Kraken

Kraken is a security testing tools used in networking, it is developed to be extendible by design.

Its main task is to orchestrate security testing modules against network targets (e.g ip:port). Doing only the recoinnassance and orchestrating tasks, kraken's modules API is structured to be very permissive, infact to the modules are passed configuration parameters and the target (ip:port). From that point on it's the module resposibility to handle everything from the connection logic to the testing logic. This gives a high level of independence for what can be done inside modules but also overload the developer with more work.

Architecture

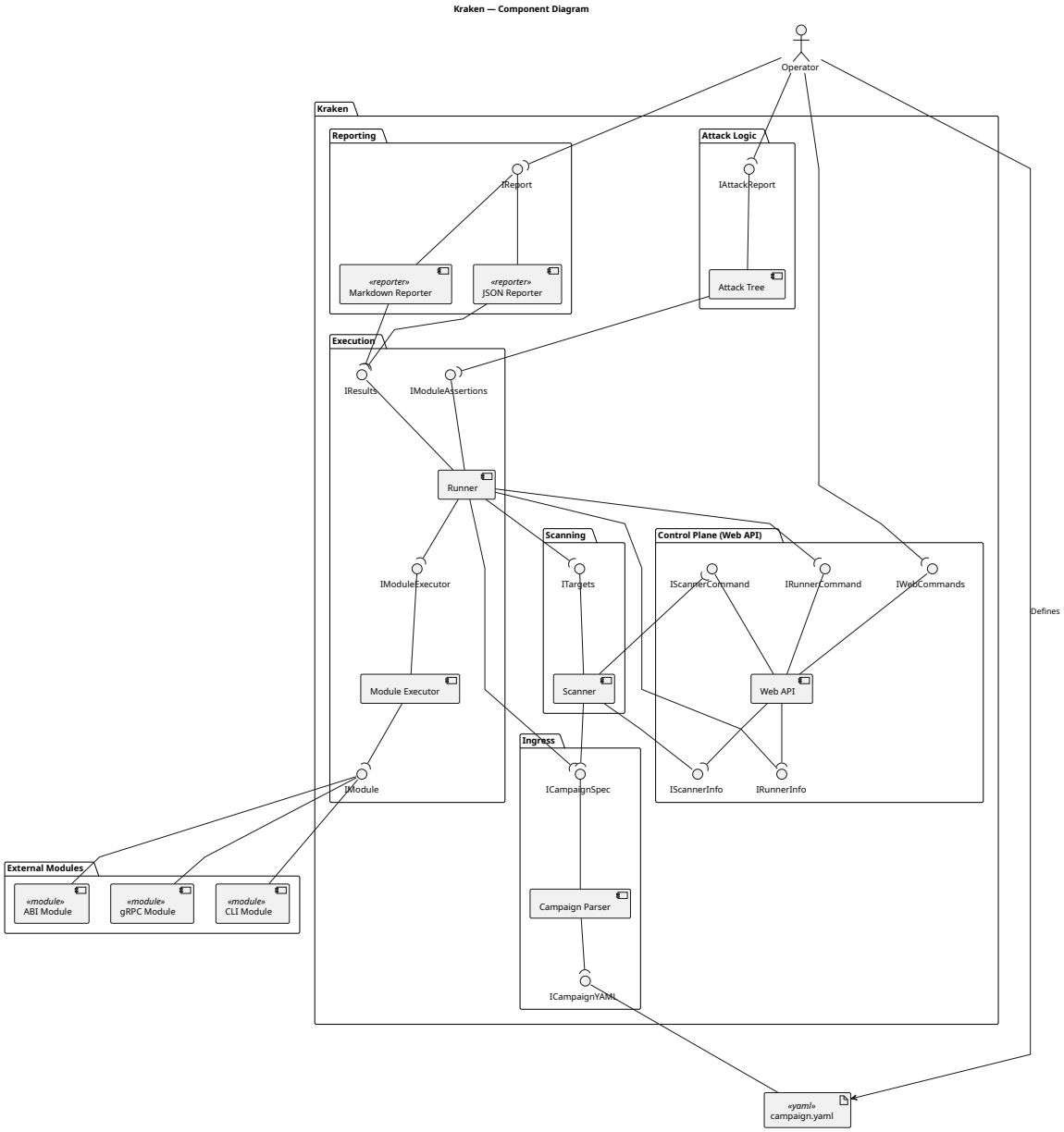

Figure 1: architecture

Campaign

The program is configured with a yaml file, this file defines what is called a *campaign*, the campaign define how the application behave and what it should do. The campaign is divided in four main sections:

- **General options**: in this section there are general information about the campaign:

```yaml
id: "iot-standard"
version: "2.0.0"
```

```yaml
name: "IoT Network Assessment"
description: |
    General description of the campaing, what it does, etc...

attack_trees_def_path: "./trees/iot.yaml"
```

- **Runner configuration**: in this section the configuration options for the runner can be configured, for now it contains only the `max_parallel_targets`:

```yaml
runner:
    max_parallel_targets: 16
```

- **Scanner configuration**: kraken uses nmap under the hood, as such the options are the one from nmap. An option that is not of nmap that can be setted is `timeout`, defines the maximum time after which the scanner will be interrupted by kraken:

```yaml
scanner:
    open_only: true
    skip_host_discovery: false
    enable_udp: false
    service_detect:
        enabled: true
        version: "ALL"
    min_rate: 100
    timeout: 30m
    timing: T3
    ports:
        - "1883,8883,8083,8084,80,443,8000,8080,8443,8888,502,4840,554,8554"
```

- **Tasks configuration**: this section define the modules to run and with which options to run them. As it can be seen a module is defined by: `id`, the `required_tags` that trigger its running on a target, `max_duration`, `type` that can be of three types lib/grpc/cli (will be explained later on), `exec` this varies from type of module and defines the runtime parameters.

```yaml
tasks:
    - id: "lib-module-api_v1"
      required_tags: ["supports:tls", "protocol:tcp"]
      max_duration: 15s
      type: lib
      exec:
          abi:
              api: v1
              library_path: "/path/to/lib(.dll/.so/.dylib)"
              symbol: kraken_run
          params:
              key1: value1
              key2: value2
              keyn: valuen

    - id: "lib-module-api_v2"
```

```yaml
      required_tags: ["supports:tls", "protocol:tcp"]
      max_duration: 20s
      type: lib
      exec:
          abi:
              api: v2
              library_path: "/path/to/lib(.dll/.so/.dylib)"
              symbol: "kraken_run_v2"
          conduit:
              kind: 1
              stack:
                  - name: tcp
                  - name: tls
                    params:
                        skip_verify: true
          params:
              key1: value1
              key2: value2
              keyn: valuen

    - id: "cli-module"
      required_tags: ["protocol:mqtt"]
      max_duration: 30m
      type: cli
      exec:
          # The orchestrator will to set the --output-dir parameter
automatically
          # so that the outputs of the binary will be placed in the right
directory.
          cli:
              exec: "/path/to/exec"
              command: "fuzz" # If necessary
          # Params will be translated in arguments, entry, e.g.:
          #     --test-case-index: 10-20
          # will be translated in: <exec> <command> --test-case-index: 10-20
          params:
              key1: value1
              -key2: value2
              --keyn: valuen

    - id: "grpc-module"
      required_tags: ["protocol:mqtt"]
      max_duration: 30m
      type: grpc
      exec:
          grpc:
              server_addr: 127.0.0.1:5053
              dial_timeout: 30s
          params:
```

```
key1: value1
key2: valuen
keyn: valuen
```

This configuration gives us flexibility and is easy to understand.

Modules

Modules come in three flavors-lib, gRPC, and CLI so Kraken stays flexible. It can be distributed by running gRPC modules on remote machines for heavy tasks, or keep everything local by using lib plugins on a single host.

LIB Modules

The lib modules can be implemented with two APIs:

- The **v1** ABI will require the module to handle everything, also the transport settings. If the module needs to use TLS then with this API the module developer needs to handle him/herself.
- The **v2** ABI will instead lift the ownership of the transport implementation from the developer, infact with the v2 there are some standard transports that can be used by the plugin (the used transport needs to be specified in the configuration file). There are some limitations such as this type of modules can not enstablish multiple connections in one session therefore attacks such as a dictionary attack can not be implemented.

The types of supported transports are:

- *tcp* and *tls*
- *udp* and *dtls*

Will be supported:

- *EtherCAT* and other ethernet protocols

CLI Modules

The cli modules have only one version, they take the target as input and run the test against it.

> Being on the same machine the CLI module can always reach the target found by Kraken.

GRPC Modules

The grpc modules have only one version, they take the target as input and run the test against it.

> The ABI module, being distributed, could be on a different machine. The developer should make sure that a certain target can be reached from the machine the module is installed.