCL (symlink)
│   └─ kernels (symlink)
│   └─ modules (symlink)
│   └─ libhcxpcapngtool.so
```

Server

The **Server** is divided into two main components:

- **Backend**
 - **Frontend**
-

What Does the Backend Do?

The **backend** performs the following tasks:

1. Initializes a connection with the **database**.
2. Starts a basic `HTTPServer` to expose the **REST API**, potentially creating **seeds/mock data** for testing purposes.
3. Initializes a `gRPC` server to handle communication with **clients**.

4. Initializes a `TCP` server to handle communication with **daemons**.
 5. Encapsulates the **core application logic queue**.
-

What Does the Frontend Do?

The **frontend** performs the following tasks:

1. Starts a basic `HTTPServer` and parses **template files** to expose a user interface.
 2. Accepts **user inputs** and communicates with the **backend** using `REST API` by performing **HTTP requests**.
-

Compile and Run

You need to export the following **environment variables**. Customize them as needed.

1. Start Database

```
cd database
docker build -t dp-database .
docker run -d \
--name dp-database \
-e MYSQL_RANDOM_ROOT_PASSWORD=yes \
--restart unless-stopped \
-p 3306:3306 \
--health-cmd="mysqladmin ping -h localhost -uagent -pSUPERSECUREUNCRACKABLEPASSWORD" \
--health-interval=20s \
--health-retries=10 \
dp-database
```

2. Export Environment Variables

```
export BACKEND_HOST="0.0.0.0"
export BACKEND_PORT="4747"
export FRONTEND_HOST="0.0.0.0"
export FRONTEND_PORT="4748"
export DB_USER="agent"
export DB_PASSWORD="SUPERSECUREUNCRACKABLEPASSWORD" # This
should be changed (remember to change it in database/
initialize.sql too)
export DB_HOST="localhost"
export DB_PORT="3306"
export DB_NAME="dp_hashcat"
export ALLOW_REGISTRATIONS="True" # Disable if needed
export DEBUG="True" # This will enable seeds for having
some accounts for testing purposes. admin:test1234 will be
created
export RESET="True"
export GRPC_URL="0.0.0.0:7777"
export GRPC_TIMEOUT="10s"
export TCP_ADDRESS="0.0.0.0"
export TCP_PORT="4749"
```

3. Compile and Run the Server

```
cd server
make proto
go mod tidy
go build main.go
./main
```

After completing these steps, the **server** should be up and running, with both the **frontend** and **backend** components functioning as expected.

External Dependencies

Ignoring gRPC and other basic deps

- **RayLib** github.com/gen2brain/raylib-go/raylib A basic graphic library
- **Gocat** github.com/mandiant/gocat/v6 Used for running hashcat in go via [hashcatlib](#)
- **Validator** github.com/go-playground/validator/v10 Validator for go structures
- **Mux** github.com/gorilla/mux HTTP router
- **Testify** github.com/stretchr/testify A test library for simplifying test syntax
- **Gopacket** github.com/google/gopacket Parse [.PCAP](#) files as layers
- **Wifi** github.com/mdlayher/wifi used by daemon for understanding if we're connected to our local network
- **Cobra** github.com/spf13/cobra used for parsing command line arguments easily in daemon