MAD

unknown

Dec 12, 2023

CONTENTS

This is a collection of articles about setting up and running MAD.

CHAPTER

ONE

WHAT IS THIS?

Map'A'Droid (MAD) is a system to collect data from the game. It uses real Android devices. MAD is able to scan for raids, quests, mon and their IVs.

1.1 MAD

MAD is the software running on your server. Devices connect to it and will then be controlled by the server.

1.2 Android devices

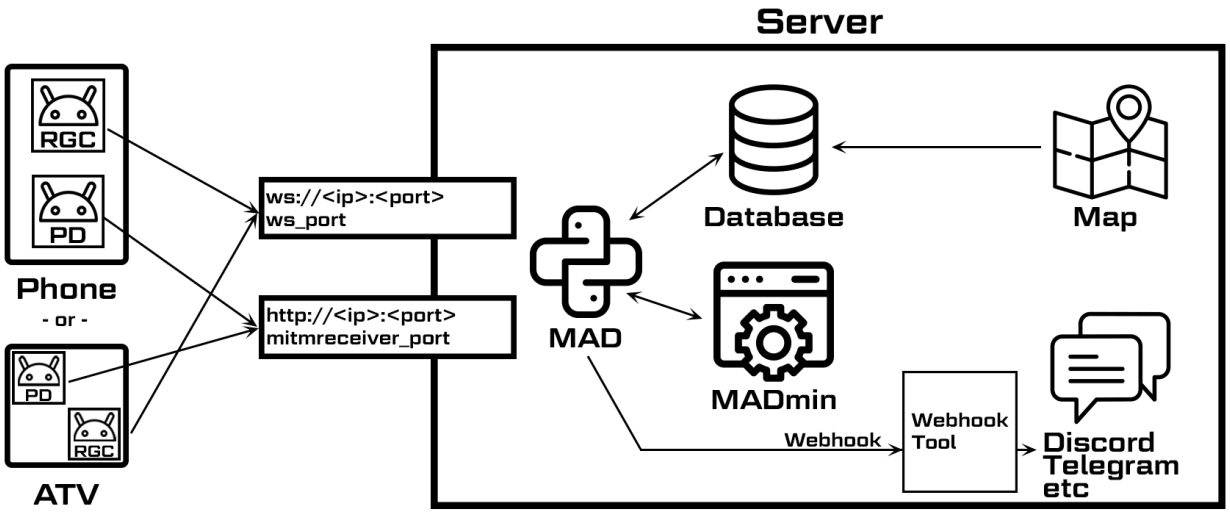
The device needs run the game, must be rooted, pass the SafetyNet Check and SELinux must be permissive or moderate. To root your device, use Magisk.

1.2.1 Remote GPS Controller (RGC)

RGC is an app developed by the MAD team to control the device in various ways. It handles the GPS spoofing, touch/text inputs, app starts/stops etc.

1.2.2 PogoDroid (PD)

This app injects into the running game process and relays the data being sent to the game to the MAD server. A valid token is required to use it, you can purchase it on the MAD website.


Server
RGC
PD
Phone
- or -
PD
RGC
ATV
ws://<ip>:<port>
ws_port
http://<ip>:<port>
mitmreceiver_port
MAD
Database
Map
MADmin
Webhook
Webhook
Tool
Discord
Telegram
etc

CHAPTER

TWO

CONTRIBUTING TO THIS WIKI

If you want to contribute to this wiki, create a pull request to the GitHub repository.

The files are either markdown (`.md`) or reStructuredText (`.rst`). Here is a cheatsheet for RST formatting, and one for markdown. If you created a new section, make sure to add it to the toctree below this section right here.

To preview your changes, make sure to install the requirements (`pip install -r requirements.txt` and `pip install sphinx-autobuild`), open a terminal and use `make auto`. Those commands works on Windows and Linux. This will start a local webserver on port 8000 with live updates pages as you save them.

2.1 Installation

This section will discuss about the different deployment options for Map'A'Droid and hardening the system. It is important to think about how you will be securing MAD before you start your installation

2.1.1 Manual Installation

Requirements

MAD requires the following things to be installed available on your server:

- A server/computer running Linux. RaspberryPis do work, but aren't recommended
- A 64-bit CPU for MAD is also highly recommended since some optional parts of MAD do require to run on 64-bit. It does have a fallback for 32-bit CPUs though
- MariaDB server
- Python 3.9 or newer and Python's package manager command line tool `pip3`.
- Use a `venv` (`virtualenv`) to install dependencies. Have a look at this and this if you're new to `virtualenv`

System preparation

Note: This whole article assumes a fresh installed Ubuntu 22.04 Server. If you're running a more recent version of Ubuntu or another Linux distribution - that's totally fine, but keep in mind there may be some difference in your setup.

First let's install all the needed packages from Ubuntu repository - there will be more description later what they do, but for now let's just install everything in one go to save time - if future *apt install* commands returns *XXX is already the newest version* then it's good - we already have that one.

```
sudo apt update
sudo apt install mariadb-server default-libmysqlclient-dev mariadb-client python3-venv␣
↪python3-pip python3-wheel python3-dev tesseract-ocr python3-opencv redis build-
↪essential pkg-config
```

MariaDB

You need a Database with full permissions. That DB can be located on a different Server, but needs to be accessible by your MAD server.

If you are planning to use PMSF as a webfrontend: use at least MariaDB 10.2 or higher!

```
sudo apt install mariadb-server mariadb-client
sudo mysql_secure_installation
```

Log in to your Database and create a dedicated user for MAD. To create a new database and grant permissions for your dedicated MAD database user you must run the following command (make sure to change "my_database_user" and "password" to the correct values):

```
CREATE DATABASE my_database_name;
CREATE USER 'my_database_user'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON my_database_name.* TO 'my_database_user'@'localhost';
FLUSH PRIVILEGES;
```

Install client libraries

```
sudo apt install default-libmysqlclient-dev
```

Database schema

MAD will install the latest database schema automatically on initial boot and no additional steps are required. If you encounter any issues the base schema can be found here.

The up-to-date models used by MAD can viewed in mapadroid/db/model.py.

Python

Install additional python dependencies:

```
sudo apt install python3-pip python3-wheel python3-dev
```

Make sure you have the right version installed, since even if python3.9 is installed, the *python3* command could still point to *python3.5* or below! `ls -lah $(which python3)` will show the current symlink of Python

Check if *pip* and *python* is installed correctly by running:

- `python3 --version` - should return 3.9.x or newer (3.10.X etc)
- `pip3 --version` - If it returns a version that is related to your python version, it is working.

Virtual Environment

A virtual environment is a way to install python packages in a different location to avoid potential version conflicts with other software like RocketMAD or MADevice. It's like a standalone version of python, independent of your "normal" python. Install it with:

```
sudo apt install python3-venv
```

MAD

MAD will also check the screen on your device every now and then to check for errors. Make sure you have the required dependencies installed on your system. Unfortunately, there's no package for opencv on RaspberryPi which means you have to build it on your own. You should be able to find out how with a quick search on the web.

```
sudo apt install tesseract-ocr python3-opencv
```

Steps below should be run as normal, non-privileged user. It's a bad practice to run everything as `root` (Administrator in Windows world). Create a new virtual environment called `mad_env` in your home directory:

```
python3 -m venv ~/mad_env
```

Whenever you see `python3` or `pip3` in the documentation, use `~/mad_env/bin/python3` and `~/mad_env/bin/pip3` instead. And, of course, use a different environment location for different python projects. For example if you are also using RocketMAD - have additional dedicated virtual environment for RocketMAD like `~/rm_env`.

You can activate the virtual environment via *source ~/mad_env/bin/activate*. This makes sure you can simply call *python3* or *pip3* wherever you are and it will perform all commands with the Python version and the dependencies form your virtualenvironment. Have a look at this or this link for more information.

Next Step is to clone this repository and install all the required `pip3` packages:

```
git clone https://github.com/Map-A-Droid/MAD.git
```

Change into in the directory of MAD and run:

```
pip3 install -r requirements.txt
```

Another but optional dependency you may want to install is ortools. MAD utilizes ortools to generate more optimized routes for your areas and it is as quick as MAD's built-in routing algorithm if not even faster. The downside of this as states in *the requirements* is, that you need a 64-bit server.

```
pip3 install ortools
```

Configuration

Copy config.ini.example (from the configs folder in the MAD repo) to "config.ini" (also in the configs folder):

```
cp configs/config.ini.example configs/config.ini
```

and then edit the config file accordingly.

The next step is to configure MAD. This will only start MAD's web frontend called MADmin.

Warning: MAD will not actually scan in configmode! The mode is for the first configuration only. Remove the `-cm` when you are done.

```
python3 start.py
```

By default MADmin will be available on http://your_server_ip:5000.

Running

If everything is set up correctly, you can start MAD:

```
python3 start.py
```

Further steps

Review and implement anything related to the security section.

2.1.2 Docker

Warning: MAD's Docker support is community driven and untested by MAD's core developers!

Set up Docker

If you do not have a clue about docker, you maybe want to check out:

- https://www.docker.com/why-docker
- https://www.docker.com/resources/what-container

First of all, you have to install Docker CE and docker-compose on your system.

- Docker CE: just execute this script - or read through https://docs.docker.com/install/
- Docker-compose: https://docs.docker.com/compose/install

These sites are well documented and if you follow the install instructions, you are good to go.

Setup MAD and RocketMAD database.

In this section we explain how to setup MAD and a RocketMAD database using docker-compose.

Preparations

You can just copy & paste this to do what is written below:

```
mkdir MAD-docker && \
cd MAD-docker && \
mkdir mad && \
mkdir mad/configs && \
mkdir rocketdb && \
touch rocketdb/my.cnf && \
touch docker-compose.yml && \
cd mad/configs/ && \
wget -O config.ini https://raw.githubusercontent.com/Map-A-Droid/MAD/async/configs/
↪config.ini.example && \
cd ../../
```

This will:

1. Create a directory *MAD-docker*.
2. Create a file *docker-compose.yml*.
3. Create a directory *MAD-docker/mad*. (here we store MAD related stuff)
4. Create a directory *MAD-docker/mad/configs*. (here we store config files for MAD). Here you store your *config.ini*.
5. Create a directory *MAD-docker/rocketdb*. (here we store config files for MariaDB). Here you store your *my.cnf*.

Your directory should now look like this:

```
MAD-docker/
  docker-compose.yml
  mad/
  rocketdb/
    my.cnf
  configs/
    config.ini
```

Writing the MariaDB config file

Fill rocketdb/my.cnf file with the following content.

```
[mysqld]
innodb_buffer_pool_size=1G
```

You should align this setting with you available memory. It should probably not exceed 50% of your available memory.

Decrease VM swappiness

```
sysctl -w vm.swappiness=1
```

For further details have a look at https://mariadb.com/kb/en/configuring-swappiness/

Writing the docker-compose file

We use docker-compose to deploy and manage our services.

Fill docker-compose.yml with the following content. Below we explain the details of every service.

```
version: '2.4'
services:
  mad:
    container_name: pokemon_mad
    image: ghcr.io/map-a-droid/mad:async
    restart: always
    volumes:
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
      - ./mad/configs/config.ini:/usr/src/app/configs/config.ini
      - ./volumes/mad/personal_commands:/usr/src/app/personal_commands
      - ./volumes/mad/files:/usr/src/app/files
      - ./volumes/mad/logs:/usr/src/app/logs
      - ./volumes/mad/apks:/usr/src/app/temp/mad_apk
      - ./volumes/mad/plugins:/usr/src/app/plugins
    depends_on:
      - rocketdb
    ports:
      - "8080:8080"
      - "8000:8000"
      - "5000:5000"

  rocketdb:
    container_name: pokemon_rocketdb
    image: mariadb:10.4
    restart: always
    command: ['mysqld', '--character-set-server=utf8mb4', '--collation-server=utf8mb4_
↪unicode_ci', '--innodb_file_per_table=1', '--event-scheduler=ON', '--sql-mode=NO_
↪ENGINE_SUBSTITUTION']
    environment:
      MYSQL_ROOT_PASSWORD: StrongPassword
      MYSQL_DATABASE: rocketdb
      MYSQL_USER: rocketdb
      MYSQL_PASSWORD: AnotherStrongPassword
      TZ: Europe/Berlin
    volumes:
      - ./volumes/rocketdb:/var/lib/mysql
      - ./rocketdb:/etc/mysql/mariadb.conf.d
    networks:
      - default
```

The docker-compose file defines a set of services.

"mad" service

The "mad" service is a docker-container based on the image ghcr.io/map-a-droid/mad:async , which is automatically built by GitHub whenever a push to the *async* happens, using this Dockerfile.

In the docker image, the whole MAD repository is located in `/usr/src/app`.

Volumes:

- The volumes define what is mounted into the docker-container.
- On one hand we mount the **configuration file (config.ini)**.
- On the other hand we "mount out" the **files/directories produced by MAD**, such as the directory "logs" and also the "files" directory, which contains all position files and stats. As usual, volumes are needed for everything **you do not want to lose** after you take the docker-container down.

Ports:

- The docker-image exposes ports 8080 (RGC), 8000 (Pogodroid) and 5000 (Madmin) by default.
- We publish these ports and map them on ports of our host. So e.g. http://your-domain.com:8080 will point to port 8080 of the container, 8000 to 8000 and 5000 to 5000. In this case in RGC you would put http://your-domain.com:8080 as target, in pogodroid http://your-domain.com:8000 and madmin would be reachable under http://your-domain.com:5000.

"rocketdb" service

The "rocketdb" service is docker-container based on mariadb:10.4. It will start a MariaDB database server and automatically create the defined used `MYSQL_USER` with password `MYSQL_PASSWORD`.

Your job here is to set secure passwords for `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD`.

The database is reachable in the default network as *rocketdb*, so in your config.ini it looks like this:

```
dbip: rocketdb                        # IP adress or hostname of the mysql server
dbusername: rocketdb                   # USERname for database
dbpassword: AnotherStrongPassword     # Password for that username
dbname: rocketdb                       # Name of the database
```

"redis" service

There's an optional way to implement a cache layer between the data from the devices and MAD itself. To add that to your stack, just add the following lines to your compose file:

```
redis:
  container_name: pokemon_cache
  image: redis:latest
```

Make sure to set `cache_host` to `redis` in the MAD config.ini. The port can stay on default.

Database deployment

Let's deploy the database, shall we? Just execute:

```
docker-compose up -d rocketdb
```

This will start the "rocketdb" service, take a look at the logs:

```
docker-compose logs -f rocketdb
```

and verify that the database was initialized without problems.

Installing a webfrontend

Add a webfrontend like RocketMAD or PMSF to your setup by just adding another container to the docker-compose.yml.

Warning: Make sure to adjust the config files just like the MAD config.

RocketMAD

```
rocket-mad:
  container_name: pokemon_rocketmad
  image: ghcr.io/cecpk/rocketmad:master
  restart: always
  volumes:
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
      - ./RocketMAD/config/:/usr/src/app/config/
  depends_on:
      - rocketdb
  networks:
      - default
  ports:
      - "5500:5000"
```

Create a new directory and download the basic config file into it: `mkdir RocketMAD && cd RocketMAD && wget -O config.ini https://raw.githubusercontent.com/cecpk/RocketMAD/master/config/config.ini.example`. This docker-compose file will expose RocketMAD on port `5500`, but the internal routing is still on port `5000`, so don't change that in the config.

PMSF

```
pmsf:
  container_name: pokemon_pmsf
  build:
      context: ./PMSF
  restart: always
  volumes:
      - ./PMSF/access-config.php:/var/www/html/config/access-config.php
      - ./PMSF/config.php:/var/www/html/config/config.php
  depends_on:
      - rocketdb
  networks:
      - default
  ports:
      - "80:80"
```

Download the three required files from the PMSF repository:

```
mkdir PMSF && \
cd PMSF && \
wget https://raw.githubusercontent.com/pmsf/PMSF/master/Dockerfile && \
wget -O config.php https://raw.githubusercontent.com/pmsf/PMSF/develop/config/example.
↪config.php && \
wget -O access-config.php https://raw.githubusercontent.com/pmsf/PMSF/develop/config/
↪example.access-config.php
```

PMSF will run on port `80`. Consider using some sort of reverse proxy!

Re-build the container for updating PMSF: `docker-compose build pmsf`.

Note: For more informations and a best practice example, check out the docker-compose used here

Using Traefik 2 as router

If you use Docker, we recommend to use Traefik 2 as router. It is easy to configure, easy to use and it handles alot of things for you, like SSL certificates, service discovery, load balancing. We will not explain, how you deploy a Traefik on your server, but we give you a production ready example for your docker-compose.yml, In this example, we assume:

- your Traefik is connected to a docker-network *proxy*,
- your domain is *example.com* and
- you use a config similar to this:

```
api:
  dashboard: true

providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
    exposedByDefault: false
    network: proxy
```

(continues on next page)

(continued from previous page)

```
entryPoints:
  web:
    address: :80
    http:
      redirections:
        entryPoint:
          to: websecure
          scheme: https

  websecure:
    address: :443
    http:
      tls:
        certResolver: letsEncResolver


certificatesResolvers:
  letsEncResolver:
    acme:
      email: bree@example.com
      storage: acme.json
      httpChallenge:
        entryPoint: web
```

We define the labels as follows:

```
version: '2.4'
services:
  mad:
    container_name: pokemon_mad
    image: ghcr.io/map-a-droid/mad:async
    init: true
    restart: always
    volumes:
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
      - ./mad/configs/config.ini:/usr/src/app/configs/config.ini
      - ./volumes/mad/files:/usr/src/app/files
      - ./volumes/mad/logs:/usr/src/app/logs
    depends_on:
      - rocketdb
    networks:
      - default
      - proxy
    labels:
      - "traefik.enable=true"
      - "traefik.http.middlewares.redirect-to-https.redirectscheme.scheme=https"
      - "traefik.http.routers.madmin.rule=Host(`madmin.example.com`)"
      - "traefik.http.routers.madmin.service=madmin"
      - "traefik.http.services.madmin.loadbalancer.server.port=5000"
```

(continues on next page)

(continued from previous page)

```
      - "traefik.http.routers.pogodroid.rule=Host(`pogodroid.example.com`)"
      - "traefik.http.routers.pogodroid.service=pogodroid"
      - "traefik.http.services.pogodroid.loadbalancer.server.port=8000"
      - "traefik.http.routers.rgc.rule=Host(`rgc.example.com`)"
      - "traefik.http.routers.rgc.service=rgc"
      - "traefik.http.services.rgc.loadbalancer.server.port=8080"

  rocketdb:
    container_name: pokemon_rocketdb
    image: mariadb:10.3
    restart: always
    command: ['mysqld', '--character-set-server=utf8mb4', '--collation-server=utf8mb4_
↪unicode_ci', '--innodb_file_per_table=1', '--event-scheduler=ON', '--sql-mode=NO_
↪ENGINE_SUBSTITUTION']
    environment:
      MYSQL_ROOT_PASSWORD: StrongPassword
      MYSQL_DATABASE: rocketdb
      MYSQL_USER: rocketdb
      MYSQL_PASSWORD: AnotherStrongPassword
      TZ: Europe/Berlin
    volumes:
      - ./volumes/rocketdb:/var/lib/mysql
    networks:
      - default

networks:
  proxy:
    external: true
```

Using these labels, traefik now will:

- route *https://madmin.example.com* to port 5000 (MADmin Flask app).
- route *https://pogodroid.example.com* to port 8000 (Pogodroid listener).
- route *https://rgc.example.com* to port 8080 (RGC listener).

Deploy MAD

To deploy MAD you just execute

```
docker-compose up -d mad
```

Look at the logs with:

```
docker-compose logs -f mad
```

Go to *http://your-domain.com:5000* and check if the MADmin is running.

Useful commands

Some useful commands to maintain MAD + DB

Dump DB:

```
docker-compose exec -T rocketdb /usr/bin/mysqldump -uroot -pStrongPassword rocketdb  >
↪$(date +"%Y-%m-%d")_rocketmap_backup.sql
```

Restore DB:

```
cat <backup>.sql | docker-compose exec -T rocketdb /usr/bin/mysql -uroot -
↪pStrongPassword rocketdb
```

MySQL CLI:

```
docker-compose exec rocketdb /usr/bin/mysql -uroot -pStrongPassword rocketdb
```

Restarting every container:

```
docker-compose down && docker-compose up -d && docker-compose logs -f
```

That will first stop every running container and then start the whole stack again in detached mode (`-d`). The last command is showing the logs.

Further useful Docker tools:

- **Router:** Traefik is recommended, which is really easy to use and also runs as Docker container. To secure the docker-socket (which traefik has access to) we recommend the docker-socket-proxy by Tecnativa.
- **Automatic updates:** Watchtower is a useful tool which will update your docker-services once there are newer images available

Further steps

Review and implement anything related to the security section

2.1.3 Security

If you are securing your MAD setup with SSL proxies, you can change the IPs from the default listeners (MADmin, RGC and PogoDroid) to localhost. MAD opens on `0.0.0.0` by default which means every network interface. But since we are using a webserver proxy, those ports don't need to be exposed on a different interface than localhost (if nginx is running locally):

```
ws_ip: localhost
mitmreceiver_ip: localhost
madmin_ip: localhost
```

The current proxies have configuration examples:

- *Apache2*
- *NGINX*

Reverse Proxy

If you are securing your MAD setup with SSL proxies, you can change the IPs from the default listeners (MADmin, RGC and PogoDroid) to localhost. MAD opens on `0.0.0.0` by default which means every network interface. But since we are using a webserver proxy, those ports don't need to be exposed on a different interface than localhost (if the proxy is running locally):

```
ws_ip: localhost
mitmreceiver_ip: localhost
madmin_ip: localhost
```

The current proxies have configuration examples:

Apache2

For our examples we will use the following:

- MAD runs on localhost
- `madmin_port` is port 5000
- `ws_port` is 8080
- `mitmreceiver_port` is 8000
- We wish to access MADmin at `madmin.example.com`
- We wish to proxy the RGC traffic to `rgc.example.com`
- We wish to proxy the PogoDroid traffic to `pd.example.com`
- The FQDN (Domain) we are using is `example.com`
- SSL Certificate is located at `/etc/letsencrypt/live/example.com/cert.pem`
- SSL Certificate Key is located at `/etc/letsencrypt/live/example.com/privkey.pem`

Make sure that the module `proxy` and `rewrite` is installed and enabled (`a2enmod proxy proxy_http`).

> Keep in mind to configure the DNS settings correctly to make the three subdomains work.

MADmin

MADmin URL: `https://madmin.example.com`

```
<VirtualHost *:443>

    ProxyPreserveHost On
    ProxyRequests Off

    ServerName madmin.example.com
    ProxyPass / http://localhost:5000/
    ProxyPassReverse / http://localhost:5000/

    SSLEngine on
    SSLCertificateKeyFile /etc/letsencrypt/live/example.com/privkey.pem
    SSLCertificateFile /etc/letsencrypt/live/example.com/fullchain.pem
```

(continues on next page)

(continued from previous page)

```
    ErrorLog ${APACHE_LOG_DIR}/madmin_error.log
    CustomLog ${APACHE_LOG_DIR}/madmin_access.log combined
</VirtualHost>
```

RGC

Please install the websocket apache module: `a2enmod proxy_wstunnel`

RGC URL: `wss://rgc.example.com`

```
<VirtualHost *:443>
    ServerName rgc.example.com

    ProxyPass / ws://127.0.0.1:8080/
    ProxyPassReverse / ws://127.0.0.1:8080/

    SSLEngine on
    SSLCertificateKeyFile /etc/letsencrypt/live/example.com/privkey.pem
    SSLCertificateFile /etc/letsencrypt/live/example.com/fullchain.pem

    ErrorLog ${APACHE_LOG_DIR}/rgc_error.log
    CustomLog ${APACHE_LOG_DIR}/rgc_access.log combined
</VirtualHost>
```

PogoDroid

PogoDroid URL: `https://pd.example.com`

```
<VirtualHost *:443>
    ServerName pd.example.com

    ProxyPass / http://127.0.0.1:8000/
    ProxyPassReverse / http://127.0.0.1:8000/

    SSLEngine on
    SSLCertificateKeyFile /etc/letsencrypt/live/example.com/privkey.pem
    SSLCertificateFile /etc/letsencrypt/live/example.com/fullchain.pem

    ErrorLog ${APACHE_LOG_DIR}/pd_error.log
    CustomLog ${APACHE_LOG_DIR}/pd_access.log combined
</VirtualHost>
```

NGINX

For our examples we will use the following:

- MAD runs on localhost
- `madmin_port` is port 5000
- `ws_port` is 8080
- `mitmreceiver_port` is 8000
- We wish to access MADmin at `example.com/madmin`
- We wish to proxy the RGC traffic to `example.com/rgc`
- We wish to proxy the PogoDroid traffic to `example.com/pd`
- The FQDN (Domain) we are using is `example.com`
- SSL Certificate is located at `/etc/letsencrypt/live/example.com/cert.pem`
- SSL Certificate Key is located at `/etc/letsencrypt/live/example.com/privkey.pem`

SSL

Every proxy endpoint will be encrypted with SSL, make sure to adjust the path:

```
ssl_certificate /etc/letsencrypt/live/example.com/cert.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
```

MADmin

The reverse proxy relies on the header, `X-Script-Name`, to inform MADmin on how to construct the URIs.

MADmin URL: `https://example.com/madmin`

```
location ~ /madmin(.*)$ {
    proxy_set_header X-Real-IP  $remote_addr;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header X-Forwarded-Proto https;
    proxy_set_header X-Script-Name /madmin;
    proxy_set_header Host $host;
    proxy_pass http://localhost:5000$1$is_args$args;
    client_max_body_size 200M;
}
```

RGC

RGC URL: `wss://example.com/rgc` (note the extra S in the protocol).

```
location /rgc {
  proxy_pass http://localhost:8080;
  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header Host $host;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

  # WebSocket support
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection "upgrade";
}
```

PogoDroid

PogoDroid URL: `https://example.com/pd` (note the extra S in the protocol).

```
location /pd {
  proxy_pass http://localhost:8000/;
  proxy_set_header X-Real-IP  $remote_addr;
  proxy_set_header X-Forwarded-For $remote_addr;
  proxy_set_header Host $host;
  proxy_set_header X-Forwarded-Proto $scheme;
}
```

For general security refer to *General Security Advice*.

General Security Advice

Here is some security advice that is not only related to MAD but to servers and software hosting in general.

- Don't run MAD inside a webhosted directory like `/var/www/html`.
- MAD does not need root privileges to run. Start it as a normal user. The only programs that need root are your webserver and your database.
- Don't use the same or similar passwords. A password manager can be useful for that.
- Use SSL whenever it's possible. Why? Read this.

Firewall

It's always a good idea to open as few ports as possible. In MADs case that's only 22 for SSH (even that is not 100% necessary in some cases), 80 and 443 for a Webserver if you are proxying everything. Read more about `iptables` here.

SSH Authentication

Follow this guide and install fail2ban.

To get a list of available arguments run

```
start.py --help
```

2.2 Device Setup

2.2.1 Android TV (ATV)

Map-A-Droid currently has support for both 32-bit and 64-bit architecture. Niantic has previously indicated 32-bit architecture support will be dropped in future releases. Be aware of the ATV's architecture when purchasing.

ATV's can have custom firmware(MADRom) flashed that will enable quick deployment of fully configured devices. All current Official MADRom releases are located here.

Both 64-bit and 32-bit architecture ATV's follow the same basic steps for MAD configuration.

- Download the correct MADRom for the device model. Refer to Download ROM for more information. * 32-bit MADRoms are generic while 64-bit MADRoms are device-specific
- Flash the device with the appropriate flashing method. Refer to Flashing instructions for more information.
- Configure the device. Refer to the appropriate architecture section below for specific configuration instructions.

32-bit

The official 32-bit MADRom supports either manual or limited interation configurations. Initial MADRom configuration will install Magisk and smali location patching. Refer to ATV Installation for more detailed information.

64-bit

The official 64-bit MADRom supports either manual or automatic configuration. Initial instillation will install Magisk on first boot. Flashing smali mock location module is not required when using an official MADRom.

The recommended installation configuration path is using the *Auto-Configuration wizard*.

Additional Info

If you're looking for ways to modify your ATV and enhance your setup (3D printed cases, alternate power supplies etc), please see PimpMyAtv for more information.

2.2.2 Android Device

You need a device that is rooted with Magisk, Smali patched (optional, but recommended), PogoDroid and Remote GPS Controller (RGC) installed.

1. Install Team Win Recovery Project (TWRP)
2. Install Magisk
3. Install the game
4. Install and configure RemoteGPSController and PogoDroid

Get started

For a list of confirmed devices that working with MAD, check out this list.

Your device needs to have an unlocked bootloader and must be able to support applications running as root. It is recommended that you install Lineage OS. They have good instructions for each device. Make sure to install the Play Store as described in the Lineage instructions.

If you feel confident to do so you can use a different OS, however, Lineage is recommended.

Always check the exact version of the device, some versions do not work with some ROMs. For example, Samsung has numerous versions numbers of the S5. Some are able to install Lineage and others are not. Lineage lists the device version numbers that it is compatible with.

Once you have Lineage installed and the Google Apps with the Play Store, move on to root your device with Magisk.

Note: Your Android ROM needs to pass Android's SafetyNet check. Make sure to test it via this app before you continue. Usually official LineageOS ROMs do pass the check.

Magisk

Warning: There is currently a bug in the very latest Magisk version that prevents our apps from clicking on the screen (which is needed for several scanning modes). It may work on your device, you have to try it or use Magisk 19.0 or lower to avoid this!

Warning: The hideoption with Magisk version **20.4** is not activated by default. Please activate that function by yourself in the magisk settings first!

1. Install Magisk to root the device via recovery. Download it here
2. Repackage the MagiskManager App and make sure to delete the folder "/sdcard/MagiskManager" after repackaging. If its not present, you are good to go.

Smali Patcher

The Smali Patcher is a little program able to patch Android's internal system files. This will allow you to use the "Mock Location" feature within Android without any apps knowing about it.

1. Follow the instructions to download and install it from the Smali Patcher forum
2. You can ether install it via TWRP or via the Magisk module option in the Magisk Manager.
3. Reboot to TWRP to delete the Davlik and the normal Cache. (WARNING: The TWRP-screen may potentially trigger seizures for people with photosensitive epilepsy.)

Applications

The game

Install the latest supported game version from the Play Store or download it from apkmirror.com. Make sure to add it in *Magisk Manager → Magisk Hide*!

Remote GPS Controller

Download Remote GPS Controller (RGC) first.

RGC must be converted to a system app after it is installed. There are two ways to do this, Link2sd or Systemizer Magisk Module. Link2sd is a bit easier to use but sometimes fails, the Magsisk Module is more reliable but a little bit more complex to use. Choose option #2 (priv-app) when using the Module.

Install Link2sd to systemize RGC. Once installed go into Link2sd, you may be prompted to give it superuser access, if so do accept this. Scroll down and find RGC. Click on it, go to the menu and convert RGC to a system app. It may ask you to reboot if so do so.

Tip: If Link2sd fails to systemize correctly, try App Systemizer for Magisk instead

---

In Android go to *Settings → Developer Options → Select mock location app* and choose "RemoteGPSController".

Open RGC for the first time. A popup will appear to grant superuser permissions to the app. Approve this. If the popup did not appear, you might've missted it. Go to *Magisk Manager → Superuser → enable the game*.

Now you can configure RGC.

- Socket section
  - **Websocket URI**: ws://ipofyourserver:8080. Default port is 8080, you can change this in MAD's config.ini
  - **Websocket Origin**: pick a short unique name for your device. The name must've been configured in MAD as well
  - **Auth**: optional, configure that via the MADmin Auth settings
- Rooted devices section
  - **Reset GMS data**: Off. Keep it off unless you face any GPS issues like rubberbanding
  - **Override OOM**: On. This will help to keep RGC running
- Location Section
  - **Reset AGPS data continuously**: Off. Turn this on when mock location is not used

  - **Reset AGPS once**: Off. Turn this on when mock location is not used
  - **Use Android Mock Location**: On
- General Section
  - **Start on Boot**: On
  - **Start RGC Delay**: 30. Play around with this setting. It's best practice to start RGC after PogoDroid to ensure that PogoDroid is injected before RGC connects starts and connects to MAD
  - **Start services on app start**: On

PogoDroid

1. Install PogoDroid on your device
2. To login to PogoDroid you need to purchase a license from the MADDev shop and follow the instructions.
3. Once logged into the backend, click "Password management" on the top
4. On the password page it should tell you your maximum allowed device count. Create a new device password and copy that
5. Go back to PogoDroid. Use your email address and that new device password you've just created in the previous step.

Now you can configure PogoDroid.

- External Communication Section
  - **Disable external comm**: Off
  - **Send selected set of serialized data (json)**: On. If your workers get stuck in the ocean even though PogoDroid says it is injected, disable and re-enable this setting
  - **Post Destination**: http://ipofyourserver:8000. Default port is 8000, you can change that in the config.ini)
  - **Post Origin**: This value needs to match the value you entered in RGC
  - **Disable last sent notifications**: Your decision, but some devices pull up the navigation bar while showing the notification which causes issues with questmode
  - **Auth**: optional, configure that via the MADmin Auth settings
- App Section
  - **Repackage**: Repackage Pogodroid to hide itself. Currently broken, dont use it
  - **Export Settings**: Export the Pogodroid settings as a file. Useful to setup other devices with the same settings
  - **Injection Delay**: Play around with that setting
  - **Lower SELinux to permissive**: On. Turn it off when the injection is not successful
  - **Full daemon mode**: On
  - **Start Pogodoid with a delay (seconds)**: Play around with that setting. Best practice is to keep that value lower than the delay from RGC
  - **Enable OOM override**: On
  - **Test feature: Mock location patching**: Off. Try this if you cant smali patch

Final Steps

1. Go into Android Settings, Security, Lock Screen Swipe, change to None. You don't want a lock screen. Locking and unlocking your device should bring you to the desktop
2. Go into Android Settings, Developer Options, Stay Awake, make sure this setting is enabled. This will prevent the screen from locking even if pokemon go isn't running
3. If you want to scan quests with that device make sure to hide the navigation bar for PoGo: `adb shell settings put global policy_control immersive.full=com.nianticlabs.pokemongo`
4. Disable vibration for Pokemon GO if you don't want your whole house shaking.

```
adb shell "cmd appops set com.nianticlabs.pokemongo VIBRATE ignore"
```

1. Before we finish, go inside of Magisk and run the Safetynet Check one last time. You need to see all green before proceeding

A device must be rootable for it to be able to perform the mapping functions. The best way to accomplish this is use Android TVs (ATVs) as they have custom ROMs for MAD mapping. It is reccomended you get a device that supports 64bit as Niantic may be deprecating 32bit in the future.

> **Warning:** Android 10 is currently not supported!

For a list of confirmed devices that working with MAD, check out this list. Below are links to the configuration guides:

- *Android TV (ATV)*
- *Android Device*

2.3 MADmin

2.3.1 MAD job processor

MAD has an internal cronjob-like job processing mechanism. With this mechanism, you can run jobs manually or schedule them.

Features:

- Install or update apps on your devices
- Run any command on your devices

Create new job

Create a *personal_commands* folder in MAD's root directory. Save custom jobs as .json file. MAD must be restarted for new jobs to appear.

Example Job to read the game's version:

```
{
  "Readout Pogo Version":
  [
    {
```

(continues on next page)

(continued from previous page)

```
      "TYPE": "jobType.PASSTHROUGH",
      "SYNTAX": "dumpsys package com.nianticlabs.pokemongo | grep versionName",
      "FIELDNAME": "POGO_Version",
      "WAITTIME": 5
    }
  ]
}
```

- `Readout Pogo Version`: Name of job that will appear in MADmin
- `TYPE: jobType.PASSTHROUGH`: Type of job. See *Job types* for a list of supported jobs
- `SYNTAX: dumpsys package com.nianticlabs.pokemongo | grep versionName`: Shell command
- `FIELDNAME: POGO_Version`: Field name for returning Value
- `WAITTIME: 5`: Wait 5 minutes before starting the job

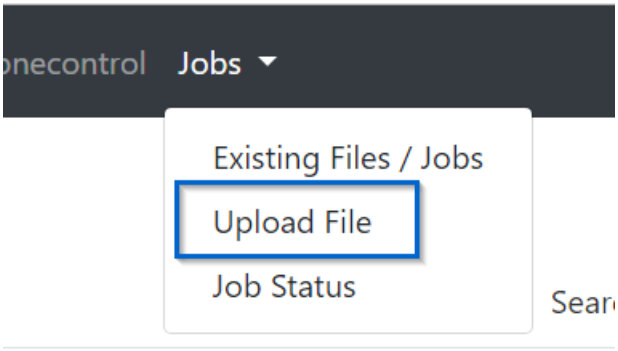
You can chain multiple jobs together. See *Nested jobs*.

APK installation job

MAD can create a job to install an APK on your devices just by uploading the app via MADmin.

Use `upload_path` in config.ini to define the software folder for these APKs (Default: `upload/` in MAD root folder)

MADmin → Jobs → Upload File



Select APK and Upload it.

Job types

MAD supports the following job types:

- `jobType.INSTALLATION`: Install APK on device
- `jobType.REBOOT`: Reboot device
- `jobType.RESTART`: Restart Pogo
- `jobType.STOP`: Stop Pogo
- `jobType.START`: Start Pogo
- `jobType.PASSTHROUGH`: Send command to device

Automatic Jobs

You can configure MAD to launch jobs based on certain timings.

Create file "autocommands.json" in your configured `file_path/` directory. You can set the path via config.ini.

Example content:

```
[
  {
    "redo": true,
    "algotype": "loop",
    "algovalue": 10,
    "startwithinit": true,
    "origins": "tv1",
    "job": "Readout Pogo Version",
    "redoonerror": true
  },
  {
    "redo": true,
    "algotype": "daily",
    "algovalue": "21:00",
    "startwithinit": true,
    "origins": "tv5|tv6|tv7",
    "job": "Readout RGC Version",
    "redoonerror": false
  }
]
```

Description:

- `redo`: `true` will reschedule jobs after finish. `false` will set this jobs to run only once
- `algotype`: `daily` runs this job once a day. `loop` will looü the job every x minutes
- `algovalue`: depends on algotype. :code: *daily* sets time like "21:30" (24h format). `loop` sets loop time in minutes (120 = every 2 hours)
- `startwithinit`: `true` will start the job after MAD start. `false` starts the job according to schedule
- `origins`: Single or list of devices (separated by `|`)
- `job`: Name of the job
- `redoonerror`: Reschedule jobs after getting an error

MADmin API endpoint for jobs

Madmin provides a read-only endpoint via `GET /jobstatus` to read all processed jobs' return value.

Example:

```
{
   "my_device_name": {
      "POGODROID_Version": "[versionName=1.1.3.0]",
      "POGO_Version": "[versionName=0.153.2]",
      "RGC_Version": "[versionName=1.9.3, versionName=1.8.34]"
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Nested jobs

You are able to combine more jobs to one nested or chained job. Nested jobs are processed from top to bottom.

Example:

MAD starts the top most job and will schedule the memory usage readout with a delay of 3 minutes. Eventually, the game will start.

```
{
  "Stop/Start Pogo and readout Memory Usage":
  [
    {
      "TYPE": "jobType.STOP",
      "SYNTAX": "STOP Pogo"
    },
    {
      "TYPE": "jobType.PASSTHROUGH",
      "SYNTAX": "dumpsys meminfo | egrep -w 'Total RAM|Free RAM|Used RAM'",
      "FIELDNAME": "MEMORY_USAGE",
      "WAITTIME": 3
    },
    {
      "TYPE": "jobType.START",
      "SYNTAX": "START Pogo"
    }
  ]
}
```

Hint: If one of the jobs results in an error, following jobs will be canceled.

dumpsys package com.mad.pogodroid | grep versionName PASSTHROUGH success
RETURNING:
[versionName=1.1.3.1]

dumpsys package de.grennith.rgc.remotegpscontroller | grep versionName PASSTHROUGH success
RETURNING:
[versionName=1.9.3, versionName=1.8.38]

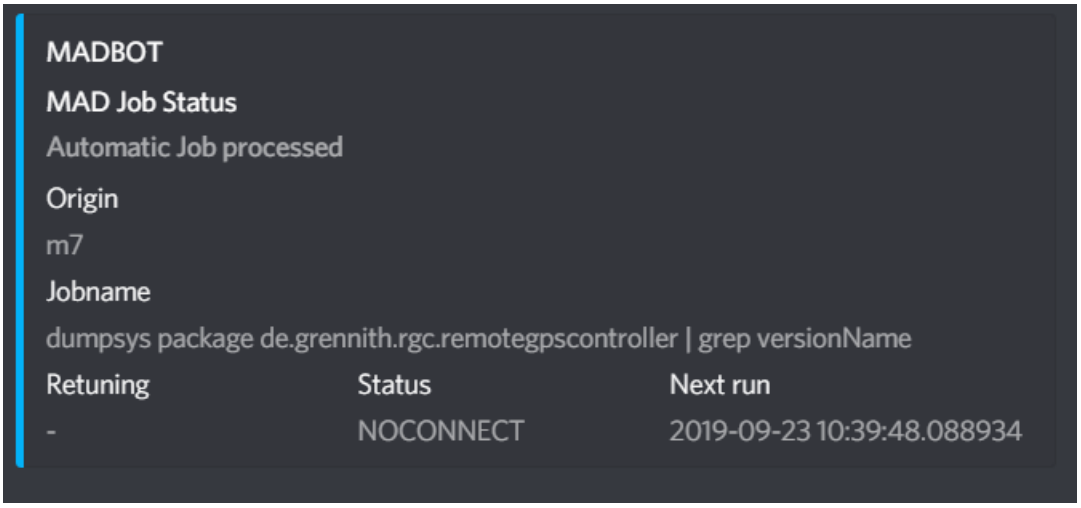
Discord integration

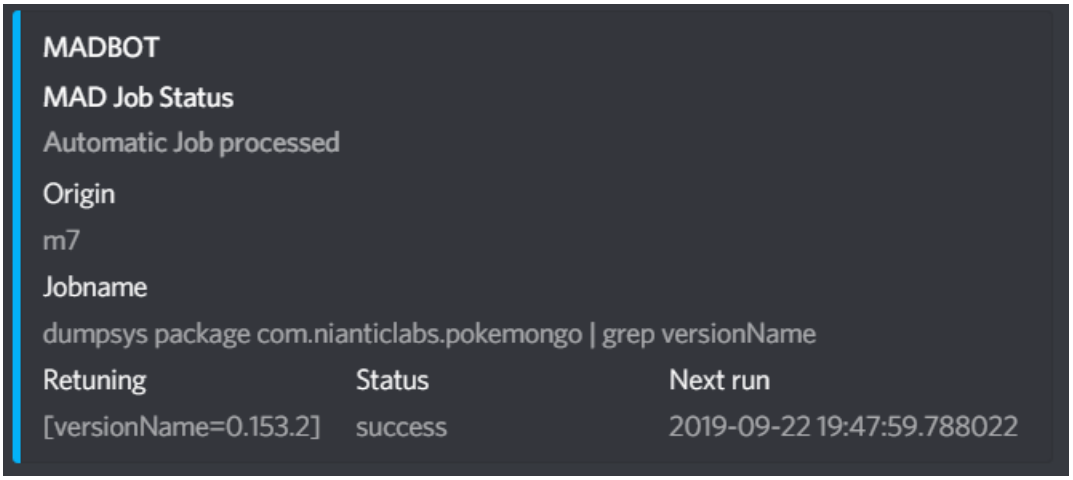
MAD is able to submit a job's state to Discord.

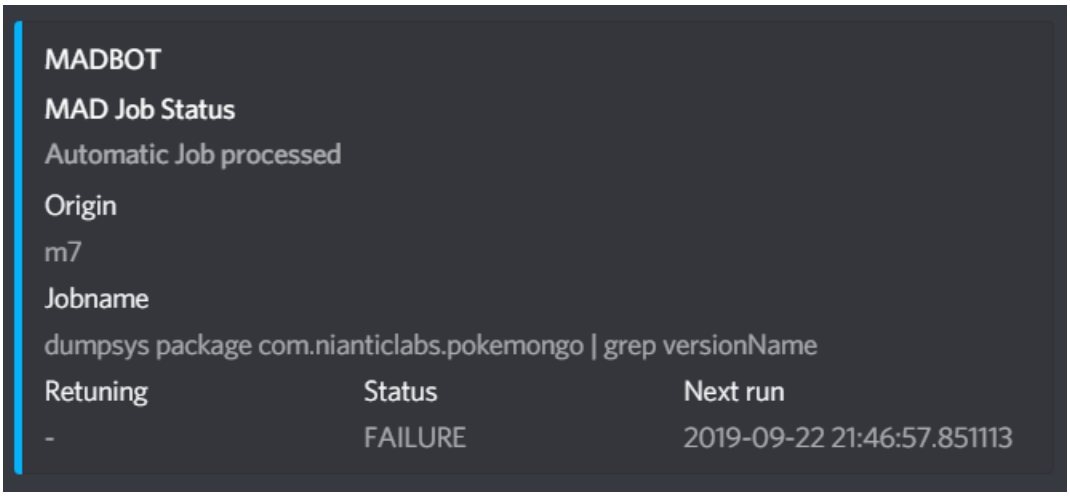
```
job_dt_wh                    # Send job status to discord (Default: False)
job_dt_wh_url:               # Discord Webhook URL for job messages
job_dt_send_type:            # Kind of Job Messages to send - separated by pipe |␣
↪(Default: SUCCESS|FAILURE|NOCONNECT|TERMINATED)
```

- `job_dt_wh`: Enable Discord support
- `job_dt_wh_url`: Discord webhook URL
- `job_dt_send_type`: Define the kind of submission (separated by |) (Default: SUCCESS|FAILURE|NOCONNECT|TERMINATED)

Examples:







2.3.2 Wizard

The wizard performs two major functions, download the latest supported versions of packages (PoGo, RGC, PD) and install the correct support packages through RGC and Smart Jobs.

Package Management

The wizard performs package management by checking the sources for new supported versions and notifying there are updated available via a download icon. The wizard does not run automatically and requires a click on the wizard hat within MADmin -> Setings -> MADmin Packages. The picture below depicts two PokemonGO versions having available updates while rgc and PD are up-to-date. To update the packages you can click on the cloud download button or the wizard hat.

Smart Downloads

The wizard will perform smart-updates for PokemonGO. For the package to update it must be supported by MAD (via addresses.json). If the package is not supported it will not be downloaded. This prevents MAD from having unsupported versions of PokemonGO that cannot be used for with PogoDroid.

Smart Jobs

A smart job executes the same as a normal job but with a few additional steps. It will check the architecture of the device to determine what package needs to be installed on the system. After that it will determine the currently installed version. If the MAD version is thee same or old it will not install the package. If MAD contains a newer package it will install the current package on the device

Manual Uploads

The wizard allows manual uploads to update packages. Prior to upload it validates the package is correct (it performs this operation by looking at the package name inside the APK) and attempting to validate the package architecture. This prevents any invalid uploads that could cause devices to think they updated properly due to a mis-matched package.

Reload / Refresh Wizard

The reload option allows multi-instance systems to refresh the currently downloaded package. Systems that use a shared database can click this once the package is downloaded on one system. For systems that use the filesystem storage option they must share the same directory for the packages.

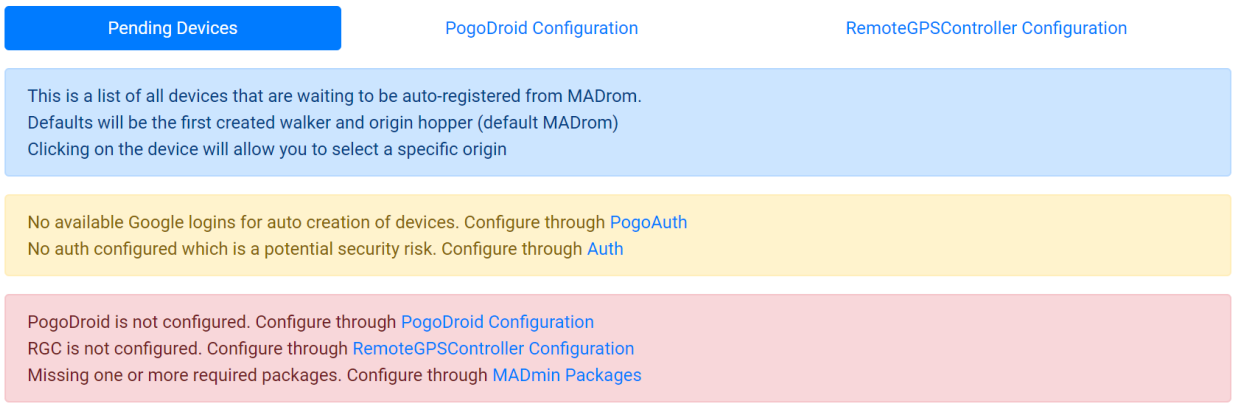
2.3.3 Auto-Configuration

Auto-Configuration supports (mostly) automatic configuration / provisioning of devices after flashing. Currently the only implementation of this is 64bit MAD-ATV ROMs. The supported device list can be found here.

Requirements

There are several requirements that must be met prior to using auto-configuration.

- One or more *walkers* must be configured
- PogoDroid must be configured
- RemoteGPSController must be configured
- Packages must be available in the *Wizard*
- The device must support the MITM endpoints



Package Configuration

PogoDroid and RemoteGPSController are required to be configured prior to the auto-configuration process. This is accomplished by editing the configuration files located in System -> Auto-Config -> PogoDroid Configuration / RemoteGPSController Configuration. The defaults are populated with ATV recommended defaults.

Automated Provisioning

If the requirements have been met the device is ready to be auto-provisioned. If there are any requirements that are missing the session cannot be accepted in MADmin. After the device makes the initial session registration it can be accepted in two different ways.

- Automatic origin assignment
  - Generate an origin via origin_hopper
  - If `autoconfig_no_auth` is not set a google login must be available
- Manual Assignment
  - Pick an origin from all configured devices
  - Logins are not required regardless of the setting `autoconfig_no_auth`

Default Configuration

While each implementation may require a different configuration for intially creating the session, a default configuration file can be automatically generated from MADmin for use with MADROM. This file needs to be placed on the USB stick prior to powering on the ATV for the automatic configuration to occur.

Logging

Messages are logged through the auto-configuration process and viewable in MADmin by click on the history icon. The default behavior of calling the auto configuration endpoints will produce a log. Devices can implement additional logging to give a better indication of the progress of the device. An example log can be found below

Wed Aug 19 2020 06:46:35 GMT-0400	2	Device is attempting to pull a config endpoint, google
Wed Aug 19 2020 06:46:20 GMT-0400	2	Device is checking status of the session
Wed Aug 19 2020 06:45:42 GMT-0400	2	Registration request from 127.0.0.1

2.3.4 Events

cec plz :'D

MADmin is the administration portal for Map'A'Droid and where you will configure the workers. MADmin is broken into several core components which will be described below. This should not be a user-facing page as that should be handled by a frontend. More information on frontends can be found at *MAD Frontends*.

2.3.5 Map

View a map with the most up-to-date information from your system. The map allows you to draw eofences without ever leaving MADmin. To do this click on the polygon icon on the top-right of the map.

2.3.6 Quests

View todays scanned quests with the ability to filter on geofence, stop name, reward, or quest.

2.3.7 Settings

Configure the workers to perform their required tasks. It is broken into several components which can be found *here*

Area

An area defines the scanning type and location that will occur. For more information on scanning types can be found *here*.

Dependencies:

- Geofence
- IV List

Auth

An auth defines credentials used for devices to talk to RGC / MITM. While not required it is highly recommended to use for hardening the system.

Device

A device is how an android device associated with MAD. This is accomplished through the Origin field and states what properties it should inherit when performing its required tasks.

Dependencies:

- PoGo Auth
- Shared Settings
- Walker

Geofence

A geofence contains the constrains in which an area is responsible for mapping. Geofences can overlap without any issues but that does lead to the overlap being scanned twice. When creating a geofence you can specify multiple areas within the fence by using the identifier tag.

A geofence with a single unnamed polygon

```
38.892068305429156,-77.0394802093506
38.89540845718734,-77.03945875167848
38.89544185791166,-77.03368663787843
38.89225201785808,-77.03372955322267
```

A geofence with a single named polygon

```
[the ellipse]
38.892068305429156,-77.0394802093506
38.89540845718734,-77.03945875167848
38.89544185791166,-77.03368663787843
38.89225201785808,-77.03372955322267
```

A geofence with a two named polygons

```
[the ellipse]
38.892068305429156,-77.0394802093506
38.89540845718734,-77.03945875167848
38.89544185791166,-77.03368663787843
38.89225201785808,-77.03372955322267
[whitehouse]
38.898072115622966,-77.0379674434662
38.897036724213834,-77.0379674434662
38.896865549179935,-77.03502774238588
38.89828921190713,-77.03500628471376
```

IV List

An IV list defines the pokemon and order you will encounter mons. The higher on the list the higher priority they have to be encountered. MAD will encounter two per jump which is why setting the priority of the pokemon is important.

Pogo Auth

A PoGo Auth defines a way to login to Pokemon Go. The auth is assigned to the device and used during *Auto-Configuration / Provisioning* of devices.

Shared Setting

A Shared Setting is a additional configuration that can be applied to a one or more devices. These values are overwritten if set on the device configuration page.

Walker

A walker defines a group of areas on when they will run. This allows a device to perform multiple different scanning types. For example, a device can scan quests between 0200 - 0400 then switch to scanning pokemon.

2.4 Updating

2.4.1 MAD

To update an existing MAD install, first stop MAD (this varies depending on how you run MAD), and then update the git repository.

```
git pull
```

Then start MAD again.

2.4.2 Devices

There are a lot of ways to update an APK on an Android device, probably the most easy way is to use the *MADmin Wizard*.

Good alternatives are:

- ADB (`adb install -r app.apk`)
- Simple MADmin job (Jobs –> Create APK install job)
- Shell scripts like update_mad.sh

2.4.3 Docker

When using Docker for hosting MAD you can just pull newer Docker images for every container from the hub:

```
docker-compose pull
```

Make sure to restart the stack afterwards.

2.5 API

A restful API has been created for interacting with MAD. This will standardize how MADmin and users can interact with the data to allow for a more consistent workflow between the two.

2.5.1 Introduction

URI

The API will be available at `/api` (no trailing slash).

Implemented data types

The following data-types have been implemented and can be used for Content-Type or Accept headers

- application/json

Using the API

For all resource roots, GET functionality has been implemented. Please refer to the section Global GET Parameters to view available options. For all sub-components, the following operations have been implemented.

- `DELETE`: Attempt to delete the resource. It will fail if it is a dependency for another object.
- `GET`: retrieve data.
- `PATCH`: update the resource but only modify the fields that have been sent.
  - If you wish to append values to a list, use the header `X-Append: 1`. If this is not present, it will replace the list.
- `POST`: create a new resource. Response content will contain the newly-created object.
- `PUT`: replace the existing object with the new one being sent.

If an error occurs, refer to the returned headers to find the issue.

Global Headers

The following global headers can be used when calling the API

- Content-Type (str): incoming data format. Default is application/json
- Accept (str): outgoing data format. Default is application/json
- X-Beautify (integer): Display the returned data in a human-readable format
- 1: Format the response body.
- All other options: no formatting

Global GET Parameters

The following parameters are available for all root resources

- fetch_all (int): Fetch all data related to the object versus the display field
- 1: Fetch all data
- All other options: only return the uri / display field
- display_field (str): Default sort field. If fetch_all is not 1, this field will be returned as the value for the key pair
- hide_resource (int): Hide any filter information related to the resource and only return the results
- 1: Hide any filtering information
- All other options: return all information

Searching via the API is done through the GET parameters. If no search logic is specified it will default to equals. To perform a search for a field include <field_name>.<operation>=search. The following operations are implemented:

- **eq: The field must be equal to the value. This is the default search if none is specified.**
  To query all devices that use the device pool /api/walker/5 `/api/device?walker.eq=/api/walker/5`

  To query all devices that use the MAC address 06:81:91:10:2d:c8 `/api/device?origin.mac_address.eq=06:81:91:10:2d:c8`
- **like: Performs a similar search.**
  To query all origins that begin with tx9s `/api/device?origin.like=tx9s`

Response Headers

The following headers can be returned for all resources:

- Location (str): Returned during a successful POST operation. States the URI of the newly created resource
- X-Status (str): Human-readable result of the command
- X-Uri (str): Returned during a successful POST operation. States the URI of the newly created resource

Status Codes

- 200: Successfully found and returned the data
- 201: Successfully created a new resource
- 204: The resource has been successfully updated or replaced
- 400: A required header was missing. Refer to the body of the response to determine the issue
- 404: The resource ID was not found
- 412: A dependency failed validation. Refer to the body for a list of failed dependencies (list of objects containing uri / display_name).
- 422: An error occured while creating the resource. Refer to the body for the errors

MAD APKs

MAD APKs allows an administrator to store APKs in the MAD Database to be used for the API or SmartJobs. Available endpoints are available on the MAD APK Page

Origin Hopper

Origin Hopper allows for rapid deployment of new devices into a MAD environment. It will generate an origin and create the device within MAD. This functionality is only available through the PogoDroid / MITM Receiver port.

Area

URI: `/api/area`

The following is unique about areas:

- A valid mode must be specified. If a mode is not specified or is not valid, 412 will be returned
- Dependencies:
- walkerarea

Authentication

URI: `/api/auth`

There is no special handling for authentication

Devices

URI: `/api/device`

There is no special handling for devices

DevicePools

URI: `/api/devicesetting`

The following is unique about device pools:

- Dependencies:
- device

MonLists

URI: `/api/monivlist`

The following is unique about Mon Lists:

- Dependencies:
- areas

Walkers

URI: `/api/walker`

The following is unique about Mon Lists:

- Removing a walker will check and remove any walkerareas assigned to the walker that are no longer in use
- Dependencies:
- device

WalkerArea

URI: `/api/walkerarea`

The following is unique about Mon Lists:

- Dependencies:
- walker

2.5.2 Examples

Getting all auth

Using default sort field

```
curl -H 'X-Beautify: 1' http://localhost:5000/api/auth
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /api/auth HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> X-Beautify: 1
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 458
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Headers: Content-Type,Authorization
< Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
< Server: Werkzeug/0.15.6 Python/3.6.8
< Date: Mon, 14 Oct 2019 18:12:32 GMT
<
{
  "resource": {
      "fields": [
          {
              "name": "username",
              "descr": "Username of device",
              "required": true
          },
          {
              "name": "password",
              "descr": "Password of device",
              "required": true
          }
      ],
      "settings": []
  },
  "results": {
      "/api/auth/1": "cec",
      "/api/auth/0": "grennith"
  }
}
* Closing connection 0
```

Using custom sort field

```
curl -v -H 'X-Beautify: 1' http://localhost:5000/api/auth?display_field=password
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /api/auth?display_field=password HTTP/1.1
> Host: localhost:5000
```

(continues on next page)

(continued from previous page)

```
> User-Agent: curl/7.58.0
> Accept: */*
> X-Beautify: 1
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 459
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Headers: Content-Type,Authorization
< Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
< Server: Werkzeug/0.15.6 Python/3.6.8
< Date: Mon, 14 Oct 2019 18:13:58 GMT
<
{
    "resource": {
        "fields": [
            {
                "name": "username",
                "descr": "Username of device",
                "required": true
            },
            {
                "name": "password",
                "descr": "Password of device",
                "required": true
            }
        ],
        "settings": []
    },
    "results": {
        "/api/auth/0": "12345",
        "/api/auth/1": "test123"
    }
* Closing connection 0
}
```

Pulling all information resource information and hiding the resource fields

```
curl -v -H 'X-Beautify: 1' 'http://localhost:5000/api/walker?hide_resource=1&fetch_all=1'
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /api/walker?hide_resource=1&fetch_all=1 HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> X-Beautify: 1
>
* HTTP 1.0, assume close after body
< HTTP/1.0 200 OK
< Content-Type: application/json
< Content-Length: 691
< Access-Control-Allow-Origin: *
```

(continues on next page)

(continued from previous page)

```
< Access-Control-Allow-Headers: Content-Type,Authorization
< Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
< Server: Werkzeug/0.15.6 Python/3.6.8
< Date: Mon, 14 Oct 2019 18:21:08 GMT
<
{
    "/api/walker/2": {
        "setup": [
            "/api/walkerarea/8",
            "/api/walkerarea/9",
            "/api/walkerarea/10",
            "/api/walkerarea/10"
        ],
        "walkername": "iv_checker"
    },
    "/api/walker/0": {
        "setup": [
            "/api/walkerarea/0",
            "/api/walkerarea/1",
            "/api/walkerarea/2"
        ],
        "walkername": "quest_example"
    },
    "/api/walker/1": {
        "setup": [
            "/api/walkerarea/3",
            "/api/walkerarea/4",
            "/api/walkerarea/5",
            "/api/walkerarea/6",
            "/api/walkerarea/7"
        ],
        "walkername": "raid_mon_example"
    }
}
* Closing connection 0
```

Creating a walker

```
curl -v -H 'X-Beautify: 1' -H 'X-Append: 1' -X POST -H 'Content-Type: application/json' -
↪d '{"setup":["/api/walkerarea/10"], "walkername": "test"}' http://localhost:5000/api/
↪walker
> POST /api/walker HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.65.3
> Accept: */*
> X-Beautify: 1
> X-Append: 1
> Content-Type: application/json
> Content-Length: 54
>
```

(continues on next page)

(continued from previous page)

```
* upload completely sent off: 54 out of 54 bytes
* Mark bundle as not supporting multiuse
* HTTP 1.0, assume close after body
< HTTP/1.0 201 CREATED
< Content-Type: application/json
< Content-Length: 63
< Location: http://localhost:5000/api/walker/6
< X-Uri: /api/walker/6
< X-Status: Successfully created the object
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Headers: Content-Type,Authorization
< Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
< Server: Werkzeug/0.16.0 Python/3.7.5rc1
< Date: Thu, 31 Oct 2019 00:13:51 GMT
<
{
    "walkername": "test",
    "setup": [
        "10"
    ]
}
* Closing connection 0
```

Adding a walkerarea to a walker

```
curl -v -H 'X-Beautify: 1' -H 'X-Append: 1' -X PATCH -H 'Content-Type: application/json'␣
↪-d '{"setup":["/api/walkerarea/10"]}' http://localhost:5000/api/walker/2
* Connected to localhost (127.0.0.1) port 5000 (#0)
> PATCH /api/walker/2 HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.58.0
> Accept: */*
> X-Beautify: 1
> X-Append: 1
> Content-Type: application/json
> Content-Length: 32
>
* upload completely sent off: 32 out of 32 bytes
* HTTP 1.0, assume close after body
< HTTP/1.0 204 NO CONTENT
< Content-Type: application/json
< X-Status: Successfully updated the object
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Headers: Content-Type,Authorization
< Access-Control-Allow-Methods: GET,PUT,POST,DELETE,OPTIONS
< Server: Werkzeug/0.15.6 Python/3.6.8
< Date: Mon, 14 Oct 2019 18:09:01 GMT
<
* Closing connection 0
```

2.5.3 APKs

MAD can store APKs in the filesystem (default) or database and allow an end-user / client to download them when requested. This information is available via MADmin and PogoDroid. Both sections require the following headers to be present:

- Origin
- Authentication (if configured on the MAD server)

The following APK Types are available:

- pogo, com.nianticlabs.pokemongo, 0
- rgc, de.grennith.rgc.remotegpscontroller, 1
- pogodroid, com.mad.pogodroid, 2

The following Architectures are available:

- noarch, 0
- armeabi-v7a, 1
- arm64-v8a, 2

APK Definition

When requesting info for an APK the following data will be returned:

- arch_disp: Friendly-Name for the architecture of the APK
- file_id: File ID database reference
- filename: Filename that will be saved (unless changed). Default format is `<apk_type_friendly_name>_<apk_architecture>_<version>.apk`
- mimetype: MIME-Type of the file (should always be 'application/vnd.android.package-archive')
- size: Size in bytes of the file
- usage_disp: Friendly-Name of the apk type
- version: Version of the APK

Endpoints

GET

All APK Information

Returns all information available for any downloaded APK.

- MADmin: `/api/mad_apk`
- PD: Not Available
- Return Information: {apk_type:{architecture:{APK Definition}}, architecture:{APK Definition}, ..}

APK Type Information

Returns all information available for the given APK. This will include all information for all architectures

- MADmin: `/api/mad_apk/<apk_type>`
- PD: `/mad_apk/<apk_type>`
- Return Information: {architecture:{}}, architecture:{}, ..}

APK / Architecture Information

Returns all information for the given APK / Architecture. If architecture is not specified it will attempt to return the data for noarch.

- MADmin: `/api/mad_apk/<apk_type>/<apk_arch>`
- PD: `mad_apk/<apk_type>/<apk_arch>`
- Return Information: {APK Definition}

APK Download

Download the currently downloaded version stored in the database. If architecture is not specified it will attempt to use noarch.

- MADmin: `/api/mad_apk/<apk_type>/<apk_arch>/download`
- PD: `mad_apk/<apk_type>/<apk_arch>/download`
- Return Information: Downloads the APK

POST

Upload an APK

Upload a new APK into the MAD database. Both fields must be specified

- MADmin: `/api/mad_apk/<apk_type>/<apk_arch>`
- PD: Not Available

Delete

Remove an APK

Delete an APK from the MAD databasee. Both fields must be specified

- MADmin: `/api/mad_apk/<apk_type>/<apk_arch>`
- PD: Not Available

2.5.4 Origin Hopper

The Origin Hopper / Generator was designed to allow for rapid deployment of devices. It utilizes the PogoDroid port as it is already configured for the MAD rom.

Headers

The following headers can be used when making requests to the origin hopper:

- Authorization (required): Authorization configured on the Auths page. If there are no configured auths this is not a required field.
- OriginBase (required): Prefix for the origin. If you wanted OriginDefault<1-n> you would use OriginDefault.
- walker (optional): Walker ID to assign to the device.
- pool (optional): Pool ID to assign to the device.

Endpoint

There is only one endpoint for the origin hopper / generator, `/origin_generator`. Due to this being generator a GET action performed with the correct headers will create the origin. The body will return the new origin to used with MAD. Once created you will see the device in MADmin and can make any required configuration changes. If the device fails to create a 400 or 404 will be returned with the reason.

2.5.5 Resources

The following resources have been implemented to the API:

Area

The following modes are available in the API:

- *idle*
- *iv_mitm*
- *mon_mitm*
- *pokestops*
- *raids_mitm*

idle

iv_mitm

Settings

mon_mitm

Settings

pokestops

Settings

raids_mitm

Settings

Auth

The following definition is available for Authentication resources

Device

The following definition is available for Device resources

Settings

Devicesetting

The following definition is available for Device Pool resources

Settings

Monivlist

The following definition is available for Mon IV Lists resources

Walker

The following definition is available for Authentication resources

Walkerarea

The following definition is available for WalkerArea resources

2.6 Extras

2.6.1 Integrations

Frontends

RocketMAD

A powerful frontend maintained by MAD's core developers and the MAD community. Written in Python.

RocketMAD on Github

PMSF

PMSF or PokeMap Standalone Frontend is a powerful map written in PHP. Right now, there's one fork that is being activately developed by whitewillem and wheel which does support MAD.

PMSF on Github

Webhook tools

MAD is able to send the scanned data via webhook to third party software.

PokeAlarm

PokeAlarm (PA) is the most common tool. It relays almost anything to several types of chats like Telegram or Discord. It also provides a lot of filter possibilities to configure it to anyone's needs.

PokeAlarm on Github

PoracleJS

PoracleJS is based on running a Discord or Telegram bot that will either send events to a group/channel or as a private message to you. So every user can set their own filters. PoracleJS is able to send mon, raids and quests.

PoracleJS on Github

2.6.2 MADevice

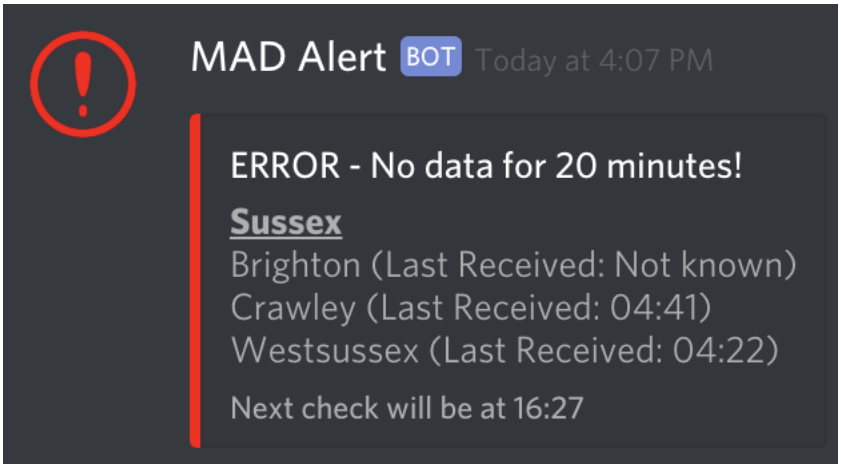
View MADevice on GitHub

What is MADevice

MADevice is a service that will alert you when a device may be having an issue. It can be used if you just want to check the status of numerous devices (phones and Android TVs) across many MAD instances from within Discord without having to load numerous MADmin windows. The following two sections detail each feature available.

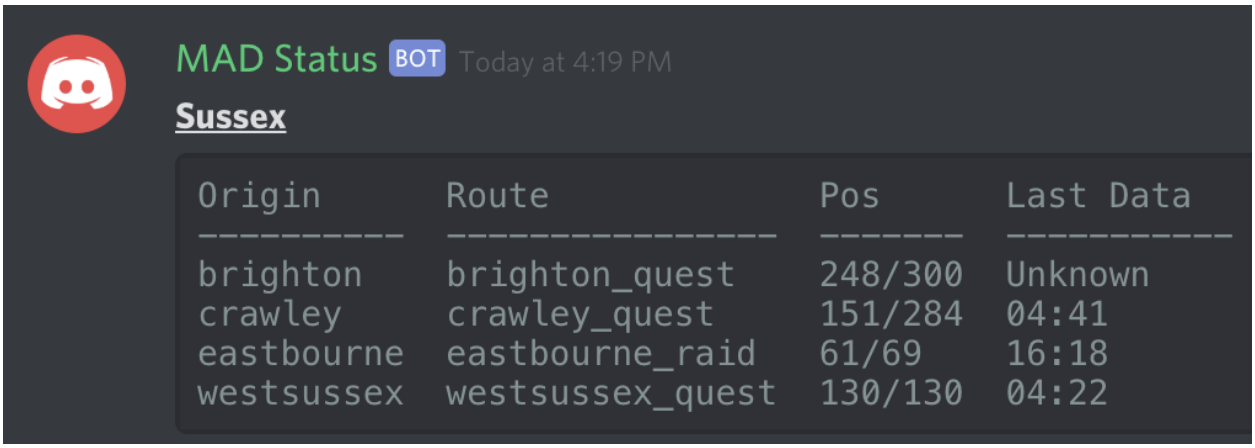
No Data Alert

When running MADevice, it will check the last received time for data from PoGoDroid and then if the time is more than 20 minutes (or the configured duration) in the past, it will post a message to the channel set by webhook in servers.json



On-Demand Status (!status)

If you type `!status` in the channel set by `status_channel_id` in `servers.json`, you get an on-demand update across all servers (set in servers.json) and posted into Discord rather than opening up multiple browsers to see the data.



Installation

For up-to-date installation steps visit MADevice's README

2.6.3 Scripts

Here is a collection of useful scripts found in the `scripts/` directory.

Spawnpoint Importer (import_allspawns.sh)

If you used to scan before and happen to still have your spawnpoints in your Monocle or Rocketmap database then you can use this script to import them to MAD. You must have the trs_spawn table already in your database and you must have filled out the Database portion of the MAD config file!

It's also possible to import spawnpoints from a RDM database.

Intel Importer (intelimport.sh)

If you ran the MITM method for the first time, you will probably notice that all gyms are missing names and pictures. If you want to add this information, you can use this script. First of all, you'll need a CSV Export from the Ingress Intel Map. Install IITC and the `IntelCsvExporterMADedition.js` from the scripts directory. Make sure to scrape all the necessary portals in your area and export the CSV file to your server. The second step is to run the script with the csv file as the first parameter. Example: `./intelimport.sh export.csv`.

Databasesetup (databasesetup.py)

This script will take care of the database schema installation automatically so for example there is no need to install a whole RocketMAD frontend to just use the schema. Make sure to fill in the database credentials in the MAD `config.ini`, create an empty database and grant all permissions to the database user before running this script.

Migrate to RocketMAD (migrate_to_rocketmap.sh)

This script will migrate your data from an RDM or Monocle database to a RocketMAD database. Before you run this you should run RocketMAD or `databasesetup.py` and let it configure its database. After it has built its empty database you can run this script. If you were using Monocle with MAD spawnpoints do not change, so I dump that table from your monocle db and import it to your rocketmap db for you. If you have old spawnpoint info from before MAD then you want to use import_allspawns.sh as well. This script does not import things like controlling team/mons, or ex status, because MAD will fill this in after 1 scan.

If you were already scanning in MAD using your Monocle database, be sure to remove version.json so MAD will update your new rocketmap schema.

Update Stops and Gyms (update_stopsgyms.sh)

This script will find and delete gyms that have changed to pokestops and pokestops that have changed to gyms.

2.7 FAQ

2.7.1 General

How many Mon/Quests/Raids can I scan with one device?

Nobody can tell you that. That depends on your area e.g the amount of spawnpoints/stops/gyms in that area and the distance between them. Start with a small area, increase it over time and see how much is possible.

Rule of thumb: while moving, the game requests data every 10 seconds. So it takes *at least* 10 seconds per location to scan for data. An area with 150 coordinates takes roughly 25 minutes to scan for single device.

Is it planned to see gym defenders + trainers in gyms?

No. Definitely not. To avoid drama *and* privacy concerns the main MAD developers decided leave out this kind of information from gym data.

2.7.2 MAD

All my stop and gym names are "unknown"

PogoDroid can't fetch this kind of information automatically in init mode. The game simply doesn't submit this information without clicking stops or gyms. You can either run quest mode to get the stop names and / or use the *intelimport.sh* script to import those names and pictures from Ingress.

How can I check if MAD receives data?

If you see a green SUCCESS line with "Processing GMO" in it, then leave it alone - it's working!

Can I use multiple area fences in one area?

Yes. Just add the second area to the same geofence like this:

```
[area1]
51.123, 9.234
51.124, 9.235
[area2]
12.345, 1.234
54.321, 2.345
```

How can I regenerate a route?

To renerate a route just hit the blue button in the area section of MADmin.

How can I remove Eventspawnpoints?

Nia sometimes activates more spawnpoints for Community Day or other events. You shoud delete them after the event has ended so MAD will not consider them in the routecalc or PrioQ.

```
DELETE FROM trs_spawn WHERE first_detection like '2019-01-01%';
```

Fill in a Date which is shortly after the event started but keep the %.

If you want, you can backup those spawns and just temporary insert them for the events. (Google "mysqldump" if you want to know how)

How can I rescan quests?

Delete the quests from the *trs_quest* table and restart MAD.

```
TRUNCATE TABLE trs_quest;
```

How can I resend webhooks?

Start MAD with `--webhook_start_time` and the epoch start timestamp. You may want to add `--webhook_max_payload_size` as well to not overload your webhook receiver.

This example will send every webhook since midnight in requests with 20 objects in it, again.

```
python3 start.py --webhook_start_time $(date -d "today 00:00" '+%s') --webhook_max_
↪payload_size 20
```

2.7.3 Game

My character is stuck in the ocean and it's not moving

If you don't have a red GPS error at the top it means that RGC is working but didn't get any commands from the MAD server.

- Check if your phone registered to the MAD server. The log line should look like this: *[INFO] Client ORIGIN-NAME registering*
- Check if that phone has something to do according to your MADmin settings. Have a look at your MAD logs for that.

PoGo (sometimes) says that my phone has an unsupported OS

Sometimes it may just be a hickup, try a reboot

- Check SafetyNet status via this app - your phone has to pass this check.
- Go into MagiskManager and repackage it if you have not done so already: MagiskManager > Setting > Repackage MagiskManager
- Add PoGo to Magisk Hide: MagiskManager > Magisk Hide > Check PoGo

I can see the red error (70) sometimes

That's nothing to worry about. It's the way Pogodroid can scan IV.

How do Spawnpoints work?

Mon are always spawning on the same spot. Those spots are called spawnpoints and each of them have a unique timer when they are active or not. If they are active, a mon is present for either 30 or 60 minutes. They act the same for every hour, so all thats important is the minute and second when the spawnpoint becomes active. That information can only be gathered in the last 90 seconds of an active spawn. If MAD does not have that information yet, it'll default to 3 Minutes. You can find more informations about that by clicking on a spawnpoint on the MADmin map.

Quest mode doesn't click anything on the screen

- Check if you are using a correct Magisk version. 19.1, 19.2 and sometimes 19.3 blocking RGC to click on the screen. 19.0 will work just fine.
- Check if you have a navigation bar on your screen. If yes: disable it with this adb command: `adb shell settings put global policy_control immersive.full=com.nianticlabs.pokemongo`. It will then be hidden in the game.

Should I be worried about the popups on the phone?

Popups from PoGo like "You are moving too fast" and "Dont drink and drive" doesn't matter except for quest scanning. But MAD will handle them.

RGC or Pogodroid are crashing randomly

Disable battery optimisations in Android and enable the OOM override option in Pogodroid/RGC

My workers aren't following my priority queue / route on the map

In fact, they do. The RouteManager removes and entry from the prioq and assigns it to a worker. While the prio route has already been updated on the map, the worker position is only set when the worker arrives at its destination. Depending on your mode and your settings it can take several seconds before the worker marker arrives at the location where once has been a prioq coordinate.

sql_mode error, MySQL strict mode, mysql mode.

For MAD to function properly you will need to adjust your MySQL/MariaDB server `sql_mode`. There are few modes that breaks MAD and you will be asked to to disable those, however for maximum comfort and to avoid problems in future updates we suggest disabling everything, not only those reported.

Set your `sql_mode` to `NO_ENGINE_SUBSTITUTION` or even to empty.

This tutorial will cover Ubuntu/Debian way with some steps to reproduce. Make sure to run those commands as **root** (or with sudo).

Step 1

Find your MySQL/MariaDB main config and check what directory should we use. Run those commands and see if they report anything.

```
grep includedir /etc/my.cnf
grep includedir /etc/mysql/my.cnf
```

If you got `No such file or directory` two times it's time to consult your distro/system manual where is yours MySQL/MariaDB config file. Expected output it something like that (**do not use those dirs from example, use your own!**):

```
$ grep includedir /etc/mysql/my.cnf
!includedir /etc/mysql/conf.d/
!includedir /etc/mysql/mariadb.conf.d/
```

or

```
# grep includedir /etc/mysql/my.cnf
!includedir /etc/mysql/conf.d/
```

The part after `!includedir` is the interesting part - it's directory where we will create our custom settings file. It will vary from distro/version - so always check it. If you have more than one result (like first example) select one directory - for now I will use `/etc/mysql/conf.d`.

Step 2

Make sure that this directory exists.

```
mkdir /etc/mysql/conf.d
```

If you got `mkdir: cannot create directory '/etc/mysql/conf.d': File exists` then nothing to worry about - directory already there, go to step 3. If you got `Permission denied` then make sure to run this command as **root** or with **sudo**.

Step 3

Create new file *MAD.cnf* in that directory

```
nano /etc/mysql/conf.d/MAD.cnf
```

Step 4

Copy-paste (right mouse click in PuTTy) below content into that file and save it (`CTRL-o`, `enter`, `CTRL-x`)

```
[mysqld]
sql_mode="NO_ENGINE_SUBSTITUTION"
```

If it complains about `Permission denied` then go back step 3 and make sure you run is as **root** or with **sudo**.

Step 5

Restart MySQL/MariaDB to apply new settings. Here are few commands - one should work. Work from top - if you see that MySQL/MariaDB server was restarted there is no need to issue rest of commands - just covering more ground. Run as **root** or with **sudo**.

```
service mariadb restart
service mysql restart
service mysqld restart
/etc/init.d/mysql restart
```

What's the difference between these scanning modes?

MITM is short for "Man In The Middle". PogoDroid will inject into the running game process to read the data which is received from the game server.

mon_mitm

`mon_mitm` will scan for mon within a 70 meter radius. By default, no mon gets encountered and checked for IV, unless you define a list (`mon_ids_iv`) of IDs that should be encountered. The order of the IDs is the priority of them. So, for example, put Snorlax before Pidgey to make sure PogoDroid will scan Snorlax first). PogoDroid has a built-in limit of 2 encounter per location.

iv_mitm

This mode is relying on already scanned and active mon in your DB (via `mon_mitm` for example). It will jump directly to them to do an IV check. `iv_mitm` will build up a "first in first out" queue.

pokestops

You can use this mode for two things. Quest scanning or leveling.

Quest scanning will walk on a pre-calculated route to every stop and spin it. When the area is set to *coords* in the walker, MAD will check every other stop in the area (even those who are not on the route). Those stops will be processed after the first round. This process will repeat itself three times. MAD is able to determinate the exact mon encounter and item type when picking up the quest.

The level option is basically the quest mode but without constantly clearing out the quests in the queststack and MAD will check if that stop is unique for the worker. In case it's a stop that has been visited in the past, it will be skipped.

raids_mitm

This mode is used to scan every gym and raid in a 490 meter radius. No interaction with any ingame objects is needed. MAD will only scan the gym color, the current gym defender, free slots and the raid or egg if present. (More about that in the FAQ).

idle

The phone will stop the game and do nothing.

2.8 Glossary

ADB: Android Debug Bridge is a command-line tool for communicating with Android devices.

APK: Android application package is the app file format used on Android devices.

ATV: Android TV. See MAD-device-list for a list of suggested devices.

Magisk: Magisk is a suite of open source tools for customizing Android and is required for running PoGo on a rooted Android device.

MAD: Map'A'Droid - this software! Sometimes referred to as Map-A-Droid; usually where ' can't be used (URLs and such).

MADevice: 3rd party tool, MADevice is a service that will alert you when a device (phone or ATV) may be having an issue.

MADmin: Map'A'Droid admin software.

MITM: Man In The Middle.

PD: PogoDroid.

PMSF: PokeMap Standalone Frontend is a powerful map written in PHP.

RGC: Remote GPS Controller.

RocketMAD: RocketMAD is a powerful frontend maintained by MAD's core developers and the MAD community.

scrcpy: scrcpy provides display and control of Android devices connected on USB (or over TCP/IP). Like a VNC viewer for android devices.

TWRP: Team Win Recovery Project is an open source recovery image for Android devices.

2.9 ASYN Migration Quick FAQ

2.9.1 First steps

Backup

If you are updating from `legacy/master` to `async` please create a backup of your current working MAD setup. Backup of database, backup of files, backup of nginx/apache2 settings - etc. Have a backup, backup is important, full backup - the more the better. Backup, yes. Backup.

async/architecture changes

If you running more than **15 (your mileage may vary) devices** you probably need to run some extra MITM Receivers. If you are not near this number you can skip reading this point here - as you will just use single `start.py` and everything will be easier. MAD is now split into 4 parts:

- `start_core.py` this is main part of MAD that handles websockets/login flow/routes/MADmin etc.
- `start_mitmreceiver.py` this is part of MAD that process data received from PD and puts that in MySQL/database. This is the heaviest (CPU usage) part of MAD as it runs 24/7 and do a lot.
- `start_mitmapper.py` this is helper part
- `start_statshandler.py` this is just stats part

There is also:

- `start.py` that starts everything above in one single process, but that also means it's limited to 1 vCPU/core so if you throw too many devices at that it will struggle to keep up.

There will be some examples how to run *Multiply MITM RECEIVERS*.

config.ini changes

There are some outdated settings like `weather`, `madmin_login`, `madmin_password` that need to be deleted. There is new setting called `apk_storage_interface` that need to be set to `db` or `fs` in `config.ini`. You can read the description in `config.ini.example`. Also please set `mitmmapper_type: redis`. Remember that commented out lines (# first in that line) are commented out and it will not use those.

Software needed before

- **at least** `python3.9` - most modern systems have that in repository (*apt install python3.9* for example) or for older one you can most likely install it from external repositories like deadsnakes. Before going to install anything check what is your current python version installed - it could be already there - try running `python3.9 --version` and `python3 --version`. Remember to install `python3[.9]-dev python3[.9]-venv python3[.9]-opencv` packages.
- redis-server installed (`apt install redis`) - it's required now, this does not any configuration by default, just install.
- *venv, python-venv, virtualenv* - use venv, not local packages, not global packages, use venv, really, it makes life easier for everyone. TODO_LINK_HERE_TO_MANUAL_INSTALL_MAD_DOCS_VENV
- `tesseract`, database `MariaDB/MySQL` is still there and if you are just updating you should have all of this.
- list of packages needed for Debian 11 clean install - do NOT blindly-copy paste this - this is just for reference `python3-dev python3-venv mariadb-server redis mariadb-client build-essential git default-libmysqlclient-dev python3-opencv tesseract-ocr pkg-config`

2.9.2 Migration

Migration

There are some (a lot!) of changes between versions and migrations are sometimes funky - please take a look at *Common problems* if you hit any error/problem. All the ports and endpoints (ws:// http://) stays the same.

- Step 1: Create Backup. Backup - first point in this Quick FAQ.
- Step 2: There are breaking *PogoAuth changes* so before running migration make sure there is only one (currently logged in on device) account mapped to single device in MADmin **Pogo Auth** section.
- Step 3: Stop current legacy/master MAD.
- Step 4: You don't want to waste 30 minutes on MAD/MySQL changing the `pokemon` and `pokemon_display` tables because you have 17851758 / 6 months of mons history there! Clear those up: `TRUNCATE` or `DELETE FROM` if you don't do it already automagically.
- Step 5: Remove `update_log.json` file from MAD master/legacy main directory.
- Step 6: Run `git checkout async` from MAD master/legacy main directory.
- Step 7: Run `git status` from MAD master/legacy main directory, it should show:

```
On branch async
Your branch is up to date with 'origin/async'.
```

- Step 8: Adjust config.ini (*config.ini changes*)
- Step 9: Install new requirmements in python3.9 (virtualenv)
- Step 10: Start `start.py` via python3.9 venv manually (not crontab, systemd, supervisor or any type of script) - just for first time to see if there are any errors/problems and to make sure you will see everything.
- Step 11: If everything working go to **Pogo Auth** in MADmin and edit level of your accounts to real level (so 30+)
- Step 12: Password protect MADmin if not running via VPN/LAN *MADmin password/login*
- Step 13: Update PD and RGC on all devices - `async` have dedicated version of those programs. You can do it via Wizzard/MADmin Packages (if ATV), Jobs, manually - whatever you like more. Links to apks
- Step 14: Make sure `screendetection` is set to `True` in every Devices settings in MADmin (and in Shared settings/settings pools if using). Crucial for PTC/Google login process.

Multiply MITM RECEIVERS

If you run more than **15** devices you probably need this. In `master/legacy` you started more instances - in `async` you start more mitm receivers that process data within the same instance. Few examples how this works: https://github.com/Map-A-Droid/MAD/blob/async/asyncio_readme.md or https://github.com/spammer23/MAD/blob/async_quest_layers/async_SimpleSetup.md

This part is little more tricky as you need to start muliply mitm receivers and distribute data to those. In a nutshell you should put load balancer/proxy and make that connect to your mitm receivers. Links above should show you how to handle most common setups (apache2/nginx), but feel free to ask on Discord if you having any problems/questions regarding this.

2.9.3 PogoAuth changes

Due to latest N behavior changes (BSOD / maintenance screen) and limiting number of mon encounters per account within some <time period> there is now a need for changing accounts on devices. MAD can fully handle PTC accounts and semi-handle Google accounts. PogoAuth section is now a list/repository of all accounts you have. MAD uses this list to automagically select valid (non banned/non maintenance/non cooldown) accounts. You need to have proper account levels there - if you are just migrating it was imported with levels 0/1 and MAD won't login into those accounts when running *mon_mitm/iv_mitm/raids_mitm* - those need higher (30/8) levels. Please adjust those levels manually via MADmin or SQL query (`UPDATE settings_pogoauth SET level = 30`).

PTC only

If you running PTC only then you make sure levels in **Pogo Auth** settings are set to 30+. You also need to remember about PTC login limits so (beta-testing) `enable_login_tracking` in `config.ini` could be an option or running bunch of proxies to have different IPs there.

Google only

MAD (atm) does not handle directly login into Google Accounts so you either had them logged in earlier via autoconfig or you did it manually. It's like that mostly because Google is picky with security and there is a lot of different things that can go wrong if you decided to login multply account within short period of time - some extra checks, temp bans etc. Because of those limitations MAD now need to know **which account is on which device** so on top of having your accounts listed in **Pogo Auth** (with correct level!) you also need to tell MAD how to map does - go to MADmin Settings -> Devices and fill the `ggl_login_mail` with correct accounts. You don't do it via **Pogo Auth** section, you do it via `ggl_login_mail` in specific Device settings. Yes, you can have multiply accounts in `ggl_login_mail`, but those **need** to be already logged in on device. Remember about setting correct levels on those accounts. If you don't remember your Google password then just type anything in password box - PogoAuth does not support login via those passwords and MAD picks accountns by username - only autoconfig need valid Google Account password.

Mixed (PTC and Google)

MAD will first try to use Google accounts mapped via `ggl_login_mail` and then use PTC accounts if those Google one are on maintenance. Please read both bullet points above :-)

Minimum Level

All accounts in **Pogo Auth** section need to have correct level set up. MAD need accounts level 30+ for Quests/Mons so it won't even try to login into lower level accounts. Remember to set level manually if you migrated or set it correctly when adding new accounts. MAD **will** log into lower levels account if it's running **Levelup Quest Mode** and update/increase levels.

Maintenance/Flag/Hourglass

Accounts hit by BSOD / maintenance screen have current timestamp saved into database and they are deemed **not valid to use** for next 24 hours - this is for how long (most of the times, Niantic) accounts are not usuable at all. There are some icons you can hover/click in **Pogo Auth** section to give you an idea when it happen/what is the status.

2.9.4 MADmin password/login

Old system using `madmin_password` and `madmin_login` is gone - you should remove those entries from `config.ini`. You can now password-protect MADmin via built-in auth levels or externally via nginx/apache2. Both systems have pros and cons, so you should decide on one, there is no "better" system, but personally because I don't share my MADMin or don't have a public quest page I prefer the nginx/apache2 proxy.

- Using MAD built-in auth system:

If you decided to use built-in MAD system you need to add new user via MADMin Settings -> Auth with `MADMIN_ADMIN` permissions and enable/uncomment `madmin_enable_auth` in `config.ini`. Restart MAD and it's all done.

- Using nginx/apache2 proxy:

You need to use standard Basic Authentication for nginx/apache2. Example config for nginx is included in https://github.com/Map-A-Droid/MAD/blob/async/configs/examples/nginx/foo.conf#L56

```
auth_basic "Restricted";
# Please to use basic auth...
auth_basic_user_file /etc/nginx/.htpasswd_mad;
```

For apache2 it's very similar:

```
<Proxy *>
    Order deny,allow
    Allow from all
    Authtype Basic
    Authname "Password Required"
    AuthUserFile /etc/apache2/.htpasswd_mad
    Require valid-user
</Proxy>
```

To create .htpasswd_mad file you use `htpasswd` program (from `apache2-utils` system repository package) via

```
[sudo] htpasswd  -c /etc/apache2/.htpasswd_mad USERNAME_HERE
[sudo] htpasswd  -c /etc/nginx/.htpasswd_mad USERNAME_HERE
```

it will ask for password twice and then create a file for you.

Remember to restart nginx/apache2 after changes.

2.9.5 Common problems

I can't find X in config.ini, I am missing settings, where is madmin_enable_auth

Please open `config.ini.example` to see everything it's there and then copy-paste specific section/settings to `config.ini`.

`git pull` cannot overwrite **yours** config.ini because it would be a total mess and you would need to restore that file every update.

init mode

init mode settings? - init mode have a dedicated type now - just create new area using **Init scanner ``init``** mode. You can specify what **type** you are interested - `forts` will jump every ~500 meters and add all pokestops/gyms to database and hardly any spawnpoints as those are visibly only within ~50 meters. `mons` will jump every ~50 meters and add a lot of more spawnpoints, but it will have a lot of more jumps/stops/position on route.

unrecognized argument

unrecognized argument when starting MAD `start.py: error: unrecognized arguments: --madmin_user= --weather` it means that this arguments (`madmin_user`, `weather`) is need to be deleted from new `config.ini` as it is not supported anymore.

ortools

ortools ortools speedup route calculation but it's not `requierments.txt` by default so you just need to install it in your **venv**

Wizzard/APK problems

If you get **[W] Unable to save/upload apk: (pymysql.err.InterfaceError) (0, 'Not connected')** in logs while trying to Wizzard PD/POGO you most likely need to update MySQL/MariaDB settings in /etc/mysql/mariadb.conf.d/50-server.conf for max_allowed_packet to something like 256M, restart MySQL/MariaDB after that.