FakerNet

Release 0.7.0

Jacob Hartman

Jan 30, 2022

CONTENTS:

FakerNet is a framework to quickly build internet-like services rapidly for home labs, testing, and research. Instead of wasting time setting up DNS, web servers, certificate authorities, and email, FakerNet uses Docker and LXC to quickly spin up these services and servers without all the hassle.

CHAPTER

ONE

GETTING STARTED

This guide will help you through the steps of getting started with FakerNet. By the time you've completed the tasks here, you should have a working instance of FakerNet.

1.1 Network Design

The goal of FakerNet is to make it easy to create internet-like services, meaning we want whole networks to be able to use the services and servers FakerNet builds. While testing, you can just access the services from the host system but for other systems, the easiest way to use FakerNet-build services is to set the FakerNet host as the default gateway for the network. This makes it very simple for hosts to access what FakerNet builds.

For more details and information on more complex FakerNet setups, refer to *Network Design* for more details.

1.2 Installing

Once you've determined the FakerNet host's place in the network, you can move onto *Installation*.

Note: Be sure you have built the Docker and LXD images before you continue!

1.3 First Run

The first thing you need to do after installing FakerNet is perform a first-run configuration. This consists of getting the basic services built:

- A DNS server (the central server that all other FakerNet systems by default will forward to and yours should too)
- A MiniCA instance to generate certificates for other services.

Starting this process is as sample as running the console:

```
./fnconsole
```

This starts the console, which recognizes that its the first run. It will prompt you for a few things:

- A network address for the services to run on. Enter in the format `X.X.X.X/PREFIX`.

- The root-level domain for you fake internet services. This could be `fake` or `test`. This will be the root-level domain for all your services, so for example, if you put `test`, the certificate authority server will have the domain `ca.test` assigned to it automatically.
- The network address for the main DNS server
- The network address for the main certificate authority server

Once this is done, these services will be automatically built and configured and you will be put on the FakerNet console. Type `exit` if you want to exit. Other services you build will utilize these two initial services for DNS and certificates.

1.4 Configuring Other Hosts

If you have the FakerNet host as the default gateway, all hosts that use the gateway should be able to access FakerNet servers automatically. If not, you need to make sure that your network can route requests to the FakerNet host.

Regardless, you'll need to configure your hosts to use the main DNS server (the one built during First Run) for their domain server. This allows them to resolve FakerNet domains.

1.5 Starting Using FakerNet

Now that you've gotten FakerNet installed and configured, you can begin to use it to build services and servers and have your network access them.

- If you're looking for quick tutorials on how to build services, check out the *Services Tutorials*.
- For a overview of how you interact with FakerNet, check out *Using FakerNet* then the *CLI Usage*.
- To get started building modules, check out *Building Modules*.
- For configuring a FakerNet web server, see *Server*.

CHAPTER

TWO

NETWORK DESIGN

When using FakerNet, you will need to consider how hosts will integrate and utilize FakerNet services, as well as if and how FakerNet users will get further real internet access.

2.1 Integration Methods

There are two main ways you can integrate FakerNet into your network infrastructure, as a Gateway or side-loaded into the network.

2.1.1 Gateway

The easiest method, and the recommended one, is to make the system the default gateway for the networks you want to connect to the fake internet. All hosts will sit behind the FakerNet host and route everything through it, including any access to the real internet. This strategy gives you the most control over access to the FakerNet systems and allows to you to redirect traffic to the FakerNet hosts. (This is especially useful to redirect DNS traffic.) Essentially, FakerNet works like your ISP.

2.1.2 Side-loaded

Another method is utilize routing protocols to add the FakerNet networks to your existing routing infrastructure. You can use the Quagga that is installed for FakerNet or another method to add FakerNet's routes so that systems can access FakerNet systems. Setting up these routes goes beyond the realm of this documentation.

2.2 DNS Requirements

To take full advantage of FakerNet, hosts should point to, directly or indirectly, to the FakerNet main DNS server (the one created during setup). Either hosts should have it configured as its only primary DNS server (don't use other DNS servers, which might cause inconsistent DNS responses), or point to a DNS that utilizes the FakerNet DNS server. If you have the FakerNet host as the default gateway, you can also use the `redirect` module to force all DNS queries to the FakerNet primary DNS server.

2.3 Real Internet Access

Depending on your setup, you may or may not want access to real Internet resources in your environment.

2.3.1 No Internet

This can be simply done by using the Gateway method without connecting the FakerNet host to any further networks. The networking will end with the FakerNet host and all hosts in your environment will only have access to the FakerNet "internet." With this setup, you are free to use any IP ranges (including real public ranges) as you want, as well as any root DNS names you want. For example, you could configure FakerNet systems in the `8.8.8.0/24`, which would normally contain Google's public DNS, and use `.com` domains.

If the FakerNet host is connected to an external network for maintenance and access purposes, without any NAT rules, hosts will not be able to reach outside the FakerNet box. Some packets will reach out, since routing is enabled on the FakerNet host, but not be able to return due to a lack of NAT. For added safety and to stop these outbound packets, you can utilize `iptables` to block outbound traffic from the internal networks.

2.3.2 "Extended" Internet

If you are using the "side-load" method, this is practically the access already available. When using the gateway method, this can be achieved by adding NAT rules for the external interface, which can be done with the *iptables* module. For example, if the external interface is `ens18`, and you want to allow all ranges:

```
local> run iptables set_external_iface
local(iptables.set_external_iface)> set iface ens18
local(iptables.set_external_iface)> execute
OK
local(iptables.set_external_iface)> run iptables add_nat_allow
local(iptables.add_nat_allow)> set range *
local(iptables.add_nat_allow)> execute
OK
```

If you want only certain networks to be restricted from internet access, you could limit certain ranges. For example, the following will allow all other ranges except the `10.88.50.0/24` network (perhaps that is your internal network connected to lab devices):

```
> run iptables add_nat_allow
local(iptables.add_nat_allow)> set range !10.88.50.0/24
local(iptables.add_nat_allow)> execute
OK
```

A few other things should be kept in mind:

- The primary FakerNet DNS should be configured with forwarders so it can resolve external addresses. Note that misspelled or misconfigured DNS names may be sent to these forwarders.
- You will only be able to use private IP ranges in FakerNet, otherwise you risk making parts of the real internet unaccessible.
- You will only be able to use unused/test root DNS names, such as `fake` or `test`. Using root names like `com` risk making large swathes of the internet unaccessible.

2.3.3 Proxied Internet

This method, only possible when using FakerNet as a gateway, limits internet access to select hosts. This is done by restricting the NAT rules to certain hosts, such as an instance of the `tinyproxy` FakerNet module.

For example, if the `tinyproxy` instance is at `10.10.10.2`, configure it alone be to allowed through NAT (given you haven't used the rules above):

```
> run iptables add_nat_allow
local(iptables.add_nat_allow)> set range 10.10.10.2
local(iptables.add_nat_allow)> execute
OK
```

You can utilize the *iptables* module to create a wide-range of configurations using the `add_raw` and `add_raw_to_table`.

CHAPTER

THREE

INSTALLATION

Warning: During installation, the current user (the one running FakerNet) will be given access to commands that can used to gain root privileges if given unfettered access on a shell.

Contents

3.1 Script Installation

3.1.1 Ubuntu

An installation script for Ubuntu (tested on Ubuntu 18.04) is available in *scripts/install_ubuntu.sh*

Now go to *Firewall Rules* and *Build Docker and LXD Images*.

3.2 Manual Installation

3.2.1 1. Install Dependencies

These are:

- LXD
- Open vSwitch
- Python 3.5 or higher, with pip and venv support
- git
- quagga routing services
- traceroute
- Python Development files (e.g. *python3-dev* on Ubuntu)

For Ubuntu, (which FakerNet has been tested on), this is the command:

```
apt-get install git python3-venv python3-pip openvswitch-switch lxd quagga traceroute
```

3.2.2 2. Install Docker

Install Docker as indicated on their website.

3.2.3 3. Setup Groups

Ensure your user is in the following groups:

- `lxd`
- `docker`
- `quaggavty`

Note: Be sure to re-login so that group permissions come into effect.

3.2.4 4. Configure Docker

Edit Docker's configuration to do uid remapping and user namespaces. This is for both security and to allow mapping of configuration files in Docker containers.

In */etc/docker/daemon.json* add the following (the file usually needs to be made):

```
{
  "userns-remap": "default"
}
```

Restart the Docker service, Docker will create the *dockremap* user and setup subuids properly.

3.2.5 5. Configure ID Mappings

To ensure the root user in the containers maps to our current user that will run FakerNet, modify */etc/[ug]id*. In both */etc/subuid* and */etc/subgid* set the following.afterwards:

```
dockremap:1000:1
```

Restart Docker

3.2.6 6. Configure `sudo`

FakerNet needs to run certain commands as root to manage networking for the containers. To do this without running the entire framework as root, we can use *sudo* rules to give the current user access to the specific commands. These commands are:

- `ovs-vsctl`: For controlling Open vSwitch
- `ovs-docker`: For connecting Docker images to Open vSwitch switches
- `iptables`: For making automatic redirects
- `ip`: For controlling interfaces

```
# Example sudoers entries. Paths may differ in your case.
user ALL=(ALL) NOPASSWD: /usr/bin/ovs-vsctl
user ALL=(ALL) NOPASSWD: /usr/bin/ovs-docker
user ALL=(ALL) NOPASSWD: /sbin/iptables
user ALL=(ALL) NOPASSWD: /sbin/ip
```

Warning: Note these commands can give the user root privileges (apart from the possibility for root privileges from Docker and LXD), so be aware of the user you are giving these controls to and restrict access to the account.

3.2.7 7. Get FakerNet

Note: If you haven't re-logged in to activated the new groups on the current user, do that now.

Note: If you haven't configured LXD, run `lxd init` now as root. The defaults will usually suffice, but don't create a managed switch during LXD setup.

Git clone the FakerNet repo and enter the root directory:

```
git clone https://github.com/bocajspear1/fakernet.git
cd fakernet
```

3.2.8 8. Install Python Dependencies

Create a virtualenv and activate it, then install dependencies:

```
python3 -m venv ./venv
. ./venv/bin/activate
pip3 install -r requirements.txt
```

3.3 Firewall Rules

Docker sets the default iptables forward rule to drop. FakerNet will set the `FORWARD` table to `ACCEPT` on start to fix this.

3.4 Build Docker and LXD Images

Once everything is installed, you'll need to tell FakerNet to build the necessary Docker and LXD images. By pre-building the base images, this allows FakerNet to be portable into internet-restricted environments after the installation process is complete.

Run the build process using the following commands:

```
. ./venv/bin/activate
python3 build.py
```

3.5 Finished

Congratulations, FakerNet is now set up and configured! For how to use FakerNet now, go to *Using FakerNet*

CHAPTER

FOUR

USING FAKERNET

Once FakerNet is installed, how do we utilize the framework to get our servers built? First, we need to take a quick look at how the framework is glued together.

4.1 Modules and Functions

Service and server building, configuration, and removal functionality is built into **modules**. Every module exposes a series of **functions** that perform a certain, single task, such as:

- Creating a server
- Adding a DNS record
- Stopping a server

This modular structure allows modules to call other modules and so forth so that functionality isn't reimplemented constantly, while being accessible and flexible. In FakerNet, this combination of function and module is referred to usually through the form:

```
<MODULE>.<FUNCTION>
```

4.2 Accessing Functions

Modules and their functions can be run through two main ways:

1. Locally: The functions are directly called locally, no server is involved. Good for smaller setups and testing.
2. A REST API server: The functions are called through a REST API server, usually running as a service. Good for more permanent setups and remote systems. (See *Server*)

To access either of these methods is most commonly through the FakerNet console. See *CLI Usage*. (You could also call the REST API directly)

4.3 Saving and Restoring

FakerNet supports saving and restoring running servers, creating a state of what is up and what is down. This can be used with the `save` and `restore` commands in the CLI. Without any arguments, the save state is named `default`, and is stored in `saves/default.json`. This will be the state loaded when the FakerNet server starts. Use these commands with a string to give the state a name or restore a named state.

```
save example
restore example
```

The named saves are stored in `saves/<NAME>.json`

CHAPTER

FIVE

CLI USAGE

The primary method of using FakerNet is using the FakerNet console. The console can run without or with a FakerNet API server.

Note: You will need to run the console without an API server at least once to perform initial configuration.

5.1 Starting the Console

To start the console:

```
./fnconsole
```

To connect to a remote system:

```
./fnconsole -s <SERVER IP>
```

You will need to login with a username and password. See *Server* for details on server setup and logins.

For local API servers, the console will automatically attempt to connect to a local server on port 5051, so you will not need the `-s` parameter.

5.2 Using the Console

The FakerNet console uses the prompt_toolkit framework, which allows for a number of features like autocomplete and command history.

- The prompt shows the address of the API server it's using, or `local` for no API server.
- Use the up and down arrow keys to go back and forth through the command history.
- Use TAB to autocomplete commands
- Commands will appear as you type. When one of these lists is open, you can use the arrow keys to select, and tab to insert the completion.

5.3 Console Modes

The console operates in two primary level, **main** level and **function** level. FakerNet is built around functions provided by modules, you will spend most of your time running in **function** mode. Using the console, you will call module functions to perform certain actions, such as creating, stop, and starting servers.

5.3.1 Main Mode

Main level is the top level, and the mode you start in. This mode performs FakerNet-wide operations as well as authentication operations. You can run the following commands:

- `run <MODULE> <FUNCTION>`: This is used to call a function, and given a module and function name, will run the function, or put you into function mode.
- `list_all`: This lists all servers running from all modules.
- `exit`: This exits the console.
- `save`: This saves the current state of up and down servers. An option name can be set afterwards to name the state. The default name is `default`.
- `restore`: This restores from a state save. An option name can be set afterwards to set the state to load. The default name is `default`.
- `useradd`: Add a user to for API authentication.
- `userls`: List users for API authentication.
- `userdel`: Remove users from API authentication.

5.3.2 Function Mode

This is where most of the magic happens. FakerNet breaks up functionality into modules. This allows modules to call other modules so we aren't reimplementing stuff unnecessarily. The console provides access to the functions from these modules, so we have control to create and destroy servers, configure them, etc.

This mode is entered when running a module function that requires parameters. Functions that don't need parameters will just run the function. The module and function name will appear in the prompt when in the function level:

For example:

```
local(dns.add_record)>
```

The following commands are available:

- `show`: Shows a brief overview of the function. This includes a short description of the function and the current function's variables and their values
- `set <VAR_NAME> <VALUE>`: This sets a value for a function parameter.
- `unset <VAR_NAME>`: This clears a value for a function parameter.
- `back`: This goes back to the main level
- `execute`: Executes the function
- `run <MODULE> <FUNCTION>`: Call another function. This also clears any currently set values for the current function.

CHAPTER

SIX

SERVER

FakerNet has a web server that is run with the `./fnserver` command. This server can also be run as a service, which allows FakerNet services to started on boot.

6.1 Adding the Service

If you used the install script, the service should be installed for you already. If not, for systemd-based distros, you can use the `.service` file template in the `scripts` directory.

6.2 Starting Servers On Boot

To restore services on-boot, you will need to create a default restore point. This can be done with the `save` command on the console. Once this is created, FakerNet will restore the running services to the up/down status when the `save` command was run.

6.3 Web Server

The web server can be accessed on port `5051` using TLS from your browser. It provides both a simple web-based UI to interact with FakerNet, as well as a REST API. You will need to create users to access the web interface from anywhere but the local host. Authentication is currently done with HTTP basic authentication.

6.3.1 Users

To add a user, you will need to open the console:

```
./fnconsole
```

Then, use the `useradd` command without any options, it will prompt for username and password for the new user. The password is hidden while typing it.

```
127.0.0.1> useradd
username> testuser
password>
password (again)>
User Added
```

The user will be immediately available for use. You can add uses regardless of the server currently running or not. If the server is on, the console automatically communciates with it.

6.3.2 Web UI

The web UI is fairly simple and minimalistic. It has three main pages:

- **Status**: This provides the current CPU, memory, and disk usage, as well as list of currently running servers.
- **Run**: This page allows you to call module functions. Select the module and function from the dropdowns, fill in the textboxes with the necessary options, then press `Submit` to run. The results will appear below the form.
- **API Docs**: This provides a Swagger web interface to show the endpoints for the REST API. This page is fully interactive, allowing calls to the API to be performed right there.

6.3.3 Web API

FakerNet provides a REST API for your integration needs. This API is also used by the console when not in local mode and the Web UI. Reference the Swagger page on the web server for API documentation.

CHAPTER

SEVEN

TUTORIALS

7.1 Services Tutorials

These tutorials will help you get started building services with FakerNet.

Table of Contents

7.1.1 Adding a Subdomain Server

Adding a subdomain server is simply one module function in the FakerNet console.

First, open the console:

```
./fnconsole
```

Then use the `smart_add_subdomain_server` function in the `dns` module:

```
run dns smart_add_subdomain_server
```

Set the options with the `set` command, ensure that there is a parent domain when you set the FQDN. For example, if you used `test` for the root domain, you can create a subdomain like the example:

```
local> run dns smart_add_subdomain_server
dns.smart_add_subdomain_server: Add subdomain server, automatically setting up root␣
↪server to point to it
local(dns.smart_add_subdomain_server)> set fqdn subdomain.test
local(dns.smart_add_subdomain_server)> set ip_addr 172.16.3.30
```

Then run `execute`:

```
execute
```

Now, if you exit out of the console, you can use `dig` to see our server is set up:

```
$ dig @172.16.3.30 ns1.subdomain.test

; <<>> DiG 9.16.1-Ubuntu <<>> @172.16.3.30 ns1.subdomain.test
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36394
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 50319db6aad37a7501000000617b6c62114c6943afdcc411 (good)
;; QUESTION SECTION:
;ns1.subdomain.test.            IN      A

;; ANSWER SECTION:
ns1.subdomain.test.     604800  IN      A       172.16.3.30

;; Query time: 4 msec
;; SERVER: 172.16.3.30#53(172.16.3.30)
;; WHEN: Thu Oct 28 23:37:06 EDT 2021
;; MSG SIZE  rcvd: 91
```

We can also query the root domain server (the one set up in setup), for example, if you set the ip to `172.16.3.2`:

```
$ dig @172.16.3.2 ns1.subdomain.test

; <<>> DiG 9.16.1-Ubuntu <<>> @172.16.3.2 ns1.subdomain.test
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37913
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 4c9842f214f44d1201000000617b6c6cfd0cf3a1dffa30d6 (good)
;; QUESTION SECTION:
;ns1.subdomain.test.            IN      A

;; ANSWER SECTION:
ns1.subdomain.test.     604790  IN      A       172.16.3.30

;; Query time: 0 msec
;; SERVER: 172.16.3.2#53(172.16.3.2)
;; WHEN: Thu Oct 28 23:37:16 EDT 2021
;; MSG SIZE  rcvd: 91
```

7.1.2 External DNS Resolution

If you want your main DNS server to resolve external addresses, you'll need to configure forwarders for it.

First, open the console:

```
./fnconsole
```

Then use the `add_forwarder` function in the `dns` module:

```
run dns add_forwarder
```

With the `set` command, configure the parameters (the ID of the main DNS server is 1):

```
set id 1
set ip_addr <FORWARDER IP>
```

Then call `execute` to run the function:

```
execute
```

7.1.3 Creating a Mail Server

Creating a mail server is easy with the `simplemail` module. It sets up the needed DNS entries and uses RoundCube to provide web-based email access.

First, open the console:

```
./fnconsole
```

Then use the `add_server` function in the `dns` module:

```
local> run simplemail add_server
```

Then we can set the necessary options:

- `fqdn`: The full domain name of the mail server (like `mail.domain.test`)
- `mail_domain`: The domain the server will send and recieve mail for. This is the domain at the end of an email address. (like user@**domain.test**). This can be the same as the `fqdn`.
- `ip_addr`: The IP address of the mail server

For example: .. code-block:

```
local(simplemail.add_server)> set fqdn mail.test
local(simplemail.add_server)> set mail_domain mail.test
local(simplemail.add_server)> set ip_addr 172.16.3.32
```

Then run `execute`:

```
local(simplemail.add_server)>  execute
```

You should get the output of `OK` if everything setup correctly.

7.2 Building Modules

7.2.1 Base Class

All modules are Python classes that inherit from a central base module, which contains helpful methods.

All classes should include something like this:

```
from lib.base_module import BaseModule


class MyModule(BaseModule):

    def __init__(self, mm):
        self.mm = mm

    __FUNCS__ = {
        "list": {
            "_desc": "View network allocations"
        },
        ...
    }

    __SHORTNAME__  = "mymodule"
    __DESC__ = "A description, appears in documentation"
    __AUTHOR__ = "You"

    def run(self, func, **kwargs) :
        dbc = self.mm.db.cursor()
        if func == "somefunc":
            pass

    ...

__MODULE__ = MyModule
```

7.2.2 `__FUNCS__`

This is a dict that contains info about the functions the module supports. The format is: .. code-block:

```
"function_name": {
    "param_name": "param_type"
}
```

This dict allows FakerNet to verify parameters and automatically build the autocomplete for the console.

A special param_name is `_desc`, which is the description of the function, and does not count as a parameter when verifiying parameters and in the console.

See *Parameter Types* for available parameter types.

7.2.3 `__SHORTNAME__`

This is the name of the module used in the console and when other modules refer to this module. It must be unique.

7.2.4 Required Functions

- `run`: This function is called with the function name as a string as the first argument, then kwargs for the paramaters. Be sure to match the parameters to the function definition in *__FUNCS__*
- `check`: This function is called on console startup to ensure the database tables and other configurations are set for the module.
- `build`: This function is called to build it the base Docker image for the service. Modules that do not make a Docker image should just make an empty function with just *pass*.
- `list_all`: This function returns a list of server instances. Used for console and API *list_all* and *list_running* commands.
- `save`: This function returns data used for restoring servers later. Used for the console and API *save* command.
- `restore`: This function takes in data from a save file and uses it to return FakerNet to a particular state. Used for the *restore* command.

7.2.5 `run` Function

This function is the main function of the module and contains the primary actions and activities of the module. A *if/elif/else* determines the function from the first parameters.

7.2.6 Examples

Simple modules, such as `pwndrop` and `inspircd` should work well as examples for basic modules.

CHAPTER

EIGHT

MODULES

The following is the currently available FakerNet modules.

8.1 pastebin-bepasty

Creates a Pastebin-clone using the Python-based Bepasty.

Available on GitHub, documentation here.

See *Parameter Types* for parameter types.

8.1.1 list

View all Bepasty servers

8.1.2 add_server

Delete a pastebin server

Table 1: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

8.1.3 remove_server

Remove a Pastebin server

Table 2: Parameters

Name	Type
id	INTEGER

8.1.4 start_server

Start a pastebin server

Table 3: Parameters

Name	Type
id	INTEGER

8.1.5 stop_server

Stop a pastebin server

Table 4: Parameters

Name	Type
id	INTEGER

8.2 inspircd

See *Parameter Types* for parameter types.

8.2.1 list

View all inspircd servers

8.2.2 add_server

Add a inspircd server

Table 5: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

8.2.3 remove_server

Delete a inspircd server

Table 6: Parameters

Name	Type
id	INTEGER

8.2.4 start_server

Start a inspircd server

Table 7: Parameters

Name	Type
id	INTEGER

8.2.5 stop_server

Start a inspircd server

Table 8: Parameters

Name	Type
id	INTEGER

8.3 lxd

This module manages LXD containers, which provide a more VM-like experience as compared to the Docker containers most services are in.

The hostname and container name is the `fqdn` with the dots replaced with dashes.

LXD addresses are set manually by FakerNet on start, so if you start the container outside FakerNet you will not get your address properly. (This is due to the built in LXD network management utilizing DHCP, which caused limitations). Configuring the container to have a static IP through its own startup scripts is currently left up to the user, as supporting all the different methods of setting a static IP in the container would be a real pain.

See *Parameter Types* for parameter types.

8.3.1 list

View containers

8.3.2 add_container

Add a new container

Table 9: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP_ADDR
password	PASSWORD
template	TEXT

8.3.3 remove_container

Table 10: Parameters

Name	Type
id	INTEGER

8.3.4 start_container

Table 11: Parameters

Name	Type
id	INTEGER

8.3.5 stop_container

Table 12: Parameters

Name	Type
id	INTEGER

8.3.6 list_templates

List available LXD templates

8.3.7 add_template

Add template by image name

Table 13: Parameters

Name	Type
image_name	TEXT
template_name	TEXT

8.3.8 remove_template

Remove template by ID

Table 14: Parameters

Name	Type
id	INTEGER

8.4 nethop

Nethop creates new networks and sets up a simple Alpine Linux router to act as its gateway. This allows for multi-tiered networks instead of just a flat one.

The router is a LXD container, not a Docker container, and runs Quagga that distributes routes currently by RIPv2.

If routes are having issues being distributed on the host, try restarting the Quagga service first.

See *Parameter Types* for parameter types.

8.4.1 list

View network hops

8.4.2 add_network_hop

Add a network hop

Table 15: Parameters

Name	Type
front_ip	IP
fqdn	TEXT
net_addr	IP_NETWORK
description	TEXT
switch	TEXT

8.4.3 remove_network_hop

Remove a network hop

Table 16: Parameters

Name	Type
id	INTEGER

8.4.4 start_hop

Start a hop router between networks

Table 17: Parameters

Name	Type
id	INTEGER

8.4.5 stop_hop

Stop a hop router between networks. Will cut off communications.

Table 18: Parameters

Name	Type
id	INTEGER

8.5 iptables

This module allows management of iptables rules from FakerNet. These rules will always be added when FakerNet starts. This makes it useful for setting up things like NAT.

Warning: Rules are always added at the top. Use the `list_order` to get a better idea of the order the rules will be added.

Warning: Rules are not removed when FakerNet stops

Note: `!` can be used in `add_nat_allow` to do a "not" of the range

See *Parameter Types* for parameter types.

8.5.1 list

View iptables rules

8.5.2 list_order

View iptables rules in order they will appear (opposite of addition)

8.5.3 show_ifaces

Show configured interfaces

8.5.4 set_external_iface

Set the external interface (used for NAT)

Table 19: Parameters

Name	Type
iface	SIMPLE_STRING

8.5.5 set_internal_iface

Set the internal inferface

Table 20: Parameters

Name	Type
iface	SIMPLE_STRING

8.5.6 add_nat_allow

Add NAT rule (adds to top of chain)

Table 21: Parameters

Name	Type
range	TEXT

8.5.7 add_raw

Add raw rule (adds to top of chain)

Table 22: Parameters

Name	Type
cmd	ADVTEXT
chain	SIMPLE_STRING

8.5.8 add_raw_to_table

Add rule to table (adds to top of chain)

Table 23: Parameters

Name	Type
cmd	ADVTEXT
table	SIMPLE_STRING
chain	SIMPLE_STRING

8.5.9 remove_rule

Remove a iptables rule

Table 24: Parameters

Name	Type
id	INTEGER

8.6 pwndrop

A file-hosting application oriented to delivering attack payloads.

Made by Kuba Gretzky and on GitHub.

See *Parameter Types* for parameter types.

8.6.1 list

View all pwndrop servers

8.6.2 remove_server

Delete a pwndrop server

Table 25: Parameters

Name	Type
id	INTEGER

8.6.3 add_server

Add a pwndrop server

Table 26: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

8.6.4 start_server

Start a pwndrop server

Table 27: Parameters

Name	Type
id	INTEGER

8.6.5 stop_server

Stop a pwndrop server

Table 28: Parameters

Name	Type
id	INTEGER

8.7 dns

This module creates and controls a BIND DNS server in an Alpine instance.

Each server has a primary zone that is configured at the creation of the server. This name is used by the module to automatically determine where DNS names should go. Servers can have multiple zones, but then you cannot use the automatic server detection and have to manually indicate where a domain name needs to go.

A domain name must have a zone defined for it, otherwise it will fail to allocate. e.g. if you don't have a server that has the *nope* zone, you will unable to create a domain of *something.nope*.

See *Parameter Types* for parameter types.

8.7.1 list

View all DNS servers

8.7.2 remove_server

Delete a DNS server

Table 29: Parameters

Name	Type
id	INTEGER

8.7.3 add_server

Add a DNS server

Table 30: Parameters

Name	Type
ip_addr	IP
description	TEXT
domain	TEXT

8.7.4 add_zone

Add a DNS zone

Table 31: Parameters

Name	Type
id	INTEGER
zone	TEXT
direction	['fwd', 'rev']

8.7.5 smart_add_record

Add a record to a DNS server, detecting server and zone

Table 32: Parameters

Name	Type
direction	['fwd', 'rev']
type	TEXT
fqdn	TEXT
value	ADVTEXT
autocreate	BOOLEAN

8.7.6 smart_remove_record

Add a record to a DNS server, detecting server and zone

Table 33: Parameters

Name	Type
direction	['fwd', 'rev']
type	TEXT
fqdn	TEXT
value	ADVTEXT

8.7.7 add_record

Add a record to a DNS server

Table 34: Parameters

Name	Type
id	INTEGER
zone	TEXT
direction	['fwd', 'rev']
type	TEXT
name	TEXT
value	ADVTEXT

8.7.8 remove_record

Remove a record from a DNS server

Table 35: Parameters

Name	Type
id	INTEGER
zone	TEXT
direction	[‘fwd’, ‘rev’]
type	TEXT
name	TEXT
value	ADVTEXT

8.7.9 add_host

Add a host to a DNS server

Table 36: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP_ADDR

8.7.10 remove_host

Remove a host to a DNS server

Table 37: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP_ADDR

8.7.11 start_server

Start a DNS server

Table 38: Parameters

Name	Type
id	INTEGER

8.7.12 stop_server

Stop a DNS server

Table 39: Parameters

Name	Type
id	INTEGER

8.7.13 get_server

Get info on a DNS server

Table 40: Parameters

Name	Type
id	INTEGER

8.7.14 list_forwarders

View forwarders for DNS server

Table 41: Parameters

Name	Type
id	INTEGER

8.7.15 add_forwarder

Add forwarder to DNS server

Table 42: Parameters

Name	Type
id	INTEGER
ip_addr	IP_ADDR

8.7.16 remove_forwarder

Remove forwarder from DNS server

Table 43: Parameters

Name	Type
id	INTEGER
ip_addr	IP_ADDR

8.7.17 smart_add_subdomain_server

Add subdomain server, automatically setting up root server to point to it

Table 44: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP_ADDR

8.7.18 smart_remove_subdomain_server

Remove subdomain server, automatically deleting entries in the parent server

Table 45: Parameters

Name	Type
id	INTEGER

8.7.19 smart_add_root_server

Add a new root domain server (e.g. .com or .net), automatically setting up root server to point to it

Table 46: Parameters

Name	Type
root_name	TEXT
ip_addr	IP_ADDR

8.7.20 smart_remove_root_server

Remove root domain server (e.g. .com or .net), automatically deleting entries in the parent server

Table 47: Parameters

Name	Type
id	INTEGER

8.7.21 smart_add_external_subdomain

Add subdomain that points to an external DNS server

Table 48: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP_ADDR

8.7.22 smart_remove_external_subdomain

Add subdomain that points to an external DNS server

Table 49: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP_ADDR

8.8 webdavalpine

This module creates a Apache-based WebDAV server on an Alpine Linux instance. Public files are accessed at `<SERVER>/files/public` while other paths require credentials. A `admin` user is created on build and their password is located in the `webdav` subdirectory in the server's working directory. For example, with server with an ID of 1: `work/webdavalpine/1/webdav/admin.pass`

8.8.1 References

- Apache WebDAV Configuration
- configure apache/webdav readonly for user x, read/write for user y

See *Parameter Types* for parameter types.

list

View all WebDAV servers

add_server

Delete a WebDAV server

Table 50: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

remove_server

Remove a WebDAV server

Table 51: Parameters

Name	Type
id	INTEGER

start_server

Start a WebDAV server

Table 52: Parameters

Name	Type
id	INTEGER

stop_server

Start a WebDAV server

Table 53: Parameters

Name	Type
id	INTEGER

8.9 netreserve

See *Parameter Types* for parameter types.

8.9.1 list

View network allocations

8.9.2 get

Get network reservation info

Table 54: Parameters

Name	Type
ip_id	INT

8.9.3 add_hop_network

Add network allocation

Table 55: Parameters

Name	Type
net_addr	IP_NETWORK
description	TEXT
switch	SIMPLE_STRING

8.9.4 add_network

Add network allocation

Table 56: Parameters

Name	Type
net_addr	IP_NETWORK
description	TEXT
switch	SIMPLE_STRING

8.9.5 remove_network

Delete a network allocation

Table 57: Parameters

Name	Type
id	INTEGER

8.9.6 get_network_switch

Get the switch for a network

Table 58: Parameters

Name	Type
net_addr	IP_NETWORK

8.9.7 get_network_by_switch

Get the switch for a network

Table 59: Parameters

Name	Type
switch	SIMPLE_STRING

8.9.8 get_ip_switch

Get the switch for a network

Table 60: Parameters

Name	Type
ip_addr	IP

8.9.9 get_ip_network

Get the mask for a network

Table 61: Parameters

Name	Type
ip_addr	IP

8.9.10 is_hop_network_by_switch

Check if a network is a hop network (behind a hop router) by switch name

Table 62: Parameters

Name	Type
switch	SIMPLE_STRING

8.10 redirect

See *Parameter Types* for parameter types.

8.10.1 enable_dns_redirect

Enable redirecting all DNS to primary DNS server

Table 63: Parameters

Name	Type
interface	SIMPLE_STRING

8.10.2 disable_dns_redirect

Disable redirecting all DNS to primary DNS server

Table 64: Parameters

Name	Type
interface	SIMPLE_STRING

8.11 ipreserve

This module manages IP reservations in the defined networks, ensuring that IPs selected do not overlap.

A network must be defined in `netreserve` that contains the IP for a reservation. Otherwise, an error will be returned.

See *Parameter Types* for parameter types.

8.11.1 list_ips

View IP allocations

8.11.2 get

Get IP reservation info

Table 65: Parameters

Name	Type
ip_id	INT

8.11.3 add_ip

Add an IP reservation

Table 66: Parameters

Name	Type
ip_addr	IP_ADDR
description	TEXT

8.11.4 remove_ip

Remove an IP reservation

Table 67: Parameters

Name	Type
ip_addr	IP_ADDR

8.12 tinyproxy

This module uses the tinyproxy application to provide HTTP proxying capabilities. This could be used for relay internally or, with the proper iptables rules, provide external internet access to select systems in the environment.

If you allow FakerNet services internet access while blocking certain internal networks, tinyproxy could be used for systems to temporarily gain at least web-based internet access by pointing to the proxy.

See *Parameter Types* for parameter types.

8.12.1 list

View all tinyproxy servers

8.12.2 add_server

Add a tinyproxy server

Table 68: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

8.12.3 remove_server

Delete a tinyproxy server

Table 69: Parameters

Name	Type
id	INTEGER

8.12.4 start_server

Start a tinyproxy server

Table 70: Parameters

Name	Type
id	INTEGER

8.12.5 stop_server

Start a tinyproxy server

Table 71: Parameters

Name	Type
id	INTEGER

8.13 minica

MiniCA is a small, Go-based CA web application used to generate certificates for FakerNet services. It has a single password as its authentication, so don't use this in any production system or untrusted network.

The password for creating certs is located in `work/minica/<SERVER_ID>/ca.pass` externally or `/etc/minica/certs/ca.pass` in the container.

The web interface is available as HTTPS (signed by itself) at the container's IP address. You'll need to upload a CSR and enter the CA password.

Source for the CA server is available on GitHub.

See *Parameter Types* for parameter types.

8.13.1 list

View all CA servers

8.13.2 remove_server

Delete a CA server

Table 72: Parameters

Name	Type
id	INTEGER

8.13.3 add_server

Add a CA server

Table 73: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

8.13.4 get_server

Get info on a CA server

Table 74: Parameters

Name	Type
id	INTEGER

8.13.5 generate_host_cert

Generate a key and signed certificate

Table 75: Parameters

Name	Type
id	INTEGER
fqdn	TEXT

8.13.6 get_ca_cert

Get a server's CA cert

Table 76: Parameters

Name	Type
id	INTEGER
type	TEXT

8.13.7 start_server

Start a CA server

Table 77: Parameters

Name	Type
id	INTEGER

8.13.8 stop_server

Stop a CA server

Table 78: Parameters

Name	Type
id	INTEGER

8.14 simplemail

Simplemail is a basic mail server that uses Postfix (for SMTP) and Dovecot (for IMAP). It also contains a webserver that runs Roundcube and a simple PHP application to add more users to the mail server. This does not require and authorization to create the account, so anybody can create one.

The webserver is HTTPS, so access it at: .. code-block:

```
https://<SERVER_IP>
```

The account creator is available at: .. code-block:

```
https://<SERVER_IP>/newaccount.php
```

See *Parameter Types* for parameter types.

8.14.1 list

View all SimpleMail servers

8.14.2 remove_server

Delete a SimpleMail server

Table 79: Parameters

Name	Type
id	INTEGER

8.14.3 add_server

Add a SimpleMail server

Table 80: Parameters

Name	Type
fqdn	TEXT
mail_domain	TEXT
ip_addr	IP

8.14.4 start_server

Start a SimpleMail server

Table 81: Parameters

Name	Type
id	INTEGER

8.14.5 stop_server

Start a SimpleMail server

Table 82: Parameters

Name	Type
id	INTEGER

8.15 init

Note: This module should not be run independently. It is called when the console starts.

See *Parameter Types* for parameter types.

8.16 external

This module manages external hosts and networks. These are hosts that are not managed by FakerNet, but you want to use the DNS and IP allocation functionality in FakerNet to connect them to infrastructure. Be sure that the networks allocated can be accessed by the source device, even if they can't access FakerNet.

Usually, you'd use this for external VMs and containers.

See *Parameter Types* for parameter types.

8.16.1 list_hosts

View external hosts

8.16.2 list_networks

View external networks

8.16.3 add_external_host

Add an external host

Table 83: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP
host_desc	TEXT

8.16.4 remove_external_host

Remove an external host

Table 84: Parameters

Name	Type
id	INTEGER

8.16.5 add_external_network

Add an external network (wrapper for netreserve)

Table 85: Parameters

Name	Type
net_addr	IP_NETWORK
description	TEXT

8.16.6 remove_external_network

Remove an external network (wrapper for netreserve)

Table 86: Parameters

Name	Type
id	INTEGER

8.17 mattermost

See *Parameter Types* for parameter types.

8.17.1 list

View all Mattermost servers

8.17.2 remove_server

Delete a Mattermost server

Table 87: Parameters

Name	Type
id	INTEGER

8.17.3 add_server

Add a Mattermost server

Table 88: Parameters

Name	Type
fqdn	TEXT
ip_addr	IP

8.17.4 start_server

Start a Mattermost server

Table 89: Parameters

Name	Type
id	INTEGER

8.17.5 stop_server

Stop a Mattermost server

Table 90: Parameters

Name	Type
id	INTEGER

CHAPTER

NINE

REFERENCE

9.1 Parameter Types

The following is a reference of parameter types that are valid in FakerNet parameters.

Table 1: Parameters

Type	Description	Example
INTEGER	A simple, non-decimal number	1 34
DECIMAL	A decimal number	1.2 88.234,2
TEXT	Basic text, limited to letters, numbers, spaces, tabs, and the following: .,-_:=/#@!*	this.dns hello there
ADVTEXT	Text with fewer limits, limited to letters, numbers, spaces, tabs, and the following: .,-_!@#$%^&*()_+<>?"':|[]{}. Be much more cautious with this	this.dns hello there
SIMPLE_STRING	Text limited to letters and numbers	this1 hello
IP	An IP address	1.1.1.1 192.168.7.6
IP_ADDR	Same as IP	1.1.1.1 192.168.7.6
IP_NETWORK	An IP network with prefix length	1.1.1.0/24 192.168.7.0/16
BOOLEAN	A boolean as a string	true false
list	A selection from the list	

CHAPTER TEN

INDICES AND TABLES

- genindex
- modindex
- search