Distributed Programming University Project

(image AI generated)
H.D.S
1st January 2025

Student: Angelo Rosa

Screenshots

Brief

> **H.D.S (Hashcat-Distributed-Service)** is a university project entirely written in **Go** that can find practical applications for collecting and then distributing hashes and handshakes cracking tasks on multiple hashcat clients through a control panel centre. Include a Daemon for collecting and verifying WPA handshakes from IOT devices. You can have multiple clients for multiple tasks and each one operates independently on different machines.

For more information about the project capabilities, please read the [feature section](#) You can find releases compiled at: https://github.com/Virgula0/H.D.S/releases

But remember to read the following compiling procedures anyway

- [Compile Daemon](#)
- [Compile Server](#)
- [Compile Client](#)

To understand how to set up env variables.

Run the Application (Test Mode) with Docker

A docker environment can be run to see everything up and running without dealing with compilations or environment setups first. 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 for this.

> This works with `X11` only at the moment

1. Installation

```
sudo apt update -y && \
sudo apt install git-lfs && \
sudo apt install x11-xserver-utils -y
```

Or on arch:

```
sudo pacman && \
sudo pacman -S xorg-xhost

git submodule init && \
git submodule update && \
git pull --recurse-submodules && \
# download wordlists
git lfs install && \
git lfs pull && \
# change display values as you need
export DISPLAY=:0.0 && \
xhost +local:docker && \
docker compose up --build
```

1. 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 be shared with to `client` using **WSL** on Windows. Future improvements may include native support for additional operating systems but at the moment compiling or running the docker file within a **WSL** works just fine. In order to use GPUs capabilities, be sure to share resources to **WSL** with nvidia-cuda packages.

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**. The daemon can travel around the world capturing handshakes and when it comes back at home, recognized home WiFI and tries to send captured handshakes to the server

2. **Frontend Management:**
   Users can access the **Frontend (FE)** to:

   - View captured handshakes.
   - Delete captured handshakes.
   - Upload other generic hash files regardless Daemon's captures.
   - Submit tasks to clients for cracking.
   - Manage connected clients and daemon devices.

3. **Independent Clients:**
   Each **client** operates independently and communicates directly with the server. Users can select which client will handle specific cracking tasks. Clients have a minimal GUI which allows to visualize the status of process without accessing the FE directly.

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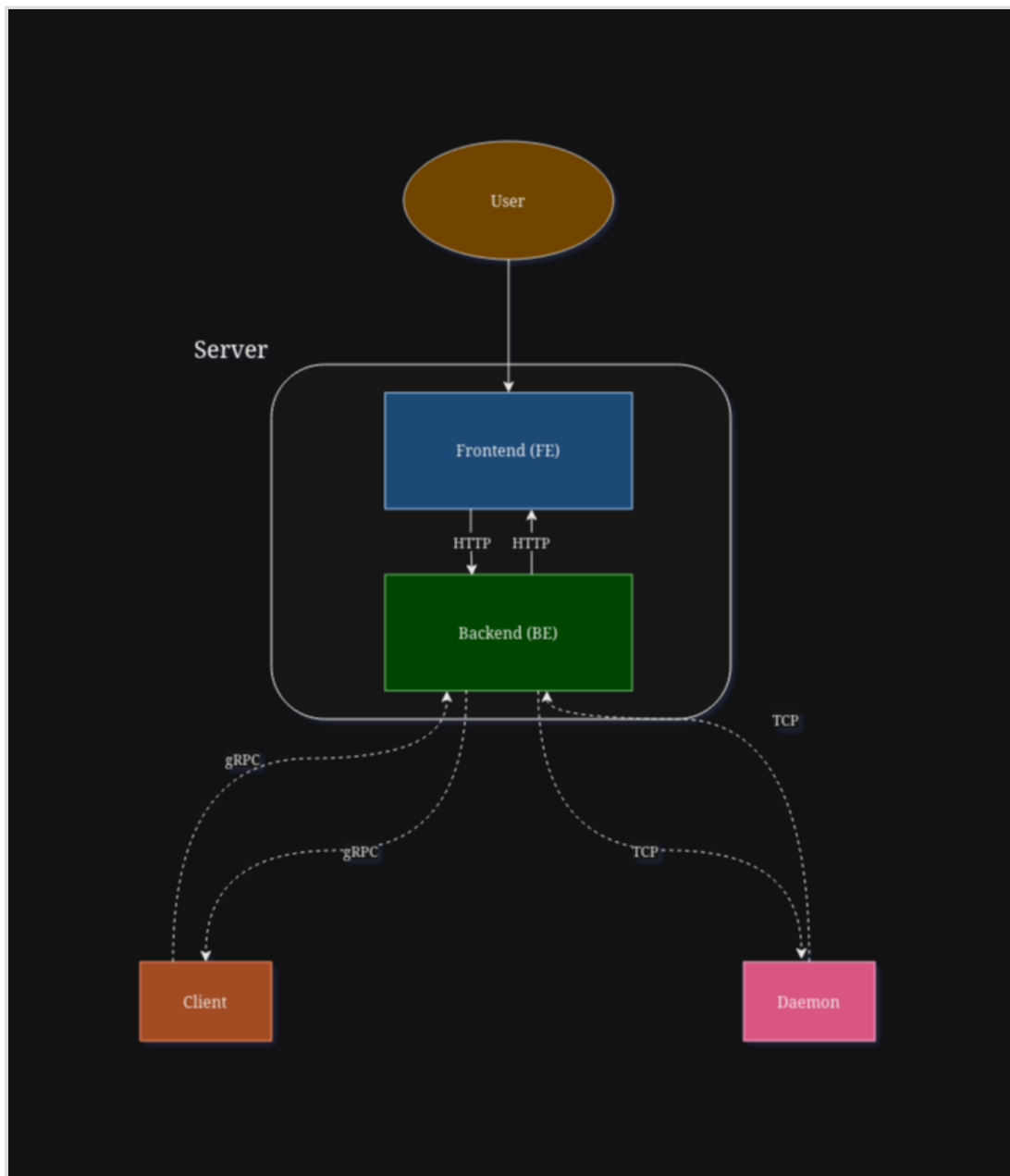
- **gRPC Communication:**
  Clients and the backend use **gRPC** for communication. Both must include compiled **protobuf** files:

```
cd client && make proto
cd server && 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.
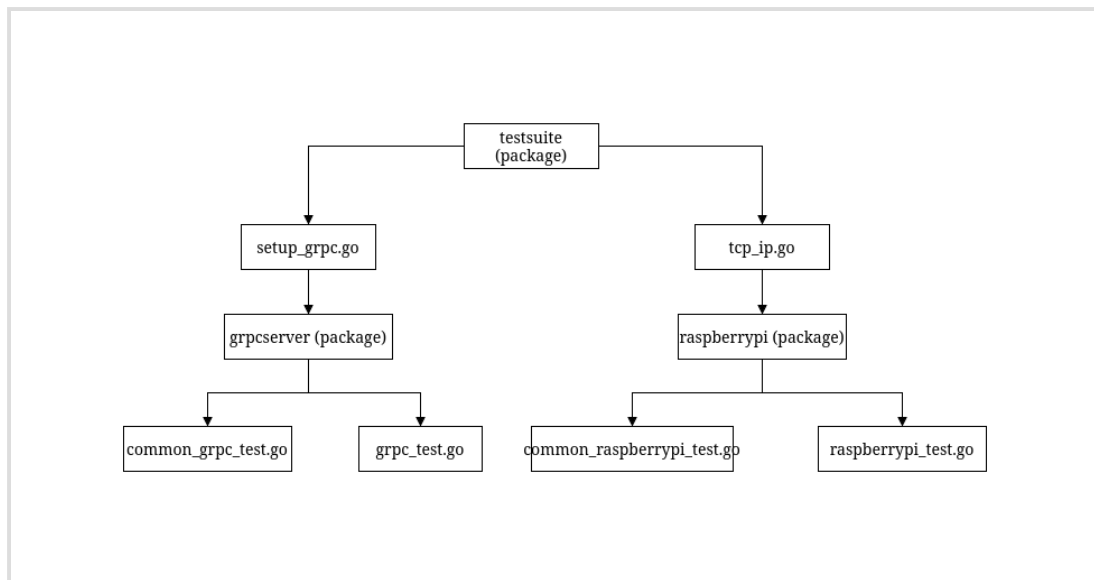
Directory Mapping:

- **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.

testsuite
(package)

setup_grpc.go

tcp_ip.go

grpcserver (package)

raspberrypi (package)

common_grpc_test.go

grpc_test.go

common_raspberrypi_test.go

raspberrypi_test.go

- `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

Take a look at Server to check environment variables needed for running the server.

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.~~ Fixed with https://github.com/Virgula0/H.D.S/pull/42
   - ~~Credentials are sent via command-line arguments, which could be stolen easily if a malicious actor have access remotely to the machine.~~ Fixed with https://github.com/Virgula0/H.D.S/pull/39
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
│   │   ├── customerrors
│   │   │   └── errors.go
│   │   ├── entities
│   │   │   ├── auth_request.go
│   │   │   ├── client.go
│   │   │   └── handshake.go
│   │   ├── environment
│   │   │   └── init.go
│   │   ├── grpcclient
│   │   │   ├── communication.go
│   │   │   └── init.go
│   │   ├── gui
│   │   │   ├── login_window.go
│   │   │   ├── message_window.go
│   │   │   └── process_window.go
│   │   ├── hcxtools
│   │   │   └── hcxpcapngtool.go
│   │   ├── mygocat
│   │   │   ├── gocat.go
│   │   │   └── task_handler.go
│   │   └── utils
│   │       └── utils.go
│   ├── main.go
│   ├── Makefile
│   ├── README.md
│   ├── resources
│   │   ├── fonts
│   │   │   ├── JetBrainsMono-Regular.ttf
│   │   │   └── Roboto-Black.ttf
│   │   └── resources.go
│   └── wordlists
│       └── rockyou.txt
├── database
│   ├── Dockerfile
│   ├── initialize.sql
│   └── my.cnf
├── docker-compose.yaml
├── docs
│   ├── diagram.drawio
```

```
|       ├── docs.pdf
|       ├── images
|       |       ├── logo.png
|       |       ├── project-structure.png
|       |       ├── Screenshots
|       |       |       ├── 1.png
|       |       |       ├── 2.png
|       |       |       ├── 3.png
|       |       |       ├── 4.png
|       |       |       └── 5.png
|       |       └── test-diagram.png
|       ├── styles
|       |       └── main.css
|       └── test-diagram.drawio
├── externals
|       ├── gocat
|       └── 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
|       |       ├── cmd
|       |       |       └── command_parser.go
|       |       ├── constants
|       |       |       └── constants.go
|       |       ├── daemon
|       |       |       ├── communication.go
|       |       |       ├── environment.go
|       |       |       └── init.go
|       |       ├── entities
```

```
|   |   |   ├── api_entities.go
|   |   |   └── handshake.go
|   |   ├── enums
|   |   |   └── daemon_enums.go
|   |   ├── tui
|   |   |   └── login_tui.go
|   |   ├── utils
|   |   |   └── utils.go
|   |   ├── wifi
|   |   |   └── wifi.go
|   |   └── wpaparser
|   |       ├── getwpa.go
|   |       └── parser.go
|   ├── main.go
|   ├── Makefile
|   └── README.md
├── README.md
├── server
|   ├── backend
|   |   ├── cmd
|   |   |   └── main.go
|   |   └── internal
|   |       ├── constants
|   |       |   └── constants.go
|   |       ├── enums
|   |       |   └── tcp_server.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_middlware.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_operation.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
|   └── README.md
└── version

85 directories, 156 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, even if you're using compiled binaries from releases.

```
        sudo apt update -y && \
        sudo apt install -y libpcap0.8-dev
```

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

1. **Compile daemon with**

```
cd raspberry-pi
make build
```

1. **Run with**

```
./build/daemon run
```

1. Insert your credentials

But remember to export these env var first, change them according to your needs

```
export SERVER_HOST=localhost
export SERVER_PORT=4747
export TCP_ADDRESS=localhost
export TCP_PORT=4749
export TEST=False
export HOME_WIFI=Vodafone-A60818803 # Change with your SSID of
your home Wireless Network
export BETTERCAP=True
```

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.

Hcxtools

For our use case, we rely specifically on `hcxpcapngtool` from the `hcxtools` 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/hcxtools/
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/hcxtools/hcxpcapngtool.go`

```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 needs to be installed before proceeding, even if you're using compiled binaries from releases

```
sudo apt update -y && \
sudo apt install -y --no-install-recommends \
    protobuf-compiler \
    libminizip-dev \
    ocl-icd-libopencl1 \
    opencl-headers \
    git \
    git-lfs \
    pocl-opencl-icd \
    build-essential \
    ca-certificates \
    libz-dev \
    libssl-dev \
    dbus \
    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 update --init --remote --recursive && \
git lfs install && \
git lfs pull
```

1. **You need to install `hashcat` 6.1.1. This step is necesary only for the first time.**

```
cd externals/gocat
sudo make install
sudo make set-user-permissions USER=${USER}
cd ../../
```

1. **Install protobuf**

> This was tested out using go `1.23.4`. Other version may have problems.

```
go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.31.0 &&
go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.3.0
```

1. **Build client**

```
cd client
make build
```

Produces the following files tree in `build`

```
├── client
├── hashcat.hctune -> /usr/local/share/hashcat/hashcat.hctune
├── hashcat.hcstat2 -> /usr/local/share/hashcat/hashcat.hcstat2
├── libhcxpcapngtool.so
├── modules -> /usr/local/share/hashcat/modules
└── OpenCL -> /usr/local/share/hashcat/OpenCL
```

1. **Run with**

```
make run-compiled
```

but remember to set these env variables first

```
export GRPC_URL=localhost:7777 # change with gRPC address
export GRPC_TIMEOUT=10s #leave this timeout by default
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

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
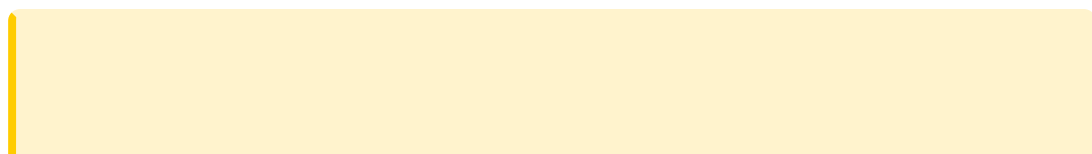
1. Start Database

```
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

For simplicity you can save these into a `.env` and the `source .env`
Change them accoding to your needs.

```
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 DB_CERT="dp_certs"
export DB_CERT_USER="certs"
export DB_CERT_PASSWORD="SUPERSECUREUNCRACKABLEPASSWORD" # This
should be changed (remember to change it in database/
initialize.sql too)
export ALLOW_REGISTRATIONS="True" # Disable if needed
```

```
        export DEBUG="True"  # leave to true this
        export RESET="False" # set to false when running the server
normally otherwise at each server restart data will be wiped from
the db
        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 build
```

4. Run the Server

```
        ./build/server
```

After completing these steps, the **server** should be up and running, with both the **frontend** and **backend** components functioning as expected. Frontend will be available to `FRONTEND_PORT` port value.

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
- **bubbletea** `github.com/charmbracelet/bubbletea` tui for login implemented in daemon
- Other dependencies could be implicitly downloaded and used because of these deps