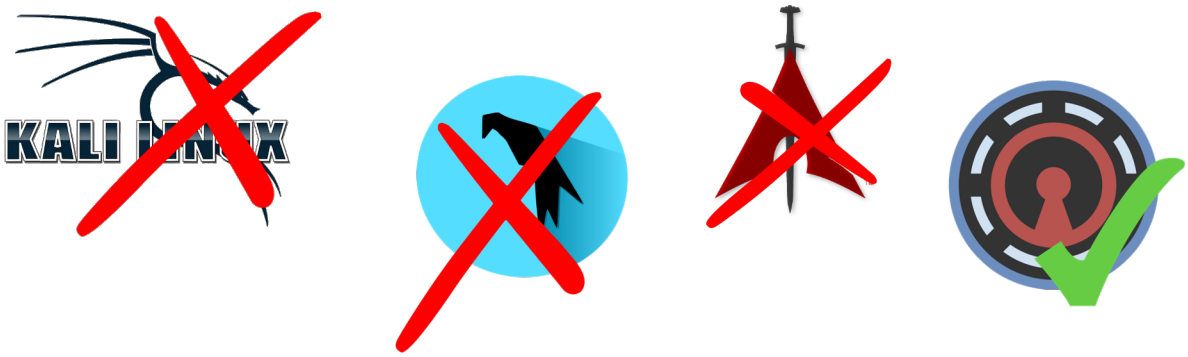
Exegol

Shutdown & Dramelac

Apr 17, 2024

GETTING STARTED

Exegol is a community-driven hacking environment, powerful and yet simple enough to be used by anyone in day to day engagements. Exegol is the best solution to deploy powerful hacking environments securely, easily, professionally. No more unstable, not-so-security-focused systems lacking major offensive tools. Kali Linux (and similar alternatives) are great toolboxes for learners, students and junior pentesters. But professionals have different needs, and their context require a whole new design.



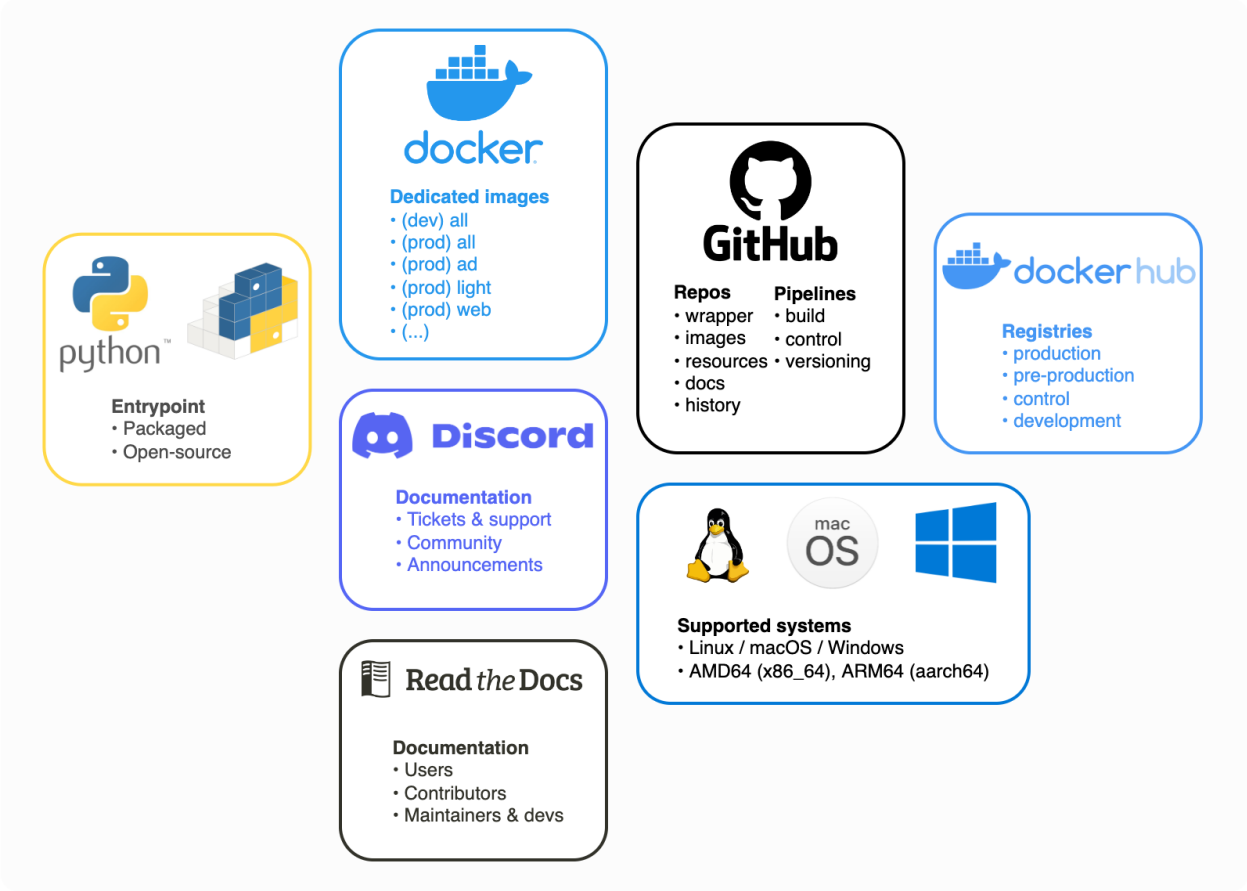
Exegol fits pentesters, CTF players, bug bounty hunters, researchers, beginners and advanced users, defenders, from stylish macOS users and corporate Windows pros to UNIX-like power users.

> **Warning:** This documentation is a work in progress. We are actively writing it, but if there are things you'd like to be documented in priority, feel free to request in on the GitHub Repo or in the Discord server.

CHAPTER

ONE

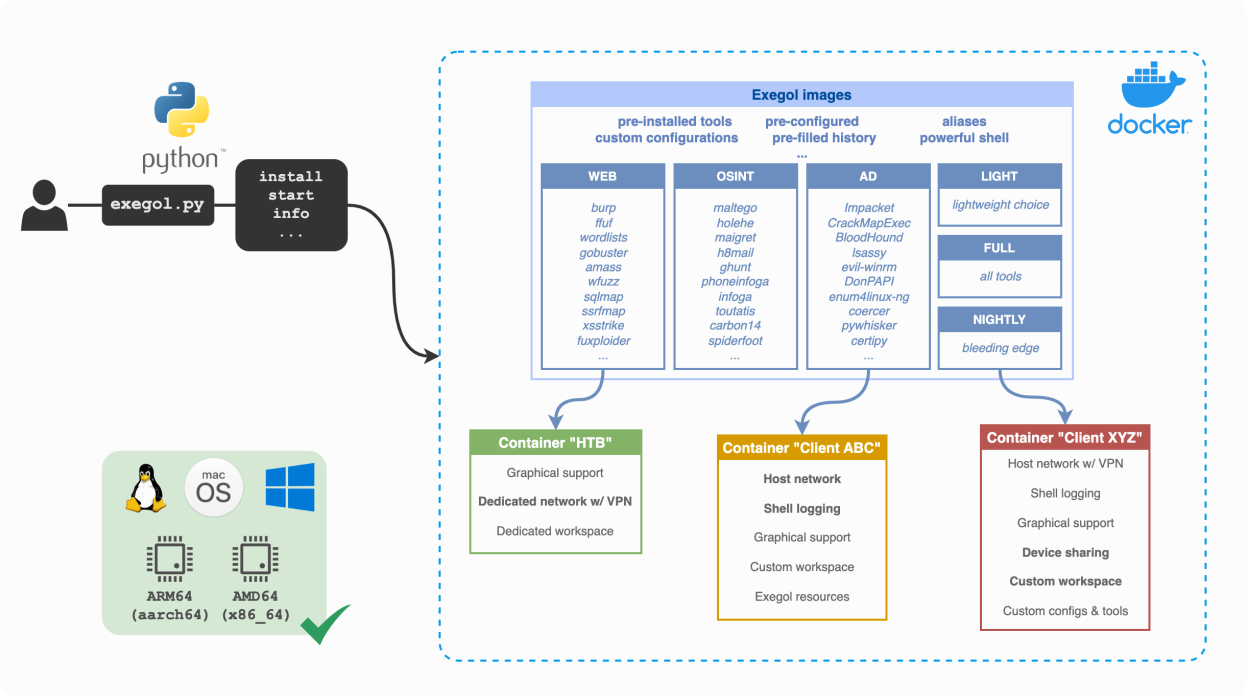
THE EXEGOL PROJECT



Exegol is many things in one. Try it, and you'll stop using your old, unstable and risky environment, no more monolithic system that gets messier, buggier and more at risk with time.

- *Python wrapper*: makes everyone's life easier. This entrypoint to the whole Exegol project handles all docker and git operations so you don't have to. **Now's the time to have a clean environment** with one Docker container per engagement without the effort. Exegol handles multiple images and multiple containers. GUI apps, Wi-Fi, USB accessories, volume mounting and many more features are supported and easier to use than ever.
- *Docker images*: a set of pre-built docker images and dockerfiles that include a neat choice of tools, zsh plugins for power users, pre-filled history ready to use with environment variables, awesome resources, custom configs and many more. Images can either be built locally or pulled from the official Dockerhub registry.
- *Offline resources*: Tired of always having to search github for your favorite privesc enumeration script? Exegol includes a set of resources, shared with all exegol containers and your host, including LinPEAS, WinPEAS,

LinEnum, PrivescCheck, SysinternalsSuite, mimikatz, Rubeus, PowerSploit and many more.



CHAPTER

TWO

GETTING STARTED

Wanna try Exegol and join our great community? You need to *install requirements* first, then proceed to the OS-specific instructions: *Linux* | *macOS* | *Windows*

CHAPTER

THREE

COMMUNITY

Have a bug report or feature request? Either open an issue on the Exegol repo or open a ticket on the Exegol discord (preferred, easier, more flexible).

Wanna chat? Need help? Join us on the Exegol discord!

3.1 Install Exegol

Installing Exegol starts with installing the entrypoint to the whole project: the Python wrapper. Once the wrapper is installed, everything else can be managed from it.

Hint: It is strongly advised to install Exegol on a Linux host, especially when planning on using Exegol for internal penetration tests. This is because Docker Desktop on Windows and macOS lacks a few features, mainly due to how these operating systems run Docker containers within an internal VM that doesn't share the host's network interfaces and USB accessories.

Once the wrapper is installed, the second step in setting up Exegol on a device is to install at least one Exegol image, either with `exegol start` (documentation *here*), or with `exegol install` (documentation *here*). Both actions will guide the user in installing an image if needed.

3.1.1 Requirements

The following elements are required before Exegol can be installed, whatever the host's operating system is:

- git (Linux | macOS | Windows)
- python3 (Linux | macOS | Windows)
- docker (Linux) or Docker Desktop (macOS | Windows)
- at least 100GB of free storage recommended (a minimum of 20GB could be enough, but only for the `light` image).

Additional dependencies may be required depending on the host OS.

Linux

macOS

Windows

No additional dependencies for Linux environments.

Tip: From Linux systems, Docker can be installed quickly and easily with the following command-line:

```
curl -fsSL "https://get.docker.com/" -o get-docker.sh
sh get-docker.sh
```

Warning: By default, `sudo` will be required when running docker, hence needed as well for Exegol. For security reasons, it should stay that way, but it's possible to change that. In order to run exegol from the user environment without `sudo`, the user must have the appropriate rights. You can use the following command to grant them to the current user:

```
# add the sudo group to the user
sudo usermod -aG docker $(id -u -n)

# "reload" the user groups with the newly added docker group
newgrp docker
```

For more information, official Docker documentation shows how to manage docker as a non root user.

Warning: Docker "Rootless mode" is not supported by Exegol as of yet. Please follow the install procedure mentionned above.

To support graphical applications (*display sharing functionality*, e.g. Bloodhound, Wireshark, Burp, etc.), additional dependencies and configuration are required:

Hint: The XQuartz requirement below is now optional if using the (beta) *Graphical Remote Desktop feature* instead of X11 sharing (join our Discord to know more about this beta feature).

- XQuartz must be installed
- The XQuartz config `Allow connections from network clients` must be set to true

- Docker Desktop must be configured with default File Sharing (see screenshot below)

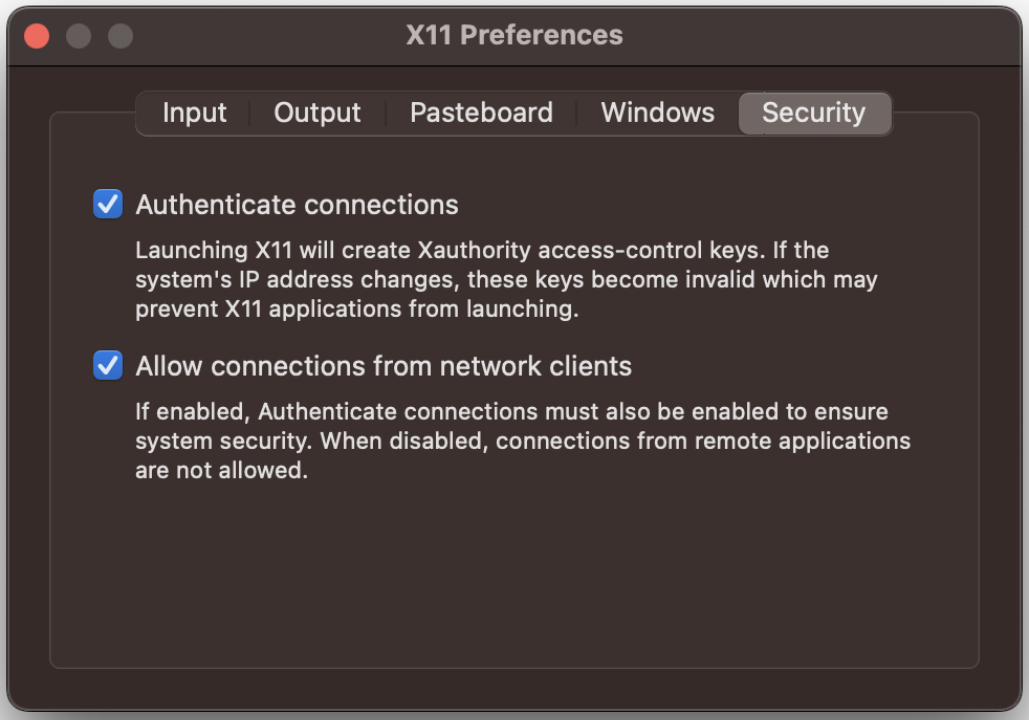


Fig. 1: macOS XQuartz configuration requirement

Tip: OrbStack for **Mac** is supported by Exegol wrapper from `v4.2.0`.

Your exegol installation cannot be stored under `/opt` directory when using OrbStack (due to OrbStack limitations).

This support is still in beta, feel free to open issues on GitHub if you encounter any bugs.

To support graphical applications (*display sharing functionality*, e.g. Bloodhound, Wireshark, Burp, etc.), additional dependencies and configuration are required:

- Windows **10** (up to date), or Windows **11**, is required
- **Docker Desktop** installed on the Windows host
- Docker Desktop must be configured to run on **WSL2** engine (how to)
- WSLg must be installed to support graphical application
- at least one WSL distribution must be **installed** as well (e.g. Debian), with **Docker integration** enabled (see screenshot below)

In a Windows environment, the Exegol wrapper can be installed **either** in a **WSL shell** or directly in your Windows environment with **Powershell**.

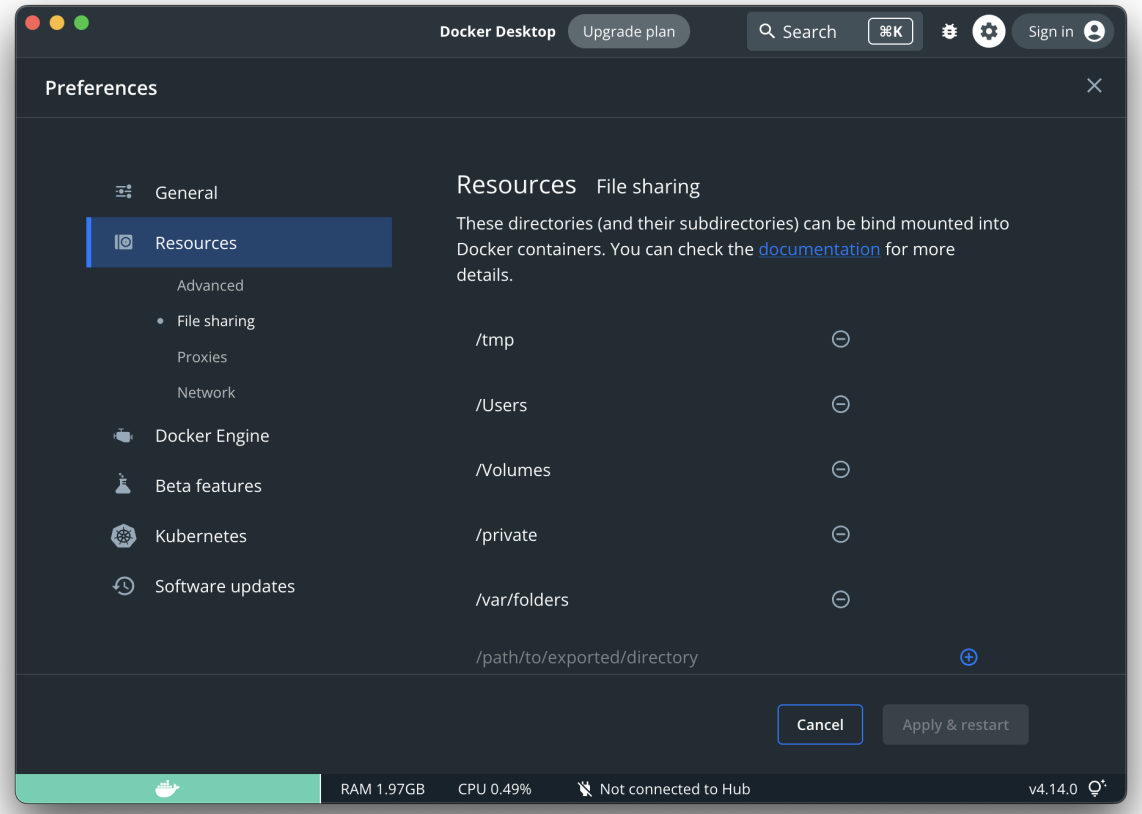


Fig. 2: macOS Docker Desktop resources requirement

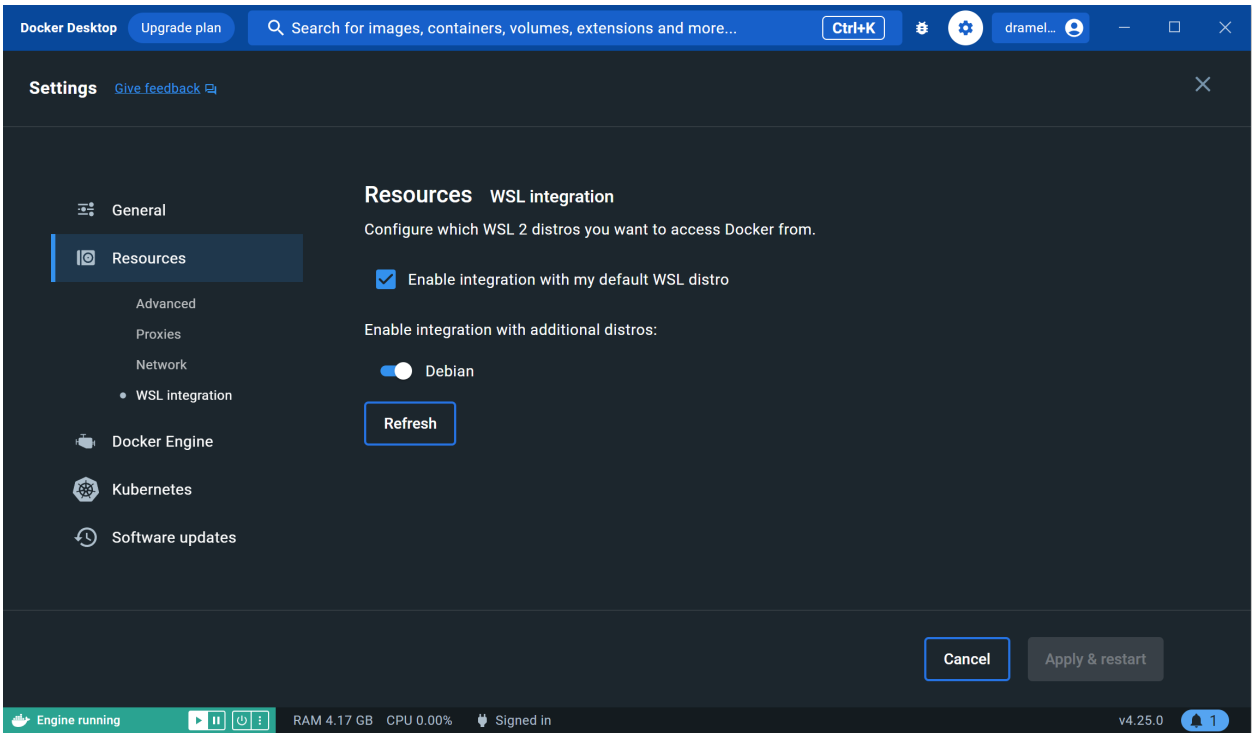


Fig. 3: Windows Docker Desktop WSL integration configuration

Warning: Please note that it is **not** advisable to use Exegol from both environments at the same time, as this could lead to conflicts and Exegol does not officially support this configuration.

3.1.2 Installation

The installation of Exegol on Linux, macOS and Windows are very similar. It can either be installed from pip (easiest, most user-friendly, but with a few missing features) or from sources (easy as well, fully featured).

1. Installation of exegol

Installing with pip

Installing from sources

Installing with pipx

Exegol's wrapper can be installed from pip. While this is the easiest and most user-friendly technique, for more advanced users it is advised to install from sources, as it allows to switch from release to dev branches easily and the auto-update feature is supported.

```
python3 -m pip install exegol
```

Warning: You may want to disable Windows Defender during the installation, as Exegol will download pre-built remote shells (or temporarily exclude `C:\Users\<username>\AppData\Local\Temp`).

> You should also add the folder `C:\Users\<user>\.exegol\exegol-resources` to the exclution list.

Exegol's wrapper can also be installed from sources (with Git). The wrapper then knows how to self-update, and switching from release and development branches is possible and very easy.

```
git clone "https://github.com/ThePorgs/Exegol"
```

Tip: If you want a **light** clone of Exegol (and **never** use the **dev** branch), you can use the following command:

```
git clone --shallow-since="2023/05/08" "https://github.com/ThePorgs/Exegol"
```

If you have access to docker directly as a user, you can install the requirements only for your current user otherwise the requirements must be installed as root to run Exegol with sudo.

With sudo

Directly as user

```
sudo python3 -m pip install --requirement "Exegol/requirements.txt"
```

```
python3 -m pip install --user --requirement "Exegol/requirements.txt"
```

Exegol's wrapper can also be installed with pipx either from sources or PyPI, which allows to install Exegol in a virtual environment of its own.

```
# install pipx if not already installed
python3 -m pip install pipx

# from sources
pipx install git+https://github.com/ThePorgs/Exegol

# packaged from PyPI
pipx install exegol
```

2. Adding Exegol to the `PATH`

Installing with pip

Installing from sources

If your pip installation is correct and functional, you have nothing more to do and you can already use the command `exegol`.

If not, remember that pip installs binaries in a **dedicated** local folder, which then **must** be in the `PATH` environment variable. Try to fix your pip installation: Linux | MacOS | Windows

Linux & MacOS

Windows

Once this is taken care of, the exegol wrapper can then be added to the `PATH` with a symlink for direct access. This allows to call exegol from wherever, instead of to use the absolute path. Exegol can then be used with `exegol <action>` instead of `python3 /path/to/Exegol/exegol.py <action>`.

```
sudo ln -s "$(pwd)/Exegol/exegol.py" "/usr/local/bin/exegol"
```

Once this is taken care of, the exegol wrapper can then can be added as a PowerShell command alias. Exegol can then be used with `exegol <action>` instead of `python3 /path/to/Exegol/exegol.py <action>`.

To create the alias file correctly, open a PowerShell and place yourself in the folder where exegol is located (applicable only for *from source* installations) and run the following commands:

Create *$PROFILE* file if it doesn't exist:

```
if (!(Test-Path -Path $PROFILE)) {
    New-Item -ItemType File -Path $PROFILE -Force
}
```

Create alias for Exegol in *$PROFILE*:

```
echo "Set-Alias -Name exegol -Value '$(pwd)\exegol.py'" >> $PROFILE
```

Warning: To automatically load aliases from the `.ps1` file, PowerShell's `Get-ExecutionPolicy` must be set to `RemoteSigned`.

If the configuration is not correct it can be configured as **administrator** with the following command:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

Tip: If you have installed Python3 manually and Windows opens the **Microsoft store** on the python page as soon as you type `python3.exe`, try this:

It is possible to disable this behavior in the Windows settings: `Apps > Apps & features > App execution aliases` and disable aliases for `python.exe` and `python3.exe`.

3. (Optional) Using Exegol auto-completion

Exegol (wrapper) supports auto-completion in many shell environments but there is a configuration to add (on the host) for this feature to work.

Tip: If you have a source installation, make sure you have installed (or updated) the `requirements.txt` packages before using the completer.

Important: The following configurations must be made in your **host** environment.

Bash

Zsh

Fish

Tcsh

PowerShell

You can enable Exegol auto-completion for your **current user** with your `.bashrc` or you can enable the auto-completion **system-wide** with `bash-completion`.

Via bash-completion

Via .bashrc

To setup the auto-completion system-wide you first need to install `bash-completion` on your system (if not already installed).

```
sudo apt update && sudo apt install bash-completion
```

At this point you should have a `/etc/bash_completion.d/` folder. It's in there that you can add any auto-completion module that you want.

To generate and install the exegol completion configuration you can execute the following command with `register-python-argcomplete`:

```
register-python-argcomplete --no-defaults exegol | sudo tee /etc/bash_completion.d/
↪exegol > /dev/null
```

Add the following command in your `~/.bashrc` config:

```
eval "$(register-python-argcomplete --no-defaults exegol)"
```

Tip: If you have multiple tools using `argcomplete` you can also use the global completion method (need bash >= 4.2).

To activate completions for zsh you need to have `bashcompinit` enabled in zsh:

```
autoload -U bashcompinit
bashcompinit
```

Afterwards you can enable completion by adding the following command in your `~/.zshrc` config:

```
eval "$(register-python-argcomplete --no-defaults exegol)"
```

To activate completions for fish use:

```
register-python-argcomplete --no-defaults --shell fish exegol | source
```

or create new completion file, e.g:

```
register-python-argcomplete --no-defaults --shell fish exegol > ~/.config/fish/
↪completions/exegol.fish
```

To activate completions for tcsh use:

```
eval `register-python-argcomplete --no-defaults --shell tcsh exegol`
```

To activate completions for PowerShell, first generate completion file :

```
register-python-argcomplete --no-defaults --shell powershell exegol > $HOME\Documents\
↪WindowsPowerShell\exegol_completion.psm1
```

Warning: If the command `register-python-argcomplete` is not found, that means that python pip script are not in your PATH. You can ty to fix your pip installation: Linux | MacOS | Windows Or find the direct Python script path, it might be something like: `$HOME\AppData\Roaming\Python\Python311\Scripts\register-python-argcomplete` (`Python311` PATH depends on the version of Python you have installed, it must be updated to match your local setup).

Then import this completion file in *$PROFILE*:

```
echo "Import-Module '$HOME\Documents\WindowsPowerShell\exegol_completion.psm1'" >>
↪$PROFILE
```

Tip: You can have Zsh style completion in PowerShell using this:

```
echo "Set-PSReadlineKeyHandler -Key Tab -Function MenuComplete" >> $PROFILE
```

4. Installation of the first Exegol image

Once the exegol wrapper is installed, you can download your first docker image with the following command:

```
exegol install
```

3.2 Updates

The whole Exegol can be updated through its own wrapper with `exegol update` (documentation *here*).

Hint: Wrappers installed with pip don't support auto-update. The wrapper itself can then can be updated as follows.

```
python3 -m pip install --upgrade exegol
```

3.3 Frequently asked questions

Below are the frequently asked questions regarding either features, the overall project or troubleshooting matters.

- *How to install Exegol on an external drive?*
- *How to add a new tool?*
- *How do I get X11 to work on a non-Linux host?*
- *Can I install docker directly on my WSL2 distro instead of Docker Desktop ?*

3.3.1 What tools are installed in Exegol?

The list of tools is dynamically generated for all Exegol images and available *here*.

3.3.2 Unable to connect to Docker

There are multiple checks to do to make sure Docker works properly.

Docker service

Docker permissions

Docker socket

Symbolic link

The Docker service must installed up and running.

- For Windows users: Docker Desktop for Windows must be up and running.
- For macOS users: Docker Desktop for Mac (or OrbStack) must be up and running.

Make sure the Docker permissions are consistent with the Exegol permissions. For instance, if you need `sudo` rights to use Docker, you'll most likely need `sudo` to run Exegol smoothly. See *the Exegol install guidance*.

The following command can be used to see the docker socket that is used by default: `docker context ls`.

- For OrbStack users (on macOS), the "orb socket" must be used.
- For Docker Desktop users (macOS/Windows), the "Docker desktop socket" must be used.
- For Linux users, the default socket should work.

Switching context can be done with `docker context use <context>`. For instance, switching from a Docker Desktop to OrbStack could be done with `docker context use orbstack`.

The following symbolic link must exist `/var/run/docker.sock` and point to the correct socket. Below is an example of what it should look like.

```
(Host) ~ $ ls -la /var/run/docker.sock
lrwxr-xr-x  1 root  daemon  38 Jul 28 09:02 /var/run/docker.sock -> /Users/someuser/.
↪orbstack/run/docker.sock
```

If the link does not exist, it could be created with the following command `ln -sf /Users/someuser/.orbstack/run/docker.sock /var/run/docker.sock`. This is an example for OrbStack. The command must be adapted to the user's context.

3.3.3 Can I contribute to the project?

Yes, please refer to the *contributors section*.

3.3.4 Can I run Exegol on a macOS?

Yes. And both CPU architectures are supported (Intel X86_64 (AMD64) and Apple Silicon M1/M2 (ARM64).

Tip: We strongly advised macOS users to replace Docker Desktop with OrbStack, allowing host network mode to work for instance, this it's not supported by Docker Desktop for Mac.

3.3.5 Can I use a VPN with Exegol?

Yes. And you have multiple choices.

- **The "YOLO" choice**: at the container creation (i.e. when "starting" a container for the first time), give all permissions to the container so that you're able to run openvpn in it and start the vpn. The command should look like `exegol start <container_name> <image_name> --privileged`.
- **The better choice**: use the `--vpn` option at the container creation: `exegol start <container_name> <image_name> --vpn <myconf.ovpn>`. It's the easiest and more secure choice. See the `start` help *here*).

Warning: Creating a **privileged** container (c.f. the "YOLO" choice) exposes you to higher security risks. This should be avoided.

3.3.6 Can I customize Exegol?

Yes, please refer to the *"my-resources" documentation* that explains how to automatically setup your changes to your Exegol containers. Also, see the *"wrapper's advanced-uses" documentation* to see how to edit Exegol's conf among other things. You could also want to *make your own Exegol image*

3.3.7 Can I make my own Exegol image?

Yes. You will need to create a dockerfile (e.g. `CUSTOM.dockerfile`) at the root of the exegol-images module next to the other dockerfiles (i.e. `/path/to/Exegol/exegol-docker-build/`) containing the instructions you want the build process to follow.

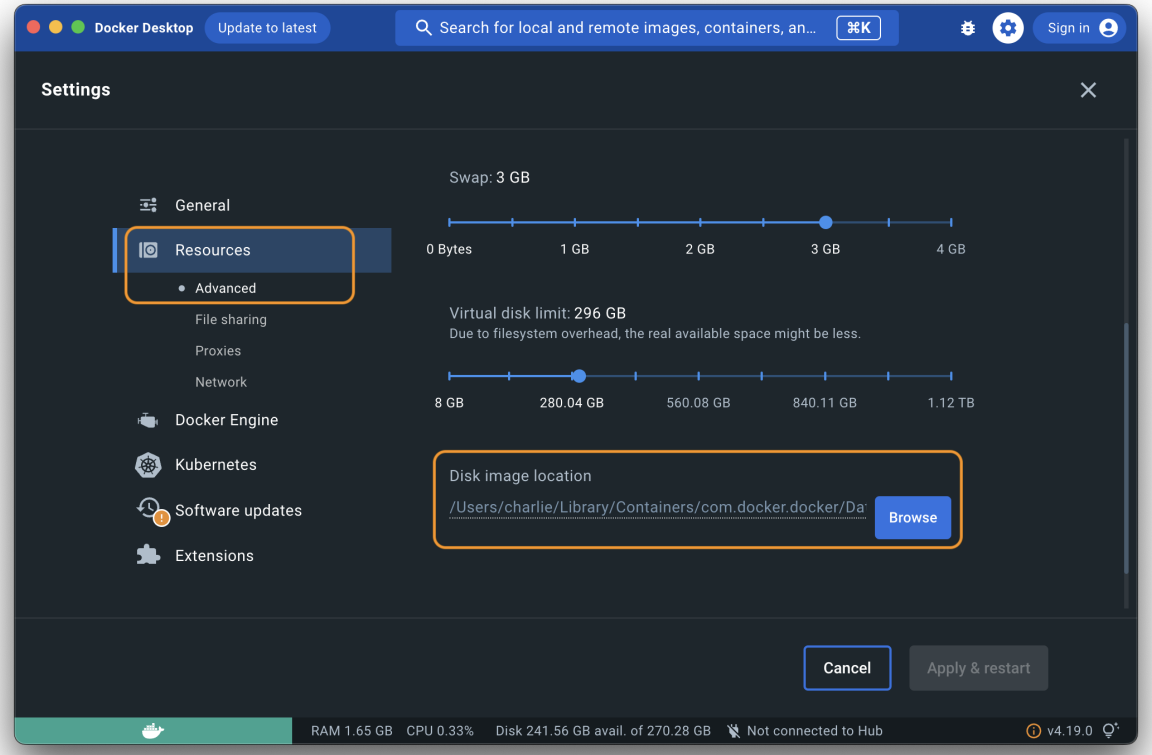
Then, run something like `exegol install "myimagename" "CUSTOM"` to build the image locally. See the `install` documentation: *install action*.

3.3.8 How to install Exegol on an external drive?

Exegol's wrapper is lightweight, but it's Docker images can take up some space, and users may not have enough room in their internal HDD/SSD, hence the question. This usually comes down to "how can I install Docker on an external drive?", and the answer depends on the host.

Tip: Use a fast drive, otherwise Exegol will get real slow.

For macOS and Windows users, this can be configured in the Docker Desktop dashboard (in `Settings > Resources > Advanced > Disk image location`).



3.3.9 How to add a new tool?

"Adding a tool" can mean many things. Depending on that, you'll get a different answer. So let's answer most of them.

If you want to add a tool:

- **in the official Exegol images**: refer to the *contribution guidance*.
- **in your own custom local image**: refer to the *contribution guidance* as well, but instead of creating a Pull Request at the end to offer your contribution, just build the image locally with the wrapper and enjoy your custom local image.
- **in a live container**: that's your container, you can do whatever you whish in it ;)
- **automatically in all containers at their creation**: refer to the *"my-resources" documentation*.

3.3.10 How do I get X11 to work on a non-Linux host?

X11, or X Window System, is a graphical windowing system that provides a framework for creating and managing graphical user interfaces (GUIs) in Unix-like operating systems.

X11 sharing between an Exegol container and a host allows a graphical application running within the container to display its GUI on the host's X11 server. This means you can run graphical applications in Exegol containers and have them appear as if they were running directly on the host machine. It enables the execution of GUI-based applications in isolated containers while interacting with them through the host's graphical interface.

For macOS users, XQuartz is needed. It's listed in the *install requirements*.

Note: Exegol's wrapper automatically starts XQuartz on macOS hosts when needed. But if for some reason it gets manually closed by the users while a container is running, X11 sharing will not work. Restarting the container with `exegol restart <container>` will restart XQuartz automatically if needed.

3.3.11 Can I install docker directly on my WSL2 distro instead of Docker Desktop ?

Yes, it's possible to install docker directly on WSL2 rather than using Docker Desktop, but you'll be restricted to your WSL2 environment and its constraints.

Although Docker Desktop is incomplete, it does offer a few advantages (exegol can be used from powershell / cmd, windows folder sharing with the exegol workstation, etc). We therefore recommend **Docker Desktop as the official support** for Exegol.

We do **not** guarantee wrapper stability with a directly installed WSL docker.

3.4 Tips & tricks

Below are some of the tips and tricks that are good to keep in mind when using Exegol.

3.4.1 Change a container's time

Changing a container's time with `date` requires elevated permissions on the container, and messes up with the host's time. There is however and alternative, using `faketime` (see faketime ubuntu manpage) that allows to change the time of the container easily, without needing particular permissions, without affecting the host. This is especially useful when working with Kerberos targets that are out of sync.

Faketime manipulates the system time for a given child command. For example with `zsh`, a new shell is opened with a spoofed time that will only be spoofed for this extact shell session and commands executed in it.

```
faketime 'YYYY-MM-DD hh:mm:ss' zsh
```

Note: Here is an example of how `faketime` can be used.

When doing Active Directory attacks against Kerberos targets, a clock skew error could be raised such as `KRB_AP_ERR_SKEW`. This means the authenticating machine (operator) and the destination (Key Distribution Center, a.k.a. KDC) are not in sync, clock-wise.

Running any Impacket with the `-debug` flag will print the server time. The operator can then use `faketime` to open a new `zsh` shell with the right time and timezone and conduct the scenario as previously intended.

The following command can be used to print the time in UTC format and compare it with the server time: `date --utc`.

Note: careful with the timezones. If they differ between the operator and the KDC, the delta needs to be taken into account

---

3.4.2 Share files or notes with targets and collaborators

The following tools or commands can be used to pop a temporary file or http server: `updog`, `goshs`, `http-server`, `http-put-server`, `ftp-server`, `smbserver.py`. In order to shares notes during an engagement, `trilium` (https://github.com/zadam/trilium) can be used.

3.4.3 Dynamic history commands

Many commands in the pre-filled history rely on environment variables such as `$DOMAIN`, `$USER`, `$PASSWORD`, etc. Those variables can be set manually or by using the `profile.sh` file in `/opt/tools/Exegol-history/`. The proper lines can be filled and uncommented, and then the shell can be reloaded with `exec zsh` in order to apply the changes. This allows users to easily look for, and use, commands in the history, without changing the values every time.

3.4.4 The best reverse shells

- `shellerator` can be used to generate a reverse shell command dynmically
- on the attacker's side, a reverse shell obtained through a `netcat` tunnel can be improved (see ropnop.com or 0xffsec.com)
- simple alternative way to have an upgrade netcat reverse shell: use `rlwrap <netcat listener command>`
- instead of using `netcat` and "upgrade" the shell manually, `pwncat-cs` (calebstewart/pwncat) can be used to obtain an even better reverse shell experience (especially with UNIX-like targets).

3.4.5 Keyboard shortcuts

- `ctrl+q`: when writing a command, let's say a user misses an information (e.g. IP address). The shortcut can be used to save the half-typed command, look for the value, and then finish the command. The user doesn't have to cancel the command, look for the info, and write the command all over again. This is known as the `push-line` feature (see sgeb.io).
- `ctrl + r`: look for something in the history
- `ctrl + t`: look for a file or directory with a fuzzy finder
- `ctrl + a`: move to the beginning of the line

- `ctrl + e`: move to the end of the line
- `ctrl + ←`: move one word backward
- `ctrl + →`: move one word forward
- `ctrl + l`: clear the screen

3.5 Python Wrapper

The Exegol project regroups many things (docker images, offline resources, custom configurations, aliases, history commands, multi-architecture support and many others). In order to make all the tech involved easy to use, and provide some unique entrypoint to the whole setup, a Python wrapper was created.

The Python wrapper handles all Docker and Git operations, can manage multiple images and containers at once and give the user the best experience possible, suited for beginners as well as advanced people.

The wrapper knows multiple actions.

- Install an image : `exegol install`
- Create/start/enter a container : `exegol start`
- Show info on containers and images : `exegol info`
- Stop a container : `exegol stop`
- Remove a container : `exegol remove`
- Uninstall an image : `exegol uninstall`
- Get help and advanced usage : `exegol --help`
- Help and examples can be obtained for each action directly from the wrapper with the following command: `exegol <action> -h` (action: `install`/`start`/`stop`/etc.).

All actions are documented in the **exegol-wrapper** part of this doc (e.g. *info*, *start*, *version*, ...)

Below is a, non-exhaustive, list of what the wrapper supports:

Note: Exegol uses Docker images and containers. Understanding the difference is essential to understand Exegol.

- **image**: think of it as an immutable template. They cannot be executed as-is and serve as input for containers. It's not possible to open a shell in an image.
- **container**: a container rests upon an image. A container is created for a certain image at a certain time. It's possible to open a shell in a container. Careful though, once a container is created, updating the image it was created upon won't have any impact on the container. In order to enjoy the new things, a new container must be created upon that updated image.

3.5.1 Features

The Exegol wrapper has many features to automatically and transparently manage different configurations to facilitate the use and creation of docker containers.

X11 sharing (GUI)

By default exegol configures the new container and host to allow the execution to the display of graphical window launched from an exegol container. This is achieved through X11 sharing.

For example, if bloodhound is launched in an exegol container, the graphical window (GUI) will be displayed in the user's graphic environment.

This feature can be disabled manually with the option `--disable-X11` of the *start action*.

Desktop

On some systems, it may be difficult to have or share an X11 environment. Some users prefer to have a full graphical desktop environment rather than just graphical applications.

To meet this need, Exegol is able to host a complete graphical environment within its container since version `4.3.0` of the wrapper and `3.1.2` of the images.

This environment can then be made available to others in a variety of ways. The default protocol is currently **HTTP**, but the user can change the configuration to use the **VNC** alternative.

This feature can be enabled manually with the option `--desktop` of the *start action*.

Tip: The default behavior and configuration of the desktop mode can be changed in the *configuration of Exegol*.

Workspace

Exegol always creates within a container a **persistent** workspace (even after deleting the container) and **shared** with the host.

By default a folder will be created on the host and shared with the container. This folder will be created in `~/.exegol/workspaces/` with the name of the exegol container.

Tip: The default location of workspace volumes can be changed in the *configuration of Exegol*.

The user can also create an Exegol container with an **existing custom workspace folder** (with already existing data) regardless of its location in the file system.

See the options `-w WORKSPACE_PATH` and `-cwd` of the *start action* for more details.

Update-fs

The root user is used by default in Exegol containers which poses problems of permissions when accessing the project documents from the host. To remedy this without compromising, a **shared permission system** exists allowing the host user to have read and write access to files created from the container.

This system is **automatically activated** when a **new** default workspace is created.

> **Warning:** When the user uses an existing custom folder as workspace, this system is **disabled** by default! This feature can be **enabled by default** by changing the *configuration of Exegol*.
>
> Its activation is possible manually (see the option `--update-fs` of the *start action*) but it will lead to the **modification** of the folder and its sub-folders **permissions** (as `g+rws`).
>
> If the user does not have the rights to perform such an operation, a **sudo command** will be proposed to the user that he will have to **execute manually** to apply the necessary permissions for the proper functioning of the functionality (as `chgrp + g+rws`).

Tip: When the default configuration of this feature is changed and the update will be **enabled by default**, the option `--update-fs` can still be used to manually **disable** the feature in specific cases.

OpenVPN connection

Exegol supports OpenVPN tunnel configuration to **automatically** establish a VPN tunnel at container **startup**.

Exegol supports certificate authentication (all files should preferably be included in a single ovpn file) but also user/password authentication through an authentication file (to allow non-interactive and transparent authentication).

Tip: A folder can also be used in the case of a **multi-file configuration** (with **relative** paths!) and the configuration file must have the `.ovpn` extension (Only **one** .ovpn file will be loaded by exegol).

See the options `--vpn VPN` and `--vpn-auth VPN_AUTH` of the *start action* for more details.

Tip: In case of problem, to troubleshoot a VPN connection, the log of OpenVPN can be retrieved within the container in the `/var/log/exegol/vpn.log` file

---

Shell logging

Within the framework of a mission, it is necessary to **log all actions** performed during a pentest, red team etc. To meet this need, Exegol has a feature to **automatically record everything** that is displayed (stdout / stderr) but also all entries (stdin).

See the option `--log` of the *start action* to enable the feature.

Hint: When the option is enabled upon **creation** of a new container, all shells created for this container **will be automatically logged**.

If the container was created **without** this option, the shells can still be logged **individually** by adding the option in the **start** command of **each** shell.

The date and time of each command is displayed thanks to the PS1 of `zsh`.

The logs are automatically saved in the `/workspace/logs` folder. Each log file is **automatically compressed** with `gzip` at the end of the session to optimize disk space. The automatic compression of log files can be **disabled** manually with the *start action* `--log-compress` parameter or change the default behavior in the *Exegol configuration file*.

Hint: When the default configuration of the log compression is changed from the config file and the feature will be **disabled by default**, the option `--log-compress` can still be used to manually **enable** the feature in specific cases.

Warning: The logs should **NOT** be consulted from the exegol container but **from the host** to avoid loops and duplication of data in the logs.

There are (since exegol images version `3.0.0`) different methods of shell logging. The shell logging method can be selected manually with the *start action* `--log-method` parameter or by default in the *configuration file of Exegol*.

asciinema (default)

script

The shell logging method **asciinema** is available from exegol images version `3.0.0`. This new mode allows to consult sessions in **video** format taking into account the interactive environment. It is also possible to **manually upload** and **share** recordings, useful for **demonstrations** for example

Here is a quick demonstration:

Tip: Logs in `.gz` format can be replay directly **without unpacking** them with the command: `gunzip -c <filename_shell.asciinema.gz> | asciinema play -`

Hint: To display the whole session **without** the "video" mode, it is possible to use the command: `asciinema cat <filename_shell.asciinema>`

Warning: Major disadvantage of this method, to view the logs from your host, you must **install** `asciinema` on your **host** machine to replay or share your records.

Tip: When you share or play an `asciinema` video, you can **copy** and **paste** any command/text it contains.

script is the "classic" method of session logging, it was also the only option available before version `3.0.0` of exegol images. This method simply records **all** incoming (stdin) and outgoing (stdout/stderr) shell actions in a file.

Tip: Logs in `.gz` format can be viewed directly **without unpacking** them with the `zcat`, `zgrep`, `zdiff` or `zmore` command!

Warning: Shell logging saves **EVERYTHING** including keyboard shortcuts, display refreshes, etc.

Complex graphical environments (such as tmux) can make it difficult to read the logs.

Shared network

By default, containers created by Exegol are in `host` mode which means that the **network interfaces** of the host are **shared** with the container.

This configuration is useful to:

- dynamically open ports and services
- have a low level access on a physical network (some operation might need privileged mode)
- share a unique ip address on the target network
- share a MAC address on the target network (to be considered as a single physical machine)

This mode can be disabled with the option `--disable-shared-network` of the *start action* to create a dedicated and isolated network instead.

Tip: When host network sharing is disabled, ports can be to expose services on the host machine's networks.

Warning: This mode is only available on **Linux** installations! Windows and MacOS installations are subject to the constraints and limitations of Docker Desktop .

You can still use the port *publishing feature* instead.

Shared timezones

For convenience and precision in the date and time of the logs of each command, exegol allows to share the timezone of the host in the container.

This feature is active by default and can be disabled with the option `--disable-shared-timezones` of the *start action*.

Exegol-resources

To save time and have at hand many tools, scripts and other resources, exegol maintains a repository *exegol-resources* contains many updated tools that are available to the host and exegol containers.

This module is not mandatory and can be downloaded later.

Hint: If an antivirus is present on your host, be careful to exclude the destination folder of the `exegol-resources` module before downloading it.

This feature is active and shared by default and can be disabled with the option `--disable-exegol-resources` of the *start action*.

My-resources

The my-resources feature is a space dedicated to the user and shared with all the containers. This space allows to store configurations and to install personal tools.

More details on the functionality of the wrapper *here* and how to take advantage of the customization system *here*.

Volume sharing

For specific needs, the exegol wrapper allows to add additional custom volumes (type bind mounts) when creating an exegol container.

See the option `--volume VOLUMES` of the *action start* for more details.

Port sharing

When the host network is not shared, it is still possible to **publish** specific ports to expose **services** or **port** ranges.

Hint: This configuration is **compatible** even with installations based on Docker Desktop.

This feature allows the user to select:

- a specific network interface (for example 127.0.0.1) or by default all interfaces (0.0.0.0).
- the port to open on the host interface.
- the destination port to be linked in the container.
- the protocol to use, docker supports `TCP`, `UDP` and `SCTP` protocols (default is TCP).

See the option `--port PORTS` of the *start action* for more details.

Env. variables

Exegol can configure custom environment variables defined by the user.

When the environment variables are defined at the first time of the container creation, these variables will be:

- accessible in the container by all processes
- present during the whole lifetime of the container

The environment variables can be defined when opening a shell in an **existing** container and will be available **only** in the user's shell until it is closed.

See the option `--env ENVS` of the *start action* for more details.

Device sharing

For the needs of some applications running on physical hardware (such as proxmark3), exegol can supply the container with one or more physical devices.

See the option `--device DEVICES` of the *start action* for more details.

> **Warning:** This feature is only available on **Linux** installations!
>
> Windows and MacOS installations are subject to the constraints and limitations of Docker Desktop.

Custom hostname

In some environments with OPSEC requirements, it may be necessary to change the hostname of your exegol container.

See the option `--hostname HOSTNAME` of the *start action* for more details.

Comments

To add more context information about an exegol container, it is possible to add **comments** to each Exegol container. The comment of an exegol container can be displayed with the command `exegol info <container_name>`

See the option `--comment COMMENT` of the *start action* for more details.

Capabilities

Exegol supports adding **Linux capabilities** to the Exegol **container** to perform tasks that require **administrative** rights **without** allowing everything.

This feature gives control to the user to configure his container but giving administrator rights remains a dangerous practice and should be used by experienced users only.

For more details on the capabilities supported by docker check here.

> **Warning:** Not all Linux capabilities are allowed by the Exegol wrapper, here is the list of available capabilities:
>
> NET_ADMIN, NET_BROADCAST, SYS_MODULE, SYS_PTRACE, SYS_RAWIO, SYS_ADMIN, LINUX_IMMUTABLE, MAC_ADMIN, SYSLOG
>
> For all other needs, consider the *privileged* mode.

Privileged

For particular needs, it is sometimes necessary to have **privileged rights** to perform certain actions. If Exegol does **not** allow you to have specifically the rights necessary, you can configure your container in privileged mode to get **full administrator rights**.

> **Warning:** This configuration is particularly **dangerous** because it gives the container **full admin control** over the **kernel** of the **host** machine.
>
> Use this option **only** if you know **exactly** what you are doing!!

If the need is specifically identified, consider adding *capabilities* instead!

See the option `--privileged` of the *start action* for more details.

Multi-architecture

Exegol supports `ARM64` architecture (in addition to the classic `AMD64`) since version `4.1.0` of the wrapper and `3.0.0` of the images.

This support allows you to fully use exegol on hardware equipped with an **ARM** processor (such as Mac M1 / M2 but also some Raspberry Pi).

Warning: Exegol only supports **64-bit ARM** architecture! If your ARM processor supports 64-bit, make sure your **OS** is also installed in **64-bit version** to use exegol!

Tip: For experienced users or developers, it is possible to explicitly modify the architecture used by the Exegol wrapper with the *general option* `--arch ARCH`.

But be **careful**, the modification of this parameter can lead to **malfunctions**!

Local image building

The wrapper allows users to locally build their images from the `exegol-images` sources.

More information in the *advanced uses* section.

Remote image pulling

To save time, pre-built images are available for download from DockerHub. These images can be downloaded and installed / updated from the exegol wrapper with the *install* and *update* actions.

Command execution

The Exegol wrapper does not only allow the opening of interactive shells, it is also possible to execute **single commands** in several ways.

Tip: To see the execution logs of the command, the user must add the parameter `-v`.

The details of this functionality are detailed in the *exec* action.

Daemon execution

One of the execution modes can be in the **background** like a daemon service. In this way the wrapper executes the **user's command**, for example an application such as bloodhound. The wrapper **launches** the task in an exegol container and **finishes immediately** without occupying the user's terminal, leaving the application **open**.

See the option `--background` of the *exec action* for more details.

Temporary containers

Another feature of the *exec* action is the execution in a **temporary** container.

In this mode, a **temporary** container will be created and **dedicated** to the execution of the command specified by the user.

This mode can be useful to run a given command with the most **up-to-date** image already installed on the host, for any **test** or for special **privacy** needs.

See the option `--tmp` of the *exec action* for more details.

3.6 Docker images

The Docker images are the heart of the Exegol project. A neat choice of tools, configurations, aliases, history commands, and various customizations are prepared in multiple images adapted for multiple uses: web hacking, Active Directory, OSINT (Open Source INTelligence), etc.

All images are available on the official Dockerhub registry. This allows to offer pre-built, compressed images, so that users don't have to build their own image, but users that choose to do so can. Pulling pre-built images, or building one, can be done with `exegol install` (documentation *here*).

Image name	Description
full	Includes all the tools supported by Exegol (warning: this is the heaviest image)
ad	Includes tools for Active Directory / internal pentesting only.
web	Includes tools for Web pentesting only.
light	Includes the lightest and most used tools for various purposes.
osint	Includes tools for OSINT.
nightly	**(for developers and advanced users)** contains the latest updates. This image can be **unstable**!

Note: Exegol uses Docker images and containers. Understanding the difference is essential to understand Exegol.

- **image**: think of it as an immutable template. They cannot be executed as-is and serve as input for containers. It's not possible to open a shell in an image.
- **container**: a container rests upon an image. A container is created for a certain image at a certain time. It's possible to open a shell in a container. Careful though, once a container is created, updating the image it was created upon won't have any impact on the container. In order to enjoy the new things, a new container must be created upon that updated image.

3.7 Offline resources

Exegol's "offline resources" are a neat choice of standalone tools and scripts that are often used during penetration tests, CTFs and red-teams, like LinPEAS, WinPEAS, LinEnum, PrivescCheck, SysinternalsSuite, mimikatz, Rubeus, PowerSploit and many more. Exegol users don't have to download those resources again every time they're needed anymore. Everything is managed by the wrapper and they are shared with every container (at `/opt/resources`).

3.8 `install` action

This action can be used to install an Exegol image. At least one Exegol image is required to create and start a container and enjoy Exegol.

When this action is used, the image can either be:

- **downloaded** (i.e. "pulled" in Docker terms) from the official Dockerhub registry. In this case, a compressed and pre-built image is downloaded in the form of layers, and then uncompressed.
- **built** locally by following the instructions of a Dockerfile offered on the Exegol-images GitHub repo. Here again, no need to download the dockerfile manually, all of them are already at `/path/to/Exegol/exegol-docker-build/`.

Hint: The `install` action can be used without any particular argument or option. the wrapper will then enter in an interactive TUI (Text-based User Interface) mode where the user will be asked to choose what image to install or build.

```
exegol install
```

3.8.1 Options

Option	Description
`IMAGE`	Optional positional argument to indicate the image to install (if downloading), or the name of the image to build (if building locally).
`BUILD_PROFI`	Optional positional argument to indicate the source profile to use if building locally.
`--build-log`	Write logs to the path specified if building locally.
`--build-pat`	Custom path to the dockerfiles and sources.

3.8.2 Command examples

```
#Install or build interactively an exegol image
exegol install

#Install or update the full image
exegol install full

#Build interactively a local image named myimage
exegol install myimage
```

(continues on next page)

(continued from previous page)

```
#Build the myimage image based on the full profile and log the operation
exegol install myimage full --build-log "/tmp/build.log"
```

3.9 `start` action

This action can be used to start a container. At least one Exegol image is required to create and start a container and enjoy Exegol. Installing an image can be done with `exegol install` (documentation *here*).

When this action is used, the following process is applied:

- if no Exegol image is installed, the user is asked to specify which one to install of build, and the process continues
- then, if the container to start doesn't already exist, it is created based on an Exegol image and a few settings to specify, and the process continues
- then, the container is started and a shell is opened

Hint: The `start` action can be used without any additional argument or option. the wrapper will then enter in an interactive TUI (Text-based User Interface) mode where the user will be asked to choose a few settings.

```
exegol start
```

3.9.1 Options

A single option exist to target an Exegol container. If this container exists, it will be started if it is not already the case and a shell will be spawned to offer an interactive console to the user

Option	Description
`CONTAINER`	Tag used to target an Exegol container

Many options exist to customize the creation of exegol container.

Tip: The default options of some commands can be changed in the *exegol configuration file*.

Global options

Option	Description
IMAGE	Tag of the exegol image to use to create a new exegol container
-w WORKSPACE_PATH, --workspace WORKSPACE_PATH	The specified host folder will be linked to the /workspace folder in the container.
-cwd, --cwd-mount	This option is a shortcut to set the /workspace folder to the user's current working directory (pwd).
-fs, --update-fs	Modifies the permissions of folders and sub-folders shared in your workspace to access the files created within the container using your host user account. (default: Disabled)
-V VOLUMES, --volume VOLUMES	Share a new volume between host and exegol (format: –volume /path/on/host/:/path/in/container/[:ro|rw]).
-p PORTS, --port PORTS	Share a network port between host and exegol (format: –port [<host_ipv4>:]<host_port>[:<container_port>][:<protocol>]. This configuration will disable the shared network with the host.
--hostname HOSTNAME	Set a custom hostname to the exegol container (default: exegol-<name>)
--cap CAPABILITIES	**(dangerous)** Capabilities allow to add specific privileges to the container (e.g. need to mount volumes, perform low-level operations on the network, etc).
--privileged	**(dangerous)** give extended privileges at the container creation (e.g. needed to mount things, to use wifi or bluetooth)
-d DEVICES, --device DEVICES	Add host device(s) at the container creation (example: -d /dev/ttyACM0 -d /dev/bus/usb/).
--disable-X11	Disable X11 sharing to run GUI-based applications. (default: Enabled)
--disable-my-resource	Disable the mount of the shared resources (/opt/my-resources) from the host (/home/dramelac/.exegol/my-resources) (default: Enabled)
--disable-exegol-reso	Disable the mount of the exegol resources (/opt/resources) from the host (/home/dramelac/Documents/tools/Exegol/exegol-resources) (default: Enabled)
--disable-shared-netw	Disable the sharing of the host's network interfaces with exegol (default: Enabled)
--disable-shared-time	Disable the sharing of the host's time and timezone configuration with exegol (default: Enabled)

Virtual desktop

In addition to the X11 sharing functionality, Exegol can generate its own graphical environment and make it available to the user in a variety of ways. By default, a web interface gives users access to their own containerized graphical desktop.

Option	Description
--desktop	Enable or disable the Exegol desktop feature (default: Disabled)
--desktop-con	Configure your exegol desktop (vnc or http) and its exposure (format: proto[:ip[:port]]) (default: http:127.0.0.1:<random>)

VPN

An additional feature of Exegol is the VPN tunnel option (OpenVPN). Just provide an ovpn configuration to exegol and the container will take care of starting the tunnel at each startup.

Option	Description
`--vpn VPN`	Setup an OpenVPN connection at the container creation (example: –vpn /home/user/vpn/conf.ovpn)
`--vpn-auth VPN_AUTH`	Enter the credentials with a file (first line: username, second line: password) to establish the VPN connection automatically (example: –vpn-auth /home/user/vpn/auth.txt)

Warning: All the options seen previously are taken into account **only** for the creation of a **new container**. It is **not possible** to modify the configuration of an existing container. These options will be **ignored** if a container with the same name already exists.

Shell logging

One of the functions of exegol very useful in a professional context is the shell logging. This feature allows the user to record **everything** that happens in the exegol container (commands typed and responses).

Option	Description
`-l`, `--log`	Enable shell logging (commands and outputs) on exegol to /workspace/logs/ (default: Disabled)
`--log-method`	Select a shell logging method used to record the session (default: `asciinema`)
`--log-compress`	Enable or disable the automatic compression of log files at the end of the session (default: Enabled)

Tip: When the `-l`/`--log` option is enabled during the **creation** of a **new** container, all future shells will be **automatically logged** for this container.

Session specific

The options specific to the start of the interactive session

Option	Description
`-e ENVS`, `--env ENVS`	And an environment variable on Exegol (format: –env KEY=value). The variables configured during the creation of the container will be persistent in all shells. If the container already exists, the variable will be present only in the current shell.
`-s SHELL`, `--shell SHELL`	Select a shell environment to launch at startup (default: zsh)

Tip: The environment variables configured with `--env ENVS` during the **creation** of a **new** container will be available to **all** processes of the container during the **entire life cycle** of the container.

3.9.2 Command examples

```
# Start interactively a container
exegol start

# Create a demo container using full image
exegol start demo full

# Spawn a shell from demo container
exegol start demo

# Create a container test with a custom shared workspace
exegol start test full -w "./project/pentest/"

# Create a container test sharing the current working directory
exegol start test full -cwd

# Create a container htb with a VPN
exegol start htb full --vpn "~/vpn/lab_Dramelac.ovpn"

# Create a container app with custom volume
exegol start app full -V "/var/app/:/app/"

# Get a shell based on tmux
exegol start --shell tmux

# Share a specific hardware device (like Proxmark)
exegol start -d "/dev/ttyACM0"

# Share every USB device connected to the host
exegol start -d "/dev/bus/usb/"
```

3.10 `info` action

The `info` action aims at displaying all the information specific to the Exegol project on the current system. This action can also be used by targeting a specific container to display its configuration in detail.

Depending on the verbosity level specified in the command-line, the information displayed will be more or less detailed accordingly.

Standard (default)

Verbose

Advanced

Debug

```
exegol info
```

- List of available Exegol Images
  - Name of the image

  - Size of each image (disk space if the image is installed, otherwise its compressed size to download for installation)
  - Status of each image
    * Not installed (Image available for download from dockerhub)
    * Up to date (The latest version of the image is installed and ready to be used)
    * Update available (A new version is available for download on dockerhub)
    * Outdated (Old version of an image that has been updated since)
    * Local image (Locally built image)
    * Discontinued (if your image is no longer available on dockerhub)
- List of Exegol Containers
  - Name of each container
  - Container status (Stopped or running)
  - Image name (Exegol image used as a base to create the container)
  - Configurations (Display of non-default configurations)

```
exegol info -v
```

In the verbose mode, the following additional elements are displayed. Everything from the lower verbosity level is still displayed.

- Enumerate every user configuration (see details *here*)
- In the list of available Exegol Images
  - Image ID
  - Build date
  - Image architecture (AMD64 / ARM64)
- In the list of Exegol Containers
  - Container id
  - Full configuration
  - List of non-technical volumes
  - List of Devices
  - List of Ports (Applicable if network sharing with host is not enabled)
  - List of custom environment variables

```
exegol info -vv
```

In the advanced mode, the following additional elements are displayed. Everything from the lower verbosity levels is still displayed.

- Enumerate the settings from the user configuration at `~/.exegol/config.yml` (see details *here*)
- List the different exegol modules
  - Modules name
  - Their update status

- Their git branch (if applicable)

```
exegol info -vvv
```

In the debug mode, everything from the lower verbosity levels is still displayed, as well as logs from internal methods and functions. Those logs can be useful for maintainers and developers in case of bug, or for making sure everything works properly.

3.10.1 Options

The info action does not have many parameters, its use is relatively simple. This action can either be used to gather general information (available images, containers, user configs, etc.), or gather information about a specific container and display its configuration.

Option	Description
`CONTAINER`	Optional positional argument to indicate the container tag of which to display the configuration.

Global options can still be used, like for any action.

Option	Description
`-h, --help`	Show the help message of any action
`-v, --verbose`	Verbosity level (-v for verbose, -vv for advanced, -vvv for debug)
`-q, --quiet`	Show no information at all
`-k, --insecure`	Allow insecure server connections for web requests, e.g. when fetching info from Docker-Hub (default: Secure)
`--offline`	Run exegol in offline mode, no request will be made on internet (default: Disable)
`--arch {arm64, amd64}`	Overwrite default image architecture (default: host's arch)

3.10.2 Command examples

```
# Print containers and images essentials information:
exegol info

# Print the detailed configuration of the "demo" container:
exegol info demo

# Print verbose information:
exegol info -v

# Print advanced information:
exegol info -vv

# Print debug information:
exegol info -vvv
```

3.11 exec action

This action allows to run a single command in a single container instead of loading a full interactive shell.

When this action is used it is possible to execute a command either in:

- a **temporary** container created especially to execute the command, and **automatically deleted** at the end of the execution: the name of an exegol **image** must be provided from which a temporary container will be created
- a standard Exegol container (already existing, or not): the name of an exegol **container** must then be provided. This container will be created in interactive mode if it does not already exist

The executed command can be executed either:

- in the **background** mode (i.e. like a daemon): exegol **terminates immediately** after the command is launched and does not wait for its execution to be completed. No process is left hanging (useful when running GUI apps for instance).
- in **standard mode**: exegol will **wait** for the end of the process to stop the container (and delete it if

Tip: In standard execution mode, it is possible to ask exegol to display the command **output** (stdout/stderr) in the terminal by adding `-v` parameter.

3.11.1 Options

Since the exec action can also create containers, it shares the same parameters as the *start action*.

There are also additional parameters, unique to the `exec` action:

Option	Description
`CONTAINER` or `IMAGE`	This option indicates the **container** name to use to execute the command. If the `--tmp` parameter is used, this name will be used to target an **image**.
`COMMAND`	Single command to execute in the container.
`-b`, `--background`	Executes the command in background as a daemon.
`--tmp`	Creates a dedicated and temporary container to execute the command.

3.11.2 Command examples

```
# Execute the command bloodhound in the container demo:
exegol exec demo bloodhound

# Execute the command 'nmap -h' with console output in the container demo:
exegol exec -v demo 'nmap -h'

# Execute a command in background within the demo container:
exegol exec -b demo bloodhound

# Execute the command bloodhound in a temporary container based on the full image:
exegol exec --tmp full bloodhound
```

(continues on next page)

(continued from previous page)

```
# Execute a command in background with a temporary container:
exegol exec -b --tmp full bloodhound

# Execute Wireshark in background, in a privileged temporary container:
exegol exec --background --tmp --privileged "nightly" "wireshark"

# Execute the command wireshark with network admin privileged:
exegol exec -b --tmp --cap NET_ADMIN full wireshark
```

3.12 `update` action

3.12.1 The update process

The exegol wrapper has an `update` action dedicated to updating the different modules (wrapper, resources, etc.) of the project as well as the (docker) Exegol images.

Modules updates

This action make sure the local copies of the following repositories are up to date:

- Exegol (wrapper code). If the wrapper has been installed with Pip, it will not be able to self-update, updating the package through Pip will be required (e.g. `python3 -m pip install --upgrade exegol`).
- Exegol-images (docker building files)
- Exegol-resources (offline resources, *exegol-resources*). This module is optional, and users can choose to install/update it at any time.

Tip: When running `exegol update -v`, the user will be able to choose from what branch them module should be synchronized with, allowing to switch easily between release and dev versions.

Images updates

Once the local code base is updated, the wrapper compares the installed Exegol images with those offered on the Dockerhub registry. If no parameters have been provided at command-line, an interactive selection will be possible to choose the images to update (if updates are available).

Hint: Older versions of images will be automatically deleted if they are no longer used by any container and if a newer version of the same image is installed. This automatic deletion behavior is a default configuration that can be modified in the *configuration file* if needed, but it's advised not to change it as disk space is not unlimited and Exegol image can take up to 30GB.

3.12.2 Options

The options of the `update` action are the following.

Option	Description
IMAGE	This option specifies what image to update.
`--skip-git`	Skip *modules updates*.
`--skip-images`	Skip *images updates*.

3.12.3 Command examples

```
# Update interactively an exegol image:
exegol update

# Update the full image:
exegol update full

# Update the full image without updating exegol modules:
exegol update --skip-git full

# Update exegol modules and have the option to change branch without updating docker␣
↪image:
exegol update -v --skip-images
```

3.13 `stop` action

The purpose of the `stop` action is to stop one or more Exegol containers.

If these containers have a VPN configuration, shutting down the container will cause the VPN tunnel to be disconnected.

3.13.1 Options

The options of the stop action are limited to selecting the container(s) to be stopped.

Option	Description
`CONTAINER`	Tag used to target one or more Exegol containers

3.13.2 Command examples

```
# Stop interactively one or more containers:
exegol stop

# Stop the "demo" container:
exegol stop "demo"
```

(continues on next page)

(continued from previous page)

```
# Stop the "demo", "test" and "dev" container:
exegol stop "demo" "test" "dev"
```

3.14 `restart` action

The purpose of the `restart` action is to stop and directly restart an Exegol container.

If these containers have a `privileged` or `device` configuration, restarting the container will refresh the available devices inside the container.

> **Warning:** Restarting a container will reset the `hosts`, `resolv.conf` files (and probably more).

3.14.1 Options

The options of the restart action are limited to selecting the container to be restarted and some starting options that can be also found on the *start action*.

Option	Description
`CONTAINER`	Tag used to target the Exegol containers to restart

3.14.2 Command examples

```
# Restart interactively one container:
exegol restart

# Restart the "demo" container:
exegol restart "demo"
```

3.15 `remove` action

The purpose of the `remove` action is to remove one or more Exegol container.

If the deleted container was using an outdated image, the wrapper will (by default) try to delete that outdated image automatically (unless this default behavior is changed in the *exegol configuration file*, which is not advised since disk space is not limited and Exegol images can take up to 30GB).

When deleting the container, the wrapper will check if the content of the `/workspace` volume is empty. If the workspace is **empty**, exegol will **automatically delete** the folder on the host, otherwise it will **explicitly ask the user** if the workspace content should be **deleted** or not.

3.15.1 Options

The options of the `remove` action are limited to selecting the container(s) to be removed and forcing the removal without asking the user for interactive confirmation.

Option	Description
`CONTAINER`	Tag used to target one or more Exegol containers
`-F`, `--force`	Remove container without interactive user confirmation (confirmation will still be required for removing non-empty workspaces).

3.15.2 Command examples

```
# Remove interactively one or more containers:
exegol remove

# Remove the "demo" container:
exegol remove "demo"

# Remove the "demo", "test" and "dev" container without asking for user confirmation:
exegol remove -F "demo" "test" "dev"
```

3.16 `uninstall` action

The purpose of the `uninstall` action is to remove one or more Exegol images.

Warning: The wrapper will try to delete the selected exegol images but this can only work if the selected images are **not used by any container** anymore. A container based on an image that doesn't exist anymore cannot run.

3.16.1 Options

The options of the `uninstall` action are limited to selecting the image(s) to be removed, and forcing the removal without asking the user for interactive confirmation.

Option	Description
`IMAGE`	Tag used to target one or more Exegol images
`-F`, `--force`	Remove image without interactive user confirmation.

3.16.2 Command examples

```
# Remove interactively one or more containers:
exegol uninstall

# Remove the "full" container:
exegol uninstall "full"

# Remove the "full", "ad" and "web" container without asking for user confirmation:
exegol uninstall -F "full" "ad" "web"
```

3.17 `version` action

The `version` action is mostly used for debugging purposes, it only displays information about the Exegol setup on the system.

In debug mode (`-vvv`), it also displays information about the system and wrapper installation context.

3.17.1 Options

The options available for the version action are the global options that affect the behavior of all exegol actions.

Option	Description
`-h, --help`	Show the help message of any action
`-v, --verbose`	Verbosity level (-v for verbose, -vv for advanced, -vvv for debug)
`-q, --quiet`	Show no information at all
`-k, --insecure`	Allow insecure server connections for web requests, e.g. when fetching info from Docker-Hub (default: Secure)
`--offline`	Run exegol in offline mode, no request will be made on internet (default: Disable)
`--arch {arm64, amd64}`	Overwrite default image architecture (default: host's arch)

3.17.2 Command examples

```
# Show version information
exegol version

# Show version and system information
exegol version -vvv
```

3.18 Advanced uses

3.18.1 Exegol home directory

The `~/.exegol` folder exists in the user's home folder to centralize *"exegol resources"*, *"my-resources"*, volumes and also the configuration file.

- The **configuration file** (YAML) is located at `~/.exegol/config.yml` and is generated by the wrapper during the first execution, with the default configurations.
- By default, every exegol container has a **workspace volume**. If the path of this volume is not specified by the user (*see start parameters*), a folder with the name of the container will be created in the *"private workspace"* folder. By default, this folder is located at `~/.exegol/workspaces/`.

3.18.2 Exegol configuration

The Exegol wrapper is configured with many default settings. Most of them can be modified with a simple argument. For productivity purposes, setting a different default behavior once and not have to add the same options everytime is interesting. For this exact purpose, a configuration file exists that allows users to persistently change the behavior and operations to be performed by default.

The user configuration currently in place can be viewed with the command: `exegol info -v`. More information on the *info page*.

Within the `~/.exegol/config.yml` file, several settings can be configured to customize the Exegol experience, all distributed in multiple sections below.

Volumes

Config

The volume section allows to change the default path for various volumes.

> **Warning:** Volume path can be changed at any time but already existing containers will not be affected by the update and will keep the original paths they were created with.

- `my_resources_path`: the "my-resources" volume is a storage space dedicated to the user to customize his environment and tools. This volume is, by default, shared across all exegol containers. See *details about it*.
- `exegol_resources_path`: exegol-resources are data and static tools downloaded in addition to docker images. These tools are complementary and are accessible directly from the host. See *details*.
- `private_workspace_path`: when containers do not have an explicitly declared workspace at their creation (i.e. with `--cwd-mount`, or `--workspace`), a dedicated folder will be created at this location to share the workspace with the host but also to save the data after deleting the container.

The config section allows you to modify the default behavior of the Exegol wrapper.

- `auto_check_update`: enables automatic check for wrapper update. (Default: `True`)
- `auto_remove_image`: automatically remove outdated image when they are no longer used. (Default: `True`)
- `auto_update_workspace_fs`: automatically modifies the permissions of folders and sub-folders in your workspace by default to enable file sharing between the container with your host user. (Default: `False`)
- `default_start_shell`: default shell command to start. (Default: `zsh`)

Shell logging

Desktop

Change the configuration of the shell logging functionality.

- `logging_method`: Choice of the method used to record the sessions, `script` or `asciinema`. (Default: `asciinema`)
- `enable_log_compression`: Enable automatic compression of log files (with gzip). (Default: `True`)

Change the configuration of the virtual Desktop feature.

- `enabled_by_default`: Enables or not the desktop mode by default. If this attribute is set to True, then using the CLI `--desktop` option will be inverted and will **DISABLE** the feature (Default: `False`)
- `default_protocol`: Default desktop protocol,can be `http`, or `vnc` depending on your wrapper / image version. (Default: `http`)
- `localhost_by_default`: Desktop service is exposed on localhost by default. If set to true, services will be exposed on `localhost` (127.0.0.1) otherwise it will be exposed on `0.0.0.0`. This setting can be overwritten with *–desktop-config*. (Default: `True`)

3.18.3 My-resources

"My-resources" is a major feature allowing Exegol users to have a volume, shared with all Exegol containers, that can centralize their own resources and configurations. It allows users to enjoy their own tools that are not available in Exegol but also to customize their Exegol setup. More information on the dedicated documentation page *My-Resources*.

This volume is accessible from the host at `~/.exegol/my-resources/` and from the containers (if the feature was left enabled at the container creation) at `/opt/my-resources`.

To facilitate its use, a read/write access system **shared** (between the host user and the container root user) has been implemented.

Hint: To allow this permissions sharing, the "my-resources" folder (and all subdirectories) must have the Set-GID permission bit set. This is done automatically by the wrapper if the current user has sufficient rights. Otherwise, the wrapper will display a sudo command to be executed manually to update the relevant permissions.

The host path of this volume can be changed from the configuration file `~/.exegol/config.yml`.

Warning:

- Be careful **not** to use a folder with **existing data**, in which case their permissions will be automatically modified to enable access sharing.
- This change will not be applied to already existing exegol containers.

3.18.4 Local builds

When installing Exegol, while downloading the pre-built and compressed Docker images from Dockerhub is advised, users can build their own images locally. The wrapper has a **local build feature** to create and manage local exegol images.

The `exegol install` command can be used for that purpose. The user must specify an image name that does **not** match one of the remote images available from dockerhub. The wrapper will suggest to build a local image with this name. If the user chooses to build an image locally, he will then have to choose a **build profile** among those available. The build profile is merely the dockerfile to follow during the build process. An arbitrary dockerfile can be added in `/path/to/Exegol/exegol-docker-build/name.dockerfile`.

Tip:

- the `-v` parameter can be added to have more details about the build process.
- the detailed logs of the docker build process can also be saved in a file with the `--build-log` parameter.

3.19 Tools list

Click *here* to see the lists of tools featured in the latest nightly images.

Error: [August 5th, 2023] - The lists descriptions were mostly AI-generated for this first version of the tools list in order to get the list shipped quickly. Many descriptions are wrong and will be fixed very soon.

Hint: The lists featured here are automatically generated. Exegol features CI/CD pipelines that build the images. At build, most tools are tested. If at least one test fails, the image doesn't get published.

Image tag	Version	Arch	Build date	Tools list
nightly	9c8eac	amd64	2024-04-17T08:22:37Z	`nightly_9c8eadb7_amd64.csv`
nightly	9c8eac	arm64	2024-04-17T08:20:32Z	`nightly_9c8eadb7_arm64.csv`

Image tag	Version	Arch	Build date	Tools list
full	3.1.3	arm64	2024-04-14T14:16:19Z	`full_3.1.3_arm64.csv`
full	3.1.3	amd64	2024-04-14T13:29:00Z	`full_3.1.3_amd64.csv`
osint	3.1.3	arm64	2024-04-14T11:55:04Z	`osint_3.1.3_arm64.csv`
web	3.1.3	arm64	2024-04-14T11:52:28Z	`web_3.1.3_arm64.csv`
ad	3.1.3	arm64	2024-04-14T11:47:33Z	`ad_3.1.3_arm64.csv`
osint	3.1.3	amd64	2024-04-14T10:46:46Z	`osint_3.1.3_amd64.csv`
web	3.1.3	amd64	2024-04-14T10:43:47Z	`web_3.1.3_amd64.csv`
ad	3.1.3	amd64	2024-04-14T10:34:36Z	`ad_3.1.3_amd64.csv`

continues on next page

Table 1 – continued from previous page

Image tag	Version	Arch	Build date	Tools list
light	3.1.3	amd64	2024-04-14T08:47:32Z	light_3.1.3_amd64.csv
light	3.1.3	arm64	2024-04-14T08:38:48Z	light_3.1.3_arm64.csv
full	3.1.2	arm64	2023-12-22T12:20:18Z	full_3.1.2_arm64.csv
full	3.1.2	amd64	2023-12-22T12:04:26Z	full_3.1.2_amd64.csv
ad	3.1.2	arm64	2023-12-22T12:03:46Z	ad_3.1.2_arm64.csv
ad	3.1.2	amd64	2023-12-22T11:41:22Z	ad_3.1.2_amd64.csv
web	3.1.2	arm64	2023-12-22T12:08:45Z	web_3.1.2_arm64.csv
web	3.1.2	amd64	2023-12-22T11:46:29Z	web_3.1.2_amd64.csv
light	3.1.2	arm64	2023-12-22T00:48:17Z	light_3.1.2_arm64.csv
light	3.1.2	amd64	2023-12-22T00:08:53Z	light_3.1.2_amd64.csv
osint	3.1.2	arm64	2023-12-22T00:43:53Z	osint_3.1.2_arm64.csv
osint	3.1.2	amd64	2023-12-22T00:02:50Z	osint_3.1.2_amd64.csv
full	3.1.1	amd64	2023-08-18T01:36:37Z	full_3.1.1_amd64.csv
full	3.1.1	arm64	2023-08-18T01:36:23Z	full_3.1.1_arm64.csv
ad	3.1.1	amd64	2023-08-18T02:59:03Z	ad_3.1.1_amd64.csv
ad	3.1.1	arm64	2023-08-18T02:58:49Z	ad_3.1.1_arm64.csv
web	3.1.1	arm64	2023-08-18T01:08:44Z	web_3.1.1_arm64.csv
web	3.1.1	amd64	2023-08-18T01:35:14Z	web_3.1.1_amd64.csv
osint	3.1.1	arm64	2023-08-18T01:04:50Z	osint_3.1.1_arm64.csv
osint	3.1.1	amd64	2023-08-18T01:34:47Z	osint_3.1.1_amd64.csv
light	3.1.1	arm64	2023-08-18T01:05:12Z	light_3.1.1_arm64.csv
light	3.1.1	amd64	2023-08-18T01:35:00Z	light_3.1.1_amd64.csv
full	3.1.0	arm64	2023-08-09T09:12:21Z	full_3.1.0_arm64.csv
full	3.1.0	amd64	2023-08-09T22:27:20Z	full_3.1.0_amd64.csv
ad	3.1.0	amd64	2023-08-10T00:33:47Z	ad_3.1.0_amd64.csv
ad	3.1.0	arm64	2023-08-10T00:11:36Z	ad_3.1.0_arm64.csv
web	3.1.0	amd64	2023-08-09T11:12:12Z	web_3.1.0_amd64.csv
web	3.1.0	arm64	2023-08-09T11:11:33Z	web_3.1.0_arm64.csv
osint	3.1.0	amd64	2023-08-09T10:50:11Z	osint_3.1.0_amd64.csv
osint	3.1.0	arm64	2023-08-09T10:48:56Z	osint_3.1.0_arm64.csv
light	3.1.0	amd64	2023-08-09T02:53:53Z	light_3.1.0_amd64.csv
light	3.1.0	arm64	2023-08-09T01:50:40Z	light_3.1.0_arm64.csv

Below is the list of tools featured in the latest nightly (AMD64) image.

3.20 My resources

"My-resources" brings great features allowing users to make Exegol their own and customize it even further. This feature relies on a simple volume shared between the host and all exegol containers, and an advanced integration in the Exegol images directly.

Warning: The "my-resources" feature will do what it's told to do. If users choose to use that feature to replace files or configuration, those replacements should take place. So if there are some additions to Exegol you're not getting, it could be because you have a "my-resources" setup that replaces it.

To learn more about the volume options, details are available *here*.

Below are the features offered by "My-resources", allowing users to extend Exegol beyond what is initially included (tools, resources).

- *Custom tools*: users can place their own custom standalone tools, binaries and scripts in the "my-resources" volume. This volume is accessible from all containers at `/opt/my-resources`.
- *Supported setups*: users can customize their exegol environments automatically and transparently without having to manually setting things up for each and every new Exegol container they create. In this functionality, a pre-set list of supported custom configuration is set, and will improve with time. It's the easier and most user-friendly approach to customizing a few configurations.
- *User setup*: In this functionality, a shell script can be populated with every command a user wishes its containers to run at their creation.

3.20.1 Custom tools

See also:

Available from version `3.0.0` of any exegol image.

In the container, the `/opt/my-resources/bin/` folder (`~/.exegol/my-resources/bin/` on the host) is automatically added to the `PATH` of the zsh shell. The user can then add tools in that folder in order to use them from the container.

Hint: The most simple approach would be to add standalone binaries, but users could also add symbolic links that would point to somewhere else in `/opt/my-resources/`.

```
# Example for a standalone binary
cp /path/to/tool ~/.exegol/my-resources/bin/

# Example for a symbolic link
git -C ~/.exegol/my-resources/ clone "https://github.com/someauthor/sometool"
ln -s ~/.exegol/my-resources/sometool/script.py ~/.exegol/my-resources/bin/script.py
```

3.20.2 Supported setups

Configuration files stored in the `/opt/my-resources/setup/` directory will be deployed on the containers and allow users to customize Exegol even further. By default, the number of officially supported configuration files is limited, and it depends on the version of the image itself, not the wrapper.

Hint: In order to see what configuration files are supported in your version, the `/opt/supported_setups.md` documentation file can be read from any container.

This documentation will reference in detail all the supported customizations available over time, and the corresponding minimum image version required for each one.

If a user wants to deploy tools and configurations that are not supported, or more advanced, they can opt for the *User setup solution*.

`apt` (packages, sources, keys)

See also:

Available from version `3.0.0` of any exegol image.

A system exists to easily install arbitrary APT packages in any new exegol container.

- Custom APT **repositories** can be added in exegol by filling in the `/opt/my-resources/setup/apt/sources.list` file
- Importing custom repositories usually requires importing **GPG keys** as well, which can be done by entering trusted GPG keys download URLs in the `/opt/my-resources/setup/apt/keys.list` file
- To install **APT packages** automatically (after updating the repository including the custom ones), just enter a list of package names in the `/opt/my-resources/setup/apt/packages.list` file

`bloodhound` (customqueries, config)

See also:

Available from version `3.1.0` of the `ad` and `full` images.

A system exists to easily add one or **several** bloodhound customqueries files, or change its configuration file in any new exegol container.

To automatically:

- overwrite the `~/.config/bloodhound/config.json` configuration file, simply create the file `/opt/my-resources/setup/bloodhound/config.json`
- replace the default exegol customqueries, place one or several valid customqueries files into the folder `/opt/my-resources/setup/bloodhound/customqueries_replacement/`.
- merge with the default exegol customqueries by placing one or several valid customqueries files into the folder `/opt/my-resources/setup/bloodhound/customqueries_merge/`

Tip: To be considered for replacing or merging, the customqueries files must be **valid** and bear the `.json` extension. The file names do not matter. The output will be saved into the single file `~/.config/bloodhound/customqueries.json`.

`firefox` (addons, CA)

See also:

Available from version `3.0.2` of any exegol image.

A system exists to easily install arbitrary firefox addons in any new exegol container.

The `/opt/my-resources/setup/firefox/addons.txt` file allows the user to list addons to install from online sources. It must be filled with their links in Mozilla's shop (for example https://addons.mozilla.org/fr/firefox/addon/foxyproxy-standard/).

The `.xpi` files in `/opt/my-resources/setup/firefox/addons/` folder will be installed as well.

See also:

Below, available from version `3.2.0` of any exegol image.

The `.der` files in `/opt/my-resources/setup/firefox/CA/` folder will be trusted .

`python3` (pip3)

See also:

Available from version `3.0.0` of any exegol image.

A system exists to easily install arbitrary PIP3 packages in any new exegol container.

The `/opt/my-resources/setup/python3/requirements.txt` file allows the user to list a set of packages to install with constraints just like a classic **requirements.txt** file.

`tmux` (conf)

See also:

Available from version `3.0.0` of any exegol image.

Exegol supports overloading its **tmux** configuration to allow all users to use their personal configuration.

- To automatically overwrite the `~/.tmux.conf` configuration file, simply create the file `/opt/my-resources/setup/tmux/tmux.conf`

Tip: It is possible to install **plugins** with the APT customization system, details *here*.

`vim` (vimrc, configs)

See also:

Available from version `3.0.0` of any exegol image.

Exegol supports overwriting its **vim** configuration to allow all users to use their personal configuration.

- To automatically overwrite the `~/.vimrc` configuration file, simply create the file `/opt/my-resources/setup/vim/vimrc`
- **vim configuration folders are also automatically synchronized:**
  - `/opt/my-resources/setup/vim/autoload/*` –> `~/.vim/autoload/`
  - `/opt/my-resources/setup/vim/backup/*` –> `~/.vim/backup/`

- `/opt/my-resources/setup/vim/colors/*` -> `~/.vim/colors/`
- `/opt/my-resources/setup/vim/plugged/*` -> `~/.vim/plugged/`
- `/opt/my-resources/setup/vim/bundle/*` -> `~/.vim/bundle/`

Tip: It is possible to install **plugins** with *the APT customization system*.

`neovim` (.config/nvim)

See also:

Will be available from version `3.1.2` of any exegol image.

Exegol supports overwriting its **neovim** configuration to allow all users to use their personal configuration.

- To automatically overwrite the `~/.config/nvim/` configuration, copy your config in `/opt/my-resources/setup/nvim/`

Tip: It is possible to install **plugins dependencies** with *the APT customization system*.

`zsh` (aliases, zshrc, history)

See also:

Available from version `3.0.0` of any exegol image.

To not change the configuration for the proper functioning of exegol but allow the user to add aliases and custom commands to zshrc, additional configuration files will be automatically loaded by zsh to take into account the customization of the user .

- **aliases**: any custom alias can be defined in the `/opt/my-resources/setup/zsh/aliases` file. This file is automatically loaded by zsh.
- **zshrc**: it is possible to add commands at the end of the zshrc routine in `/opt/my-resources/setup/zsh/zshrc` file.
- **history**: it is possible to automatically add history commands at the end of `~/.zsh_history` from the file `/opt/my-resources/setup/zsh/history`.

Tip: It is possible to install **plugins** with the APT customization system, details *here*.

3.20.3 User setup

See also:

Available from version `3.0.0` of any exegol image.

The `/opt/my-resources/setup/load_user_setup.sh` script is executed on the first startup of each new container that has the "my-resources" feature enabled. Arbitrary code can be added in this file, in order to customize Exegol (dependency installation, configuration file copy, etc).

Warning: It is strongly advised **not** to overwrite the configuration files provided by exegol (e.g. `/root/.zshrc`, `/opt/.exegol_aliases`, ...), official updates will not be applied otherwise.

3.20.4 Troubleshooting

In case of problem, the customization system logs all actions in the `/var/log/exegol/load_setups.log` file.

If the whole installation went smoothly the log file will be compressed by gunzip and will have the name `/var/log/exegol/load_setups.log.gz`

Tip: Logs in `.gz` format can be viewed directly **without unpacking** them with the `zcat`, `zgrep`, `zdiff` or `zmore` command!

3.21 Credentials

Some tools are pre-configured with the following credentials

Element	User	Password
neo4j database	neo4j	exegol4thewin
bettercap ui	bettercap	exegol4thewin
trilium	trilium	exegol4thewin
empire	empireadmin	exegol4thewin
wso-webshell (PHP)		exegol4thewin

3.22 Services list

This section lists the services that can be used in Exegol containers and their associated default ports.

Note: Note that, as of 25/10/2023, a utility is being developped in order to randomize those ports so that multiple containers being used concurrently don't have their services step on one another if they share a network interface. This utility will be mostly transparent, and will modify the services configuration files dynamically.

Service	Por	Commands	Comments
neo4j • bolt • http • https	768 747 737	`neo4j start`, `neo4j stop`, `neo4j restart`	Used by BloodHound, and BloodHound-related projects.

BloodHound-CE	103	`bloodhound-ce`
		`bloodhound-ce-reset`
		`bloodhound-ce-stop`
			BloodHound Community Edition Web Interface

postgresql	543	`service postgresql [...]`	Used by BloodHound CE
Trilium	199	`trilium-start`, `trilium-stop`	Collaborative note taking app. https://github.com/zadam/trilium
Burp Suite	808	`burpsuite`	HTTP(S) Proxy
Starkiller (Empire)	TBI	`ps-empire server`	GUI for the Empire post-exploit framework (https://github.com/BC-SECURITY/Empire)
Havoc	400	`havoc client/server`	C2 Framework in GO (https://github.com/HavocFramework/Havoc)
Desktop • vnc • web-sock-ify	633 ran-don	`desktop-start`, `desktop-stop`, `desktop-restart`	Remote graphical desktop feature (beta). Used with the `--desktop` from up-to-date wrapper.

3.23 Resources

Exegol's "offline resources" are a neat choice of standalone tools and scripts that are often used during penetration tests, CTFs and red-teams. While many penetration testers download those resources again every time they need them, Exegol users don't have to. Everything is managed by the wrapper and they are shared with every container by default (at `/opt/resources`).

3.23.1 Resources list

Hint: The list featured here is automatically generated. Exegol features CI/CD pipelines that build the images, update the resources, etc. When a change is made on the Exegol-resources repository, it's reflected here, in the list.

Resource	Link
SysInternals	https://learn.microsoft.com/en-us/sysinternals
pspy	https://github.com/DominicBreuker/pspy
PEASS-ng	https://github.com/carlospolop/PEASS-ng
linux-smart-enumeration (lse.sh)	https://github.com/diego-treitos/linux-smart-enumeration
LinEnum	https://github.com/rebootuser/LinEnum
Linux Exploit Suggester	https://github.com/The-Z-Labs/linux-exploit-suggester
Mimikatz	https://github.com/gentilkiwi/mimikatz
SharpHound.exe	https://github.com/BloodHoundAD/BloodHound
JuicyPotato.exe	https://github.com/ohpe/juicy-potato

Table 2 – continued fr

Resource	Link
PrintSpoofer	https://github.com/itm4n/PrintSpoofer
GodPotato	https://github.com/BeichenDream/GodPotato
static netcat (linux)	https://github.com/andrew-d/static-binaries
static netcat (windows)	https://gitlab.com/onemask/pentest-tools
SpoolSample.exe	https://gitlab.com/onemask/pentest-tools
DiagHub.exe	https://gitlab.com/onemask/pentest-tools
LaZagne	https://github.com/AlessandroZ/LaZagne
Sublinacl.exe	https://gitlab.com/onemask/pentest-tools
plink.exe	https://www.cog-genomics.org/plink/
deepce	https://github.com/stealthcopter/deepce
Some webshells	
ysoserial	https://github.com/pwntester/ysoserial
http-put-server	https://gist.githubusercontent.com/mildred/67d22d7289ae8f16cae7/raw/214c213c9415da18a471d
Chisel	https://github.com/jpillora/chisel
ligolo-ng	https://github.com/nicocha30/ligolo-ng
bitleaker	https://github.com/kkamagui/bitleaker
napper	https://github.com/kkamagui/napper-for-tpm
mimipenguin	https://github.com/huntergregal/mimipenguin
p0wny-shell	https://github.com/flozz/p0wny-shell
Inveigh	https://github.com/Kevin-Robertson/Inveigh
MailSniper	https://github.com/dafthack/MailSniper
PowerSploit	https://github.com/PowerShellMafia/PowerSploit
PrivescCheck	https://github.com/itm4n/PrivescCheck
SharpCollection	https://github.com/Flangvik/SharpCollection
WinEnum	https://github.com/neox41/WinEnum
impacket-examples-windows	https://github.com/maaaaz/impacket-examples-windows
nishang	https://github.com/samratashok/nishang

3.24 Users

This part of the documentation is meant for Exegol users, those who want to understand the project a bit more, open issues, get in touch with the community, etc..

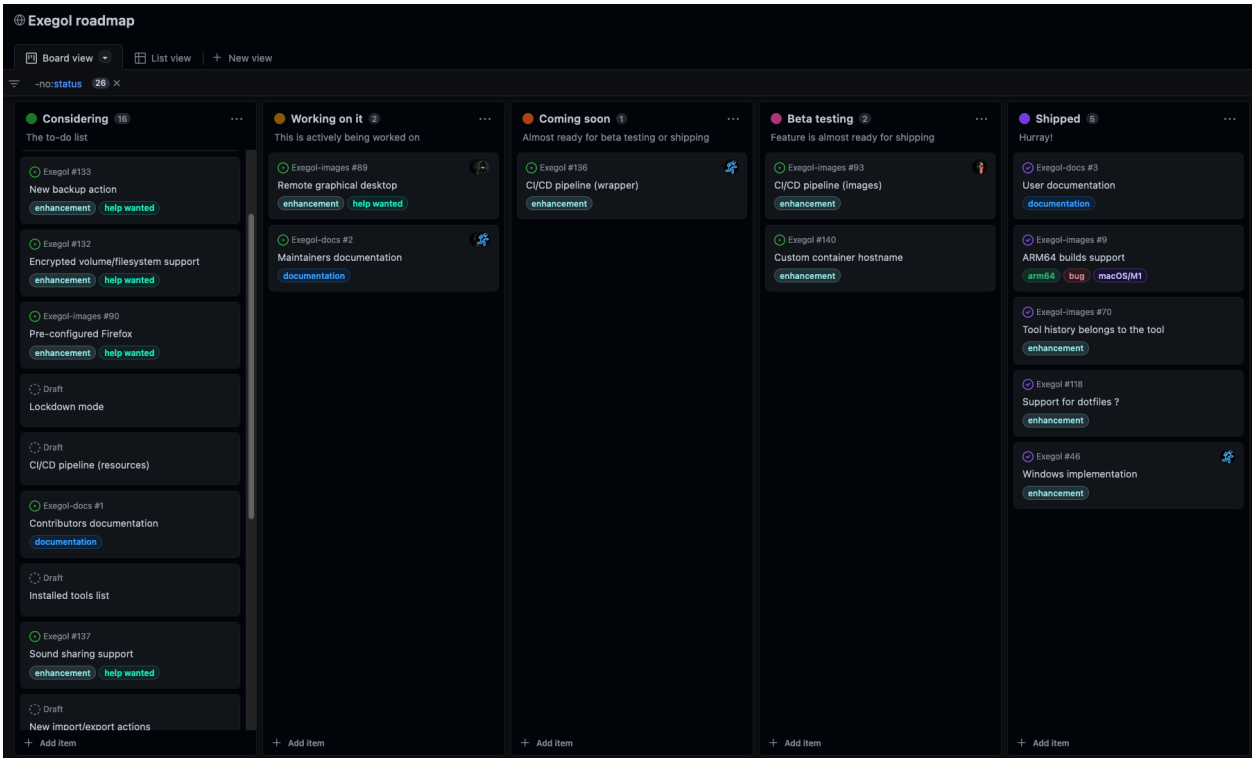
3.24.1 Opening issues

Create an issue in the correct repository:

- For any problem concerning Exegol WRAPPER (the exegol command).
- For any problem concerning Exegol IMAGE (the exegol environment).
- For any problem concerning Exegol RESOURCE (the exegol offline resources).
- For any problem concerning Exegol DOCUMENTATION (the exegol documentation).

3.24.2 Roadmap

The roadmap is available on GitHub: https://github.com/orgs/ThePorgs/projects/1/views/1



3.24.3 Discord

An Exegol discord has been created to facilitate exchanges between the community, open tickets, share ideas, vote on future features to prioritize etc..

3.25 Contributors

This part of the documentation is meant for Exegol contributors, those who write code and open pull requests. If adds up to the *users* documentation.

First things first, once you know on what module you want to contribute (wrapper, images, documentation, resources, etc.) fork it, checkout to the `dev` branch, then come back to this page to start coding.

3.25.1 Documentation

A new feature, whether it's on the wrapper, images, or any other module, must be documented accordingly. Make sure to open a pull request to the appropriate Exegol docs branch on top of your wrapper/images/whatever pull request.

Table 3: Exegol-docs branches

Branch	Purpose
main	nothing gets pushed there. This branch is made to merge with the other branches.
dev-wrapper	Related to the wrapper (Exegol)
dev-images	Related to the images (Exegol-images)
dev	General purpose

Before pushing a pull request on the documentation repository, it is advised to try and compile locally to make sure there are no errors and everything renders as expected. First, the requirements listed in `requirements.txt` must be installed (e.g. `pip install --user -r ./requirements.txt`). Then, the one-liner below can be used to remove any previous build, compile again and open the build in a browser.

```
rm -r build; make html; open "build/html/community/contributors.html"
```

Nota bene: in the example above, the `open` command opens an Internet browser (it's a macOS command), but it can be replaced by anything else that fits the contributor's environement (e.g. `firefox`).

3.25.2 Images

The Docker images are the heart of the Exegol project. A neat choice of tools, configurations, aliases, history commands, and various customizations are prepared in multiple images adapted for multiple uses: web hacking, Active Directory, OSINT (Open Source INTelligence), etc.

If you want to contribute to this part of the project, there are some things you need to know and some rules you need to follow.

Adding a new tool

In order to add a new tool to an image, here is how it goes. First, you need to figure out in what package your tool installation function must go to: packages.

Function structure

When preparing the install function to the package, don't forget to include the following functions:

- `colorecho "Installing yourtool"`: this is needed to raise logs inside the CI/CD pipeline
- `catch_and_retry <some command>`: this one is optional. When a command uses the Internet and could potentially fail randomly, the `catch_and_retry` wrapper is here to retry that commands multiple times with increasing time intervals in order to avoid having a whole build fail because of one temporary network error. Nota bene: most standard Internet-involved commands are transparently put behind a `catch_and_retry` (e.g. `git`, `wget`, `curl`, `go`, etc.).
- `add-aliases yourtool`: if your tool needs to have one or multiple aliases to work properly. You will need to create the aliases file in `/sources/assets/shells/aliases.d/` named after your tool. This file must contain the alias(es) to set as follows.

```
alias tool.py='python3 /opt/tools/yourtool/tool.py'
```

- `add-history yourtool`: if it's relevant to give some command example of your tool. No need to populate the history with a command that's very short or never used. Using long arguments is preferred. Using environment variables is preferred (e.g. `$USER`, `$PASSWORD`, `$TARGET`, etc.). You will need to create the history file in `/sources/assets/shells/history.d/` named after your tool. This file must contain the history command(s) like the example below.

```
yourtool.py --user "$USER" --password "$PASSWORD" --target "$TARGET"
yourtool.py --mode enum --user "$USER" --target "$TARGET"
yourtool.py --mode unauthenticated
```

- `add-test-command "testcommand"`: this is needed by the CI/CD pipeline to conduct unit tests for all tools to make sure they are installed properly before publishing new images. The test command needs to return `0` if the tool works properly, anything else if it doesn't. For instance, something like `yourtool.py --help` usually works, but not always! In order to find what command can be used for unit tests, you can do something like `yourtool.py --help; echo $?` to see what code is returned after the command is executed. Once trick that can be used when the `--help` command returns something `!=0` is to do some grep like `yourtool.py --help|& grep 'Usage:'`.
- `add-to-list "yourtool,https://link.to/the/tool,description"`: this is used by the CI/CD pipeline to automatically export tools in the *Tools list*. The format of the entry is standard 3-columns CSV (comma separated values). The first column is the tool name, then the link to the tool, then the description. Be careful to not have more than 2 commas and replace any comma in the description by something else.

In case your tool doesn't need aliases or history commands, add the following comment at the beggining of the tool install function: `# CODE-CHECK-WHITELIST=`. Then add a comma-separated list of the exclusions. Below are some examples.

```
# CODE-CHECK-WHITELIST=add-aliases
# CODE-CHECK-WHITELIST=add-aliases,add-history
```

TL;DR, your tool installation function should look something like this:

```
function install_yourtool() {
    colorecho "Installing yourtool"
    # tool install commands [...]
    add-aliases yourtool
    add-history yourtool
    add-test-command "yourtool.py --help"
    add-to-list "yourtool,https://link.to/the/tool,description"
}
```

Install standards

When installing a tool, depending on how it gets installed, here are the rules.

- Most tools have their virtual environment, in order to avoid dependencies conflicts. Python virtual environments must have access to the system site-packages, to avoid redundancy on already install common dependencies.
- Most tools are installed either in their own directory in `/opt/tools/` or have the binary (or a symlink) in `/opt/tools/bin/`.
- Disk space being limited, we're not pull every code source around. When possible, add the `--depth 1` option to your usual `git clone` command.

Python sources (pipx)

Python sources (venv)

APT install

Go

Ruby

Compile sources

Download compiled binary

The easiest way to install a Python tool is to use pipx.

```
# from github.com example
python3 -m pipx install git+https://github.com/AUTHOR/REPO

# from local sources
git -C /opt/tools/ clone --depth 1 https://github.com/AUTHOR/REPO
python3 -m pipx install --system-site-packages /opt/tools/yourtool/
```

But some tools cannot be installed this way, either because they're missing the `setup.py` or for any other obscure reason. In that case, opt for the "Python (venv)" solution.

In this example, the tool sources are downloaded, a virtual python environment is set up, requirements are installed, and an alias is created.

Nota bene 1: when the requirements are installed, it's better to have the command put behind a `catch_and_retry` so that if their is a temporary network outage during the build, the command will be tried multiple times with increased delays to avoid having the whole build fail.

Nota bene 2: there is no need to put standard `git`, `wget`, `curl`, `go`, and similar commands behind a `catch_and_retry` as its already handled transparently.

```
git -C /opt/tools/ clone --depth 1 https://github.com/AUTHOR/REPO
cd /opt/tools/yourtool || exit
python3 -m venv --system-site-packages ./venv/
source ./venv/bin/activate
pip3 install -r requirements.txt
deactivate
add-aliases yourtool
```

And add the following alias to your new alias file in `/sources/assets/shells/aliases.d/`

```
alias yourtool='/opt/tools/yourtool/venv/bin/python3 /opt/tools/yourtool/yourtool.py'
```

APT installations are regrouped to go faster and save some bandwith. In the `package_whatever.sh` file you're editing, look for a function called `install_*_apt_tools()`. The package you want to install needs to be added there, along with the `add-history`, `add-test-command` and `add-to-list` instructions.

Go tools can be installed with a standard `go install -v github.com/AUTHOR/REPO@latest` command.

A typical Ruby tool install will look like this:

```
function install_yourtool() {
    colorecho "Installing yourtool"
    rvm use 3.0.0@yourtool --create
    gem install yourtool
    rvm use 3.0.0@default
    add-aliases yourtool
    add-history yourtool
    add-test-command "yourtool --help"
    add-to-list "yourtool,https://github.com/AUTHOR/REPO,description"
}
```

And the alias file will look something like this.

```
alias yourtool='/usr/local/rvm/gems/ruby-3.0.0@yourtool/wrappers/ruby /usr/local/rvm/
↪gems/ruby-3.0.0@yourtool/bin/yourtool'
```

When installing a binary tool (pre-compiled or compiled live), it needs to be moved or linked in `/opt/tools/bin`. Below is an example of tool compilation and installation.

```
function install_yourtool() {
    colorecho "Installing yourtool"
    git -C /opt/tools/ clone --depth 1 https://github.com/AUTHOR/REPO
    cd /opt/tools/yourtool
    ./configure
    make
    ln -s "/opt/tools/yourtool/bin/yourtool" "/opt/tools/bin/yourtool"
    add-history yourtool
    add-test-command "yourtool --help"
```

(continues on next page)

(continued from previous page)

```
    add-to-list "yourtool,https://github.com/AUTHOR/REPO,description"
}
```

It's not uncommon to have tools already compiled, sometimes available in the "releases" section of a GitHub repository. In the following example, the latest .tar.xz release archive is dynamically fecthed from the repo, by grepping the right strings to match the name of the file and extracted. And then a symbolic link is created. The extact context can differ for each and every tool, but the example function below can serve as codebase. Trying to find similar examples in the code could also help a contributor find similar contexts and how they got implemented.

```
function install_yourtool() {
    colorecho "Installing yourtool"
    local URL
    URL=$(curl --location --silent "https://api.github.com/repos/AUTHOR/REPO/releases/
↪latest" | grep 'browser_download_url.*somestring.*tar.xz"' | grep -o 'https://[^"]*')
    curl --location -o /tmp/tool.tar.xz "$URL"
    tar -xf /tmp/yourtool.tar.xz --directory /tmp
    rm /tmp/yourtool.tar.xz
    mv /tmp/yourtool* /opt/tools/yourtool
    ln -s "/opt/tools/yourtool/bin/yourtool" "/opt/tools/bin/yourtool"
    add-history yourtool
    add-test-command "yourtool --help"
    add-to-list "yourtool,https://github.com/AUTHOR/REPO,description"
}
```

Other standards

If your tool opens ports, or if there are credentials at play, please take a look at the corresponding documentations

- *Credentials*
- *Ports & services*

Multi-architecture builds

Know that Exegol images are built by, and for, AMD64 and ARM64 systems. Most systems are AMD64 (x86_64), but some other people use ARM64 (M1/M2 Apple Sillicon chips, 64bits Raspberry-Pies, ...). Whenever possible, try to make sure your tool install function works for both architectures. Rest assured, if you don't have both architectures at your disposal it's perfectly fine, we'll take care of this part for you. If you do, and if your tool installation function includes some commands that differ wether they run on an ARM64 or AMD64 host, you can use the following structure.

```
if [[ $(uname -m) = 'x86_64' ]]
then
    # command for AMD64
elif [[ $(uname -m) = 'aarch64' ]]
then
    # command for ARM64
else
    criticalecho-noexit "This installation function doesn't support architecture $(uname␣
↪-m)" && return
fi
```

Calling the install function

Once the install function is over with, it needs to be called in the function that holds the same name as the package. For instance, if you're adding your tool install function in the `package_web.sh` package, you'll need to call that function in the `package_ad()` function (usually at the bottom of that file).

It will look something like this.

```
function package_web() {
    [...]
    install_yourtool
    [...]
}
```

Submitting the pull request

Hint: Once all your changes are over, and before submitting a pull request, it is advised to test your installation process locally. The Exegol wrapper can be used to build local images. Run `exegol install --help` to see some examples. You can also run the unit tests yourself by creating

```
# build the local image
exegol install "testimage" "full" --build-log "/tmp/testimage.log"

# create and start a container for the tests
exegol start "testcontainer" "testimage"

# run the tests (from the container)
cat /.exegol/build_pipeline_tests/all_commands.txt | grep -vE "^\s*$" | sort -u > /.
↪exegol/build_pipeline_tests/all_commands.sorted.txt
python3 /.exegol/build_pipeline_tests/run_tests.py
cat /.exegol/build_pipeline_tests/failed_commands.log
```

Warning: Your pull request needs to be made against the `dev` branch.

Once you submit your pull request, and once the various changes that may be requested are made, a CI/CD pipeline will run to make sure your code is compliant and that the tool is installed and works as intended. The pipeline may raise some issues, but if they're not related to your tool (e.g. network issues are common) don't worry about it. If the errors are due to your tool install, then you'll need to make the necessary changes to make your install work.

Once everything works, the pull request will be merged, the pipeline will run again in order to test, build and publish a new `nightly` image. Congrats, you're now an Exegol contributor!

Temporary fixing a tool

Tools sometimes have their own issues along their development. A temporary fix can be added as follows, in order to let builds pass successfully, while the respective tool is not fixed. The fix depends on the way the tool is supposed to be installed.

Git (checkout)

Git (merge PRs)

Applying the temporary fix for a tool installed through git goes as follows when checking out a previous commit

1. Find the commit id that made the tool install fail. This can be found in a try & repeat manner by installing the tool in an exegol container, checking out on a commit ID, try installing again, and repeat until it works.
2. Comment out the inital `git clone` command.
3. Add the temporary fix (`git clone` and `git checkout`) in a if statement that makes sure the fix won't stay there forever. The error message will be raised and noticed in the pipeline.
4. (bonus) create an issue on the repo (if it doesn't exist already) with the appropriate logs to help the tool's maintainers notice the installation error and fix it.

```
function install_TOOL() {
    [...]
    # git -C /opt/tools/ clone --depth 1 https://github.com/REPO/TOOL.git
    local temp_fix_limit="YYYY-MM-DD"
    if [ "$(date +%Y%m%d)" -gt "$(date -d $temp_fix_limit +%Y%m%d)" ]; then
      criticalecho "Temp fix expired. Exiting."
    else
      git -C /opt/tools/ clone https://github.com/REPO/TOOL.git
      git -C /opt/tools/TOOL checkout 774f1c33efaaccf633ede6e704800345eb313878
    fi
    [...]
}
```

When merging PRs on the fly, the temp fix goes like this

1. Find the PRs the need to be merged. **Warning: only PRs from trusted authors must be hot-merged in this manner**.
2. List the PR numbers in the PRS array
3. Merge. In the example below the `--strategy-option theirs` strategy is chosen, but it can be changed if needed.

```
function install_TOOL() {
    [...]
    git -C /opt/tools/ clone --depth 1 https://github.com/REPO/TOOL.git
    local temp_fix_limit="YYYY-MM-DD"
    if [ "$(date +%Y%m%d)" -gt "$(date -d $temp_fix_limit +%Y%m%d)" ]; then
        criticalecho "Temp fix expired. Exiting."
    else
        git config --local user.email "local"
        git config --local user.name "local"
        local PRS=("111" "222" "333")
        for PR in "${PRS[@]}"; do git fetch origin "pull/$PR/head:pull/$PR" && git merge␣
↪--strategy-option theirs --no-edit "pull/$PR"; done
```

(continues on next page)

(continued from previous page)

```
    fi
    [...]
}
```

Adding to my-resources

Hint: This documentation is not written yet... Please contact us if you would like to contribute to this part and don't know how.

3.25.3 Wrapper

Hint: This documentation is not written yet... Please contact us if you would like to contribute to this part and don't know how.

3.25.4 Signing commits

To make the project as secure as possible, signed commits are now required to contribute to the project. Using signatures for commits on GitHub serves several important purposes :

- **Authentication**: it verifies the authenticity of the commit, ensuring that it was indeed made by the person claiming to have made it.
- **Integrity**: it ensures that the commit hasn't been tampered with since it was signed. Any changes to the commit after it has been signed will invalidate the signature.
- **Trust**: this ensures that all contributions come from trusted sources.
- **Visibility**: on GitHub, signed commits are marked with a "verified" label, giving users and collaborators confidence in the commit's origin and integrity.

GitHub offers an official documentation on the matter that can be followed to setup and sign commits properly. Exegol's documentation will sum it up briefly and link to it whenever it's needed.

While **SSH (+ FIDO2)** is preferred since it offers better multi-factor signing capabilities (knowledge + hardware possession factors), people that don't have the required hardware can proceed with GPG or SSH.

GPG

SSH

SSH (+ FIDO2)

Generating a GPG key can be done by following GitHub's official documentation on the matter (generating a new GPG key). TL;DR, the commands look something like this:

```
# for the email, indicate your public email (ID+Name@users.noreply.github.com) from␣
↪https://github.com/settings/emails
gpg --quick-generate-key "YOUR_NAME <ID+Name@users.noreply.github.com>" ed25519 sign 0
gpg --list-secret-keys --keyid-format=long
gpg --armor --export $KEYID
```

Once the GPG key is generated, it can be added to the contributor's GitHub profile. Again, GitHub's documentation explains how to achieve that (adding a GPG key to your GitHub account).

Once the GPG key is generated and associated to the GitHub account, it can be used to sign commits. In order to achieve that, the contributor must configure `git` properly on his machine (telling git about your GPG key).

TL;DR: the commands look something like this to set it up for `git` CLI:

```
gpg --list-secret-keys --keyid-format=long
git config --global user.signingkey $KEYID

# (option 1) configure locally on a specific repo
cd /path/to/repository && git config commit.gpgsign true

# (option 2) configure for all git operations
git config --global commit.gpgsign true
```

To set it up on IDEs, proper official documentations can be followed (e.g. GitKraken, PyCharm).

Generating an SSH key can be done by following GitHub's official documentation on the matter (generating a new SSH key). TL;DR, the commands look something like this:

```
# for the email, indicate your public email (ID+Name@users.noreply.github.com) from␣
↪https://github.com/settings/emails
ssh-keygen -t ed25519 -C "YOUR_NAME <ID+Name@users.noreply.github.com>"
```

Once the SSH key is generated, the public part can be added to the contributor's GitHub profile. Again, GitHub's documentation explains how to achieve that (adding a new SSH key to your GitHub account).

Once the SSH key is generated and associated to the GitHub account, it can be used to authenticate and sign commits. In order to achieve that, the contributor must configure `ssh` and `git` properly on his machine (telling git about your SSH key).

TL;DR: the commands look something like this:

Hint: The `git` client version must be 2.34 or later.

```
# if setting up for the first time, configure git
git config --global user.name "YOUR_NAME"
# for the email, indicate your public email (ID+Name@users.noreply.github.com) from␣
↪https://github.com/settings/emails
git config --global user.email "ID+Name@users.noreply.github.com"

git config --global gpg.format ssh
# replace the public key path if needed, below is an example
git config --global user.signingkey "$HOME/.ssh/id_ed25519.pub"

# configure git to sign commits and tags by default
git config --global commit.gpgsign true
git config --global tag.gpgsign true

# verify commits locally, associate SSH public keys with users
mkdir -p ~/.config/git
echo "$(git config --get user.email) $(cat ~/.ssh/id_ed25519.pub)" | tee ~/.config/git/
```

(continues on next page)

(continued from previous page)

```
↪allowed_signers
git config --global gpg.ssh.allowedSignersFile "$HOME/.config/git/allowed_signers"
```

The SSH connection can then be tested as follows (testing your SSH connection).

```
# load the SSH agent into the current shell
eval "$(ssh-agent -s)"

# test the SSH authentication to GitHub servers
ssh -T git@github.com
```

This part of the doc explains how to setup and use FIDO2 security keys, such as YubiKeys, Google's Titan, etc.

First of all, a new FIDO2 key can be configured as follows to set up a PIN.

```
# list FIDO2 devices
fido2-token -L

# set a PIN for the device
fido2-token -S $device
```

Then, a resident key can be created and stored on the YubiKey as follows (see Yubico's documentation).

Hint: Some FIDO2 keys (e.g. recent YubiKeys, and probably others) support **resident keys**. A resident key is stored on the hardware key itself and easier to import to a new computer because it can be loaded directly from the security key. In order to use that feature, the `-O resident` option can be added to the `ssh-keygen` command chosen below.

```
# (default) touch only
ssh-keygen -t ed25519-sk

# PIN + touch
ssh-keygen -t ed25519-sk -O verify-required

# nothing (could be unsupported by some OpenSSH clients)
ssh-keygen -t ed25519-sk -O no-touch-required

# PIN (could be unsupported by some OpenSSH clients)
ssh-keygen -t ed25519-sk -O verify-required -O no-touch-required
```

Once the SSH key is generated, the public part can be added to the contributor's GitHub profile. GitHub's documentation explains how to achieve that (adding a new SSH key to your GitHub account).

Once a key is created and added on GitHub, it can be added to the contributor's machine SSH environment as follows. This is as easy as copy-pasting the public and private key parts to `~/.ssh`.

Hint: If you opted for a **resident key** setup, the SSH key can be loaded from the hardware key itself.

Note that those steps shouldn't be needed when the key has just been created, as the keys should automatically be added to `~/.ssh`. The commands below are mostly relevant when using **existing** resident keys on **a new system**.

```
# temporary
# needs to be done again after a reboot
```

(continues on next page)

(continued from previous page)

```
ssh-add -K

# permanent
# will download the private and public resident security keys in the current directory
# private key is to be moved in ~/.ssh (physical FIDO2 key will always be needed)
ssh-keygen -K
# it's on purpose, the "_rk" part is removed, otherwise it doesn't work.
mv id_ed25519_sk_rk ~/.ssh/id_ed25519_sk
mv id_ed25519_sk_rk.pub ~/.ssh/id_ed25519_sk.pub
```

Warning: While the `ssh-keygen -K` command saves names files `id_ed25519_sk_rk[.pub]`, it's on purpose the `_rk` part is then removed on the host. Otherwise, SSH fails at handling the keys. The files must be named `id_ed25519_sk[.pub]` on the system.

Once the SSH environment is ready, `git` CLI can be configured to rely on the security key for signing commits and authenticating (telling git about your SSH key).

Hint: The `git` client version must be 2.34 or later.

```
# if setting up for the first time, configure git
git config --global user.name "YOUR_NAME"
# for the email, indicate your public email (ID+Name@users.noreply.github.com) from␣
↪https://github.com/settings/emails
git config --global user.email "ID+Name@users.noreply.github.com"

git config --global gpg.format ssh
# replace the public key path if needed, below is an example
git config --global user.signingkey "$HOME/.ssh/id_ed25519_sk.pub"

# configure git to sign commits and tags by default
git config --global commit.gpgsign true
git config --global tag.gpgsign true

# verify commits locally, associate SSH public keys with users
mkdir -p ~/.config/git
echo "$(git config --get user.email) $(cat ~/.ssh/id_ed25519_sk.pub)" | tee ~/.config/
↪git/allowed_signers
git config --global gpg.ssh.allowedSignersFile "$HOME/.config/git/allowed_signers"
```

The SSH connection can then be tested as follows (testing your SSH connection).

```
# load the SSH agent into the current shell
eval "$(ssh-agent -s)"

# test the SSH authentication to GitHub servers
ssh -T git@github.com
```

Hint: The contributor's GitHub account can be configured to mark unsigned commits as unverified or partially verified.

While it's not mandatory regarding contributions to Exegol since the requirement is managed on Exegol repositories directly, it's a nice thing to do. See GitHub's documentation on Vigilante mode.

3.26 Maintainers

This part of the documentation is meant for Exegol maintainers. It adds up to the *contributors* documentation.

3.26.1 Wrapper release

Hint: The wrapper documentation must be aligned with the wrapper features. . The docs PR can be merged once the wrapper is released.

Preparation

1. Git updates

The first step is to update the project and sub-modules, meaning pointing the exegol-images and exegol-resources sub-modules to the latest master version. Even if the wrapper is able to auto-update itself, it is always better to keep the base reference at least up to date.

With git

With Exegol

- Update current wrapper repo:

```
git pull
```

- Update git submodules and checkout to **main** branch for release:

```
git -C exegol-docker-build checkout main
git -C exegol-docker-build pull
git -C exegol-resources checkout main
git -C exegol-resources pull
```

Update to the latest version of the **main** branches (checkout if needed, **except for the wrapper** which remains in branch dev)

```
exegol update -v
```

Important: Don't forget to **reload and commit** any **submodule update** at this step !

2. Config reviews

- Review exegol.utils.ConstantConfig variables
  - Change version number ! (remove the alpha or beta tag at the end of the version number)
- Review documentation
- Review README.md

Tests & build

First, test the code with mypy:

```
mypy exegol.py --ignore-missing-imports --check-untyped-defs
```

You can execute this one-liner to check the project and build it.

> **Warning:** **Require** build package installed!

Hint: Exegol can only be published through a **source** build distribution because of the source code files for building local images.

```
python3 setup.py clean test && \
   (rm -rf Exegol.egg-info && python3 -m build --sdist) || \
   echo "Some tests failed, check your code and requirements before publishing!"
```

Post build

- Upgrade tests.test_exegol.py version number to the next version build to avoid future mistake
- Commit updates
- Publish PR
- Wait for review and merge

Manual Upload

Important: PyPi packaging and upload is now handle by **GitHub action**. It will be triggered with the creation of the **new tag** in the next-step with the release creation.

This step is no longer needed.

After validation of the PR, we can upload the new version package to pypi.

> **Warning:** **Require** twine package installed and token configured on `~/.pypirc`!

- Check package upload on the test repository (optional)

```
python3 -m twine upload --repository testpypi dist/* --verbose
```

- Upload to the production repository

```
python3 -m twine upload dist/*
```

Post-Deploy

- Create new github **release** with **new** version tag
- Fast-forward dev branch to the latest master commit
- Change the wrapper version on the dev branch to `x.y.zb1`

3.26.2 Images release

Hint: The images documentation must be aligned with the images features. Make sure to add code to the appropriate Exegol docs branch and have a pull request ready. The docs PR can be merged once the images are released.

Prepare changes

The first step consists in preparing the `dev` branch for merge.

1. create a pull request `dev -> main` named `Release X.Y.Z` (`Release X.Y.ZbI` is also accepted, X, Y, Z and I being numbers. Creating this pull request will trigger the **pre-release** workflows. The PR comment must indicate all major changes.
2. edit the `dev` branch until the pull requests checks (pipeline) all pass, effectively publishing all images to the preproduction Dockerhub registry
3. once all checks are good, the PR needs to be approved by a maintainer.

Merge changes

Once the PR is approved and ready for merge, it can be merged

1. merge the PR with **Create a merge commit**
2. Synchronize the `dev` branch with the latest `main` update with a **fast-forward merge**

```
git checkout main
git pull --all
git checkout dev
git pull --all
git merge --ff-only main
git push
```

New tag

The `X.Y.Z` (or `X.Y.ZbI`) tag then needs to be placed on the same commit the `dev` and `main` branches point to.

Optionally, the "Annotated Tag Message" can be set to the PR initial comment with the `--file message.txt` argument in the `git tag` command below.

```
git tag "X.Y.Z"
git push origin --tags
```

Puhing this tag will trigger the **release** workflow. Simply put, the workflow will migrate the images from preprod registry to production registry.

Maintainers needs to make sure workflow goes as planned and images end up in the prod Dockerhub registry. If the release fails for some reason, the tag can be deleted, changes pushed, and then the tag can be created again to trigger the release again (`git tag -d "X.Y.Z" && git push --delete origin "X.Y.Z"`).

Publish release

The final step is to create a "release" in github (https://github.com/ThePorgs/Exegol-images/releases/new).

1. The release must point to the tag created before.
2. The release must be named `Exegol images X.Y.Z`.
3. The release notes can be created with the **Generate releases notes** button.
4. Set it as **latest release**.
5. Publish

3.26.3 CI/CD Pipeline

The Exegol project relies on a continuous integration and continuous deployment (CI/CD) pipeline for multiple scenarios. At the time of writing, Tue 31 Jan 2023, the pipeline is structured as follows:

wrapper

images

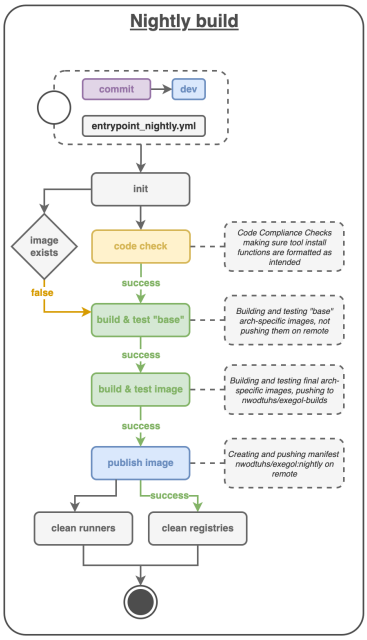
resources

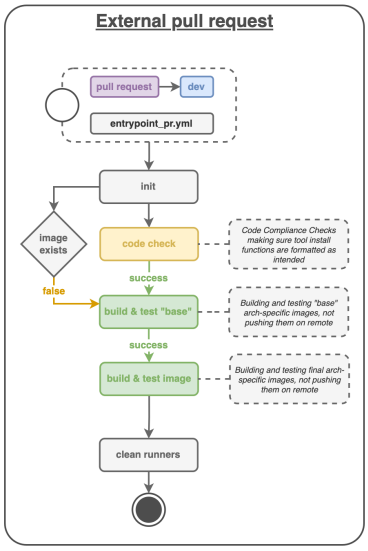
docs

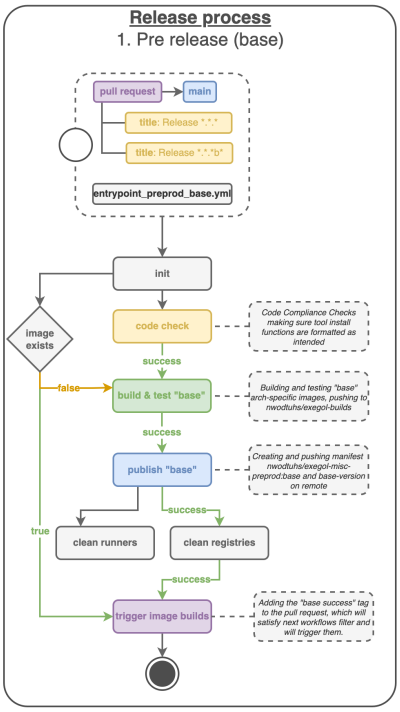
The GitHub Actions platform is used on *the Exegol module*. Its workflows are used for internal and external pull requests, new releases and testing on every commit. The workflows build, and push Python packages on the official PyPI registry, and run tests to make sure everything works as it should.

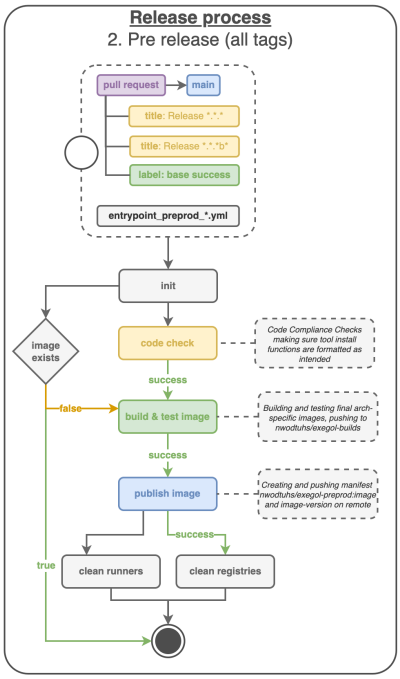
The GitHub Actions platform is used on *the Exegol-images submodule*. Its workflows run for internal and external pull requests, new commits, new tags, and allow to:

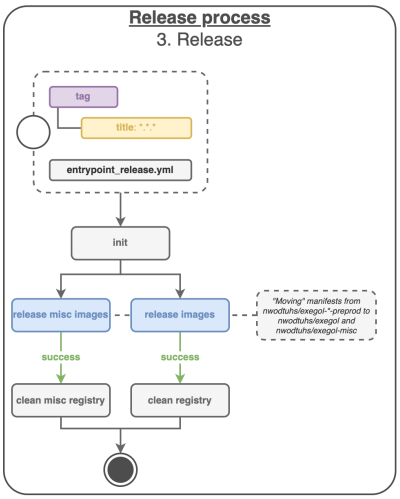
- build AMD64 and ARM64 images on self-hosted runners
- run tests to make sure the tools are installed properly
- automatically export tools list to the documentation
- push the images on the official Dockerhub registry











The GitHub Actions platform is used on *the Exegol-resources submodule*. Its workflows are used to automatically update the resources (monthly) and automatically export the list of resources to the documentation.

The GitHub Actions platform is used for the documentation you're reading. Its workflows are used to build on every commit and pull request to make sure everything works as it should, but also automatically merge changes between the various branches in order to help with development.

ReadTheDocs then builds the final version on every commit for multiple branches (main, dev, dev-images, dev-wrapper) and hosts it online at https://exegol.readthedocs.io/.

GitHub Actions

The GitHub Actions pipeline(s) need runners to operate the various jobs configured for each workflow. The Exegol project relies on self-hosted runners instead of the GitHub-hosted runners mainly for costing reasons.

At the time of writing, Tue 31 Jan 2023, the Exegol-images pipeline(s) require ARM64 and AMD64 runners in order to build, and run corresponding architectured images.

1. Setting up secrets

There are some operations that the runner will operate that will require authentication, including: - pushing Python packages on PyPI - pushing Docker images on Dockerhub

In order to allow this, GitHub Actions can be set up with secrets that the runner will be able to use later on. This part of the documentation shows what secrets must be set up and how.

PyPI

Dockerhub

API Tokens can be created in the maintainer account's PyPI account settings, in the **API Tokens** part. The scope must be set to `Project: Exegol`. The tokens are linked to the personal PyPI account.

Access Tokens can be created in the maintainer account's Dockerhub security settings. Permissions must be set to `Read, Write, Delete`. The tokens are linked to the personal Dockerhub account.

Once the token is created, it can be added as follows:

- For Exegol-images, go to the Exegol-images repo settings > secrets > actions. At the time of writing, 11 Feb. 2023, Dockerhub secrets are named `DOCKER_USERNAME` and `DOCKER_PASSWORD` in the workflows.
- For the Python wrapper, go to the Exegol repo settings > secrets > actions. At the time of writing, 11 Feb. 2023, the PyPI token is named `PYPI_API_TOKEN` in the workflows.

2. Deploying a runner

The runner can either run on macOS, Linux, or Windows, as those three operating systems are supporting by the GHA (GitHub Action) platform. x64 and ARM64 are supported for macOS and Windows, and for Linux, ARM is supported as well.

Below are the hardware requirements for each runner:

- enough RAM *(to be defined)*
- enough CPU *(to be defined)*
- enough free disk space (at least ~100GB, bare minimum)

Before deploying a GHA agent on a runner, software requirements must be met:

- Docker (or Docker Desktop for Windows and macOS)
- jq (lightweight and flexible command-line JSON processor)

Linux

macOS

For Linux systems, Docker is required in order to have the GitHub Actions agent running.

Tip: Docker can be installed quickly and easily with the following command-line:

```
curl -fsSL "https://get.docker.com/" -o get-docker.sh
sh get-docker.sh
```

Warning: To run exegol from the user environment without `sudo`, the user must have privileged rights equivalent to root. To grant yourself these rights, you can use the following command

```
# add the sudo group to the user
sudo usermod -aG docker $(id -u -n)

# "reload" the user groups
newgrp
```

The `jq` utility is also required and can be installed with the following command line:

```
apt install jq
```

Once the requirements are met, the agent can be deployed as follows (with sufficient permissions in the GitHub repository):

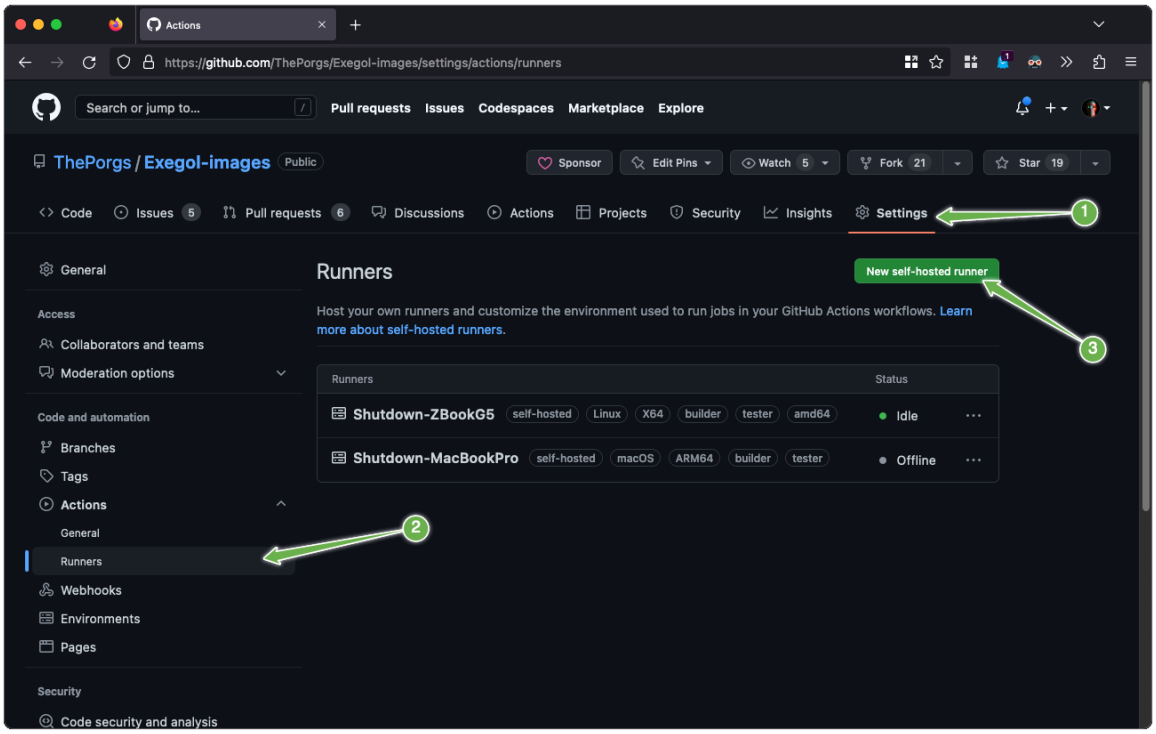
- go to https://github.com/ThePorgs/Exegol-images/settings/actions/runners
- click on "New self-hosted runner"
- select `Linux` as operating system, as well as the right architecture and follow the instructions
- when running the `config.sh` script, the following settings must be set
  - name of the runner group: Default
  - name of the runner: *up to you*
  - additional labels: `builder,tester` (adapt this if the runner is to be used for only one of those actions). If the runner is an X64/AMD64, the `AMD64` tag needs to be set as well. If the runner is ARM64, the right tag will be set automatically.
  - name of work folder: *up to you*
- start the runner with the `run.sh` script
- (option) configure the agent as a service if it is to be run unattended/headless with `sudo ./svc.sh install <user>`, more info at https://docs.github.com/en/actions/hosting-your-own-runners/configuring-the-self-hosted-runner-application-as-a-service

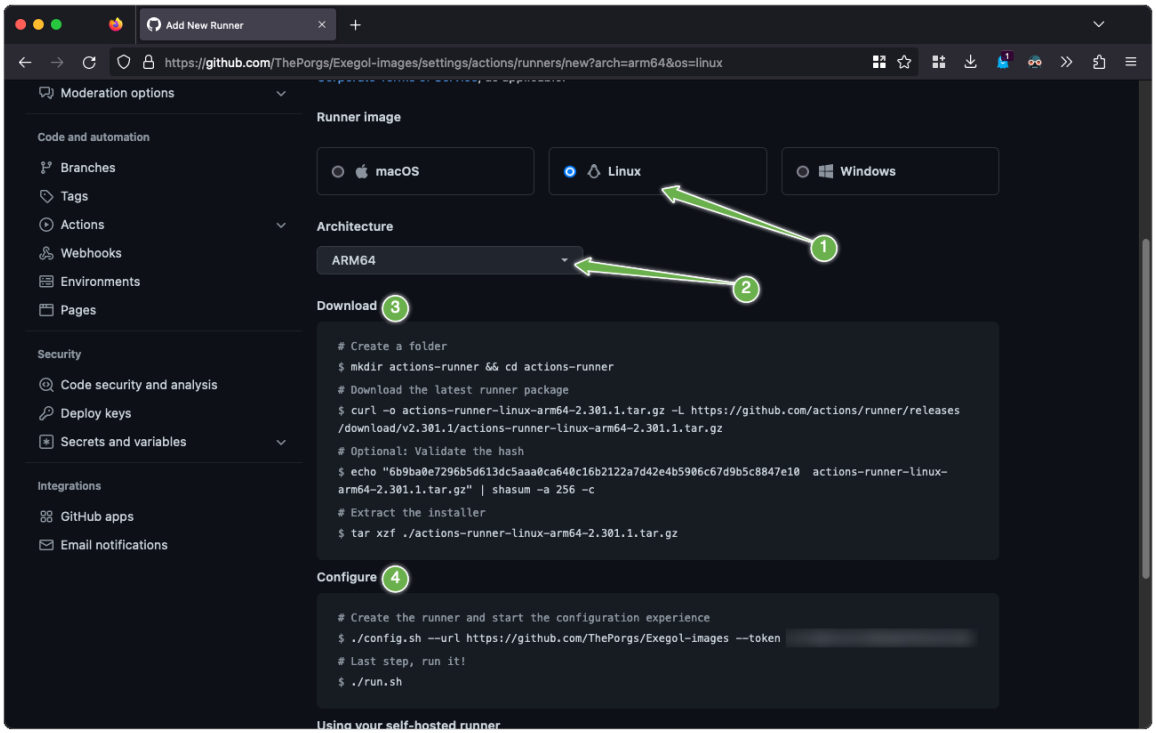
Note: When configuring the agent as a service, it will be enabled, meaning it will start at boot. The `systemctl is-enabled` command should return `enabled`.

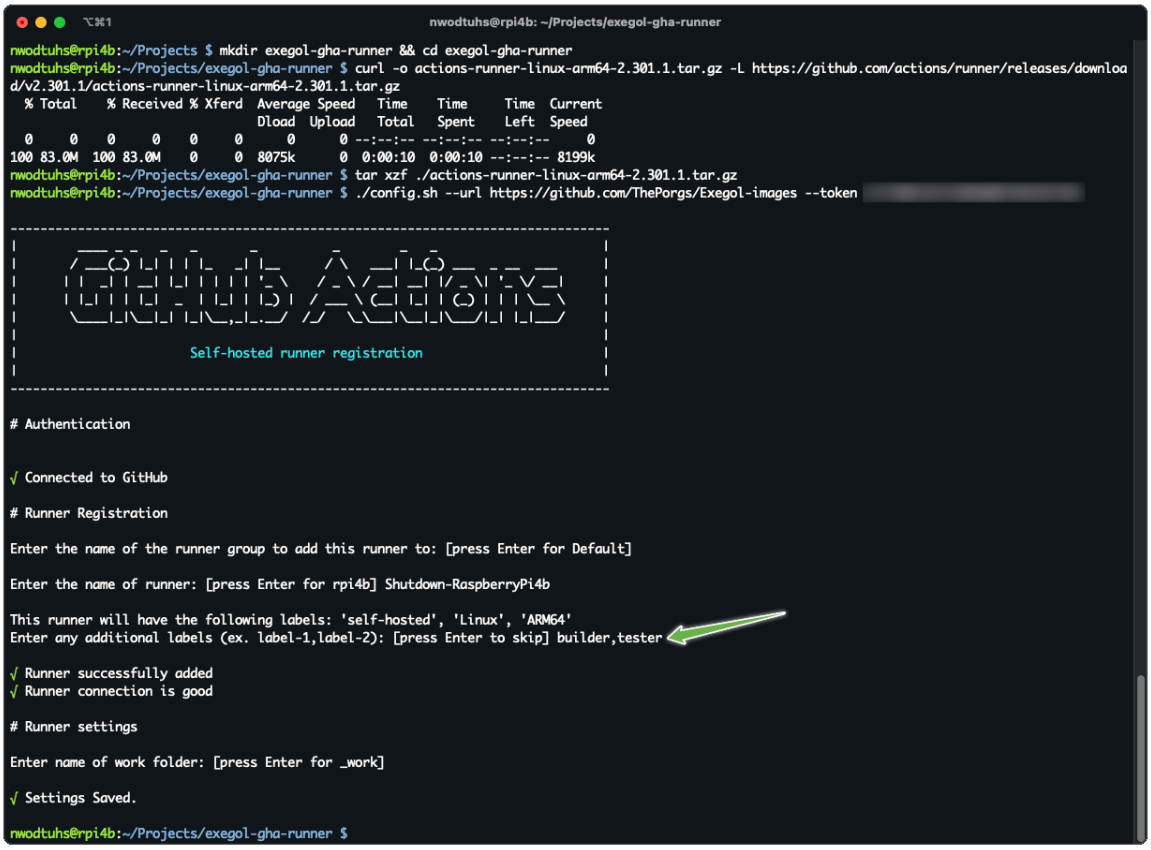
```
sudo systemctl is-enabled actions.runner.ThePorgs-Exegol-images.<runner-name>.service
```

In order to start the service, either reboot the runner, or use `systemctl`.

```
sudo systemctl start actions.runner.ThePorgs-Exegol-images.<runner-name>.service
```


Actions
https://github.com/ThePorgs/Exegol-images/settings/actions/runners
Search or jump to...
Pull requests
Issues
Codespaces
Marketplace
Explore
ThePorgs / Exegol-images
Public
Sponsor
Edit Pins
Watch 5
Fork 21
Star 19
Code
Issues 5
Pull requests 6
Discussions
Actions
Projects
Security
Insights
Settings
General
Access
Collaborators and teams
Moderation options
Code and automation
Branches
Tags
Actions
General
Runners
Webhooks
Environments
Pages
Security
Code security and analysis
Runners
New self-hosted runner
Host your own runners and customize the environment used to run jobs in your GitHub Actions workflows. Learn more about self-hosted runners.
Runners
Status
Shutdown-ZBookG5
self-hosted
Linux
X64
builder
tester
amd64
Idle
Shutdown-MacBookPro
self-hosted
macOS
ARM64
builder
tester
Offline
1
2
3


Add New Runner
https://github.com/ThePorgs/Exegol-images/settings/actions/runners/new?arch=arm64&os=linux
Moderation options
Code and automation
Branches
Tags
Actions
Webhooks
Environments
Pages
Security
Code security and analysis
Deploy keys
Secrets and variables
Integrations
GitHub apps
Email notifications
Runner image
macOS
Linux
Windows
Architecture
ARM64
Download
3
# Create a folder
$ mkdir actions-runner && cd actions-runner
# Download the latest runner package
$ curl -o actions-runner-linux-arm64-2.301.1.tar.gz -L https://github.com/actions/runner/releases/download/v2.301.1/actions-runner-linux-arm64-2.301.1.tar.gz
# Optional: Validate the hash
$ echo "6b9ba0e7296b5d613dc5aaa0ca640c16b2122a7d42e4b5906c67d9b5c8847e10 actions-runner-linux-arm64-2.301.1.tar.gz" | shasum -a 256 -c
# Extract the installer
$ tar xzf ./actions-runner-linux-arm64-2.301.1.tar.gz
Configure
4
Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/ThePorgs/Exegol-images --token
# Last step, run it!
$ ./run.sh
Using your self-hosted runner
1
2



Note: Screenshots annotated with https://annotely.com/

For macOS, **Docker Desktop** must be installed: https://docs.docker.com/desktop/install/mac-install/.

- In `Settings` > `Resources` > `Advanced`, the `virtual disk limit` must be set to at least 100GB.
- In `Settings` > `Resources` > `Advanced`, allocate enough CPUs, Memory and Swap.

The **jq** tool can be installed as follows.

```
# install brew
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)
↪" < /dev/null 2> /dev/null

# install jq
brew install jq
```

Xcode Command Line Tools are also required, and they can be installed with the following command line.

```
xcode-select --install
```

Once the requirements are met, the agent can be deployed as follows (with sufficient permissions in the GitHub repository):

- go to https://github.com/ThePorgs/Exegol-images/settings/actions/runners

- click on "New self-hosted runner"
- select `macOS` as operating system, as well as the right architecture and follow the instructions
- when running the `config.sh` script, the following settings must be set
  - name of the runner group: Default
  - name of the runner: *up to you*
  - additional labels: `builder,tester` (adapt this if the runner is to be used for only one of those actions). If the runner is an X64/AMD64, the `AMD64` tag needs to be set as well. If the runner is ARM64, the right tag will be set automatically.
  - name of work folder: *up to you*
- start the runner with the `run.sh` script
- the agent must **not** be configured as a service with `./svc.sh install`. Some errors have been raised when setting up the pipeline like this.

Note: **TODO** : how to make that service run at boot unattended without using svc.sh install?

---

3. Checking runners status

Go to https://github.com/ThePorgs/Exegol-images/settings/actions/runners

4. Understanding the pipelines

TODO explain the pipelines, include diagrams.

4. Common errors

1. docker login

When configuring a macOS agent as a service with `./svc.sh install`, the following error was met during workflow run.

```
Run docker/login-action@v2
with:
    username: ***
    password: ***
    ecr: auto
    logout: true
Logging into Docker Hub...
Error: Error saving credentials: error storing credentials - err: exit status 1, out:␣
↪`error storing credentials - err: exit status 1, out: `User interaction is not allowed.
↪``
```

In order to avoid that error, the runner was started interactively with `./run.sh`.

2. Disk space

When there's not enough disk space, the following error is usually raised by the pipelines.

```
You are running out of disk space. The runner will stop working when the machine runs␣
↪out of disk space. Free space left: 62 MB
```

3.26.4 Pull Requests

When handling pull requests, maintainers may need to synchronize a contributor's fork with latests changes. In command-line, this can be achieved as follows.

```
git clone "git@github.com:USER/FORK" "dest_dir"
cd dest_dir
git remote add upstream "git@github.com:ThePorgs/REPO"
git fetch upstream
git checkout "TARGET_FORK_BRANCH"
git merge --no-edit upstream/"ORIGIN_BRANCH"
# solve conflicts if any
git push
```

3.27 Sponsors

Dramelac and I work at Capgemini and we thank them for believing in the project since day 1, and for allowing us to have this personal initiative keep going.

We thank **HackTheBox** for continuously supporting the community and for helping us financially to acquire the necessary hardware for supporting multiple architectures (AMD64, ARM64). Show some love at https://www.hackthebox.com/ !

We thank **JetBrains** for supporting this community project through its OpenSource support program. More information at https://jb.gg/OpenSourceSupport !