Exegol

Release 4.1.0

Shutdown & Dramelac

Nov 28, 2022

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.

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

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.

Contents

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 | macOS | Windows)
- at least 20GB of free storage

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
```

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

```
sudo usermod -aG docker $(id -u -n)
```

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

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

- 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)

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

- Windows **11** is needed
- Docker must run on **WSL2** engine (how to)
- WSLg must be installed
- at least one WSL distribution must be **installed** as well (e.g. Debian), with **Docker integration** enabled

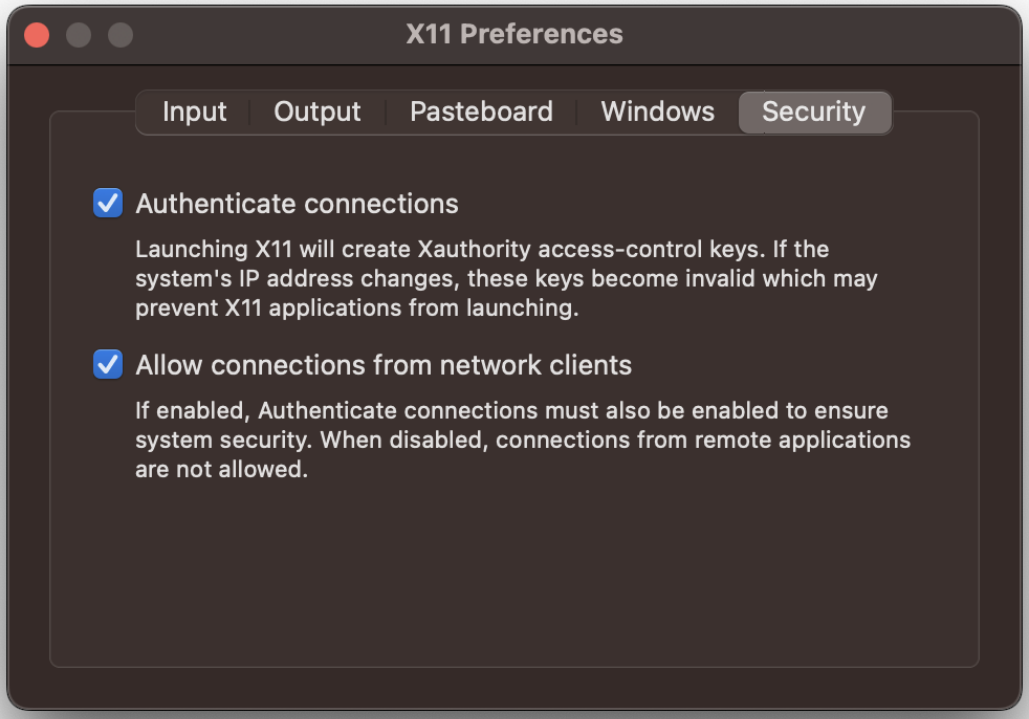

Fig. 1: macOS XQuartz configuration requirement

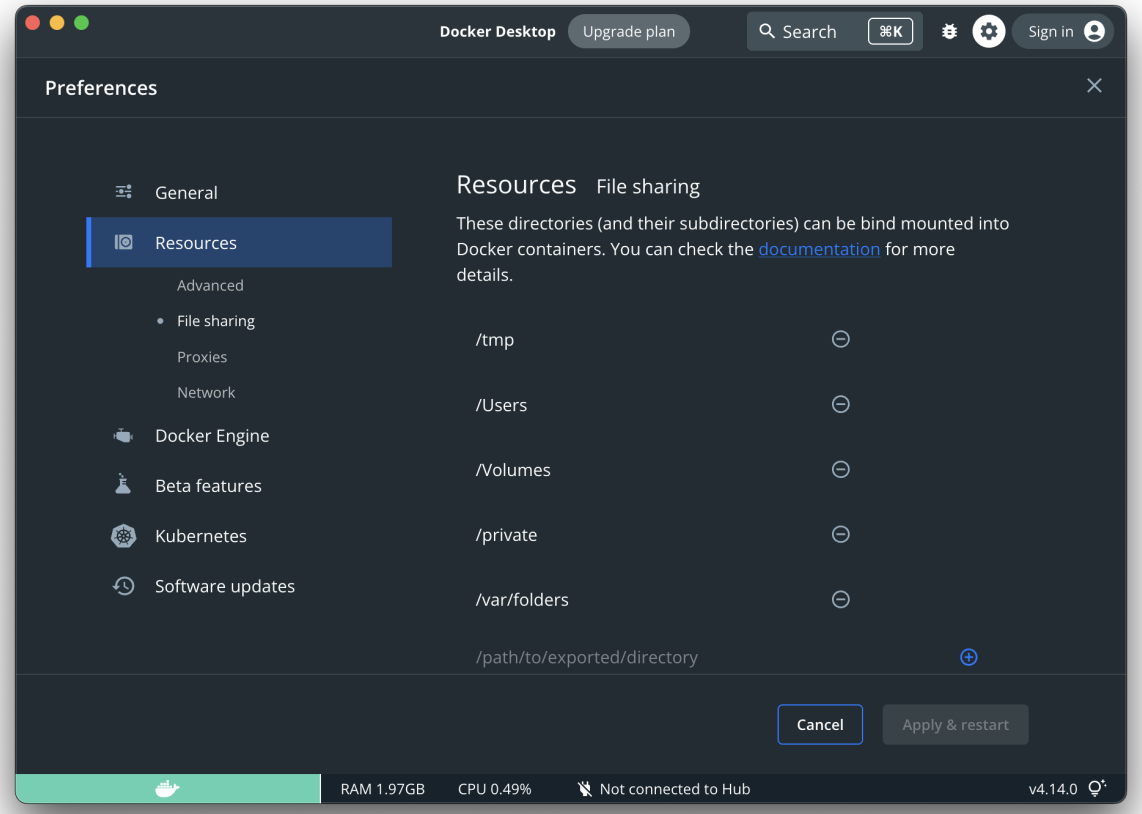

Fig. 2: macOS Docker Desktop resources requirement

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

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
```

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

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.py" "/usr/local/bin/exegol"
```

Once this is taken care of, the exegol wrapper can then can be added as a PowerShell command alias and saved for persistence in `$HOME\PowershellAliasesExport.txt` then loaded from `$PROFILE` script at PowerShell startup. 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:

```
$AliasFile = "$HOME\PowershellAliasesExport.txt"
Set-Alias -Name exegol -Value "$(pwd)\exegol.py"
Get-Alias -Name "exegol" | Export-Alias -Path $AliasFile
echo "Import-Alias '$AliasFile'" >> $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. 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 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:

Feature	Description
Display sharing	Sharing of the graphic environment between the container and the host
Workspace	Persistent and shared workspace with the host
Update-fs	Permission sharing between the container and the host
OpenVPN connection	Opening an isolated VPN tunnel dedicated to the exegol container
Shell logging	Recording of sessions (input and output) in log files with date and time
Shared network	Sharing the host's network interfaces
Shared timezones	Sharing the host's timezone configuration
Exegol-resources	Easy access to a collection of resources and tools
My-resources	User space dedicated to customization
Volume sharing	Support for specific volume addition
Port sharing	Support for port publishing
Env. variables	Support for environment variable configuration
Device sharing	Support for hardware sharing
Capabilities	Support for adding specific capabilities
Privileged	Support of the privileged mode
Multi-architecture	Support for AMD64 and ARM64 architectures
Local image	Customized local image building
Remote image	Pre-built image available for download
Command execution	Execution of specific command
Daemon execution	Support of the command execution in the background
Temporary containers	Support for command execution in a dedicated and temporary environment

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.

3.3.1 Features

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

Display sharing

By default exegol configures the new container and host to allow the execution to the display of graphical window launched from an exegol container.

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*.

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+s`).

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.

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:
>
> SYS_RAWIO, SYS_MODULE, NET_BROADCAST, NET_ADMIN, SYS_CHROOT, MKNOD, NET_RAW, SYS_ADMIN, SETFCAP, SYS_PTRACE
>
> 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.

3.4 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.

3.5 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`).

3.6 `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
```

3.6.1 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_PROFILE`	Optional positional argument to indicate the source profile to use if building locally.
`--build-log`	Write logs to the path specified if building locally.

3.6.2 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

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

3.7 `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
```

3.7.1 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*.

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/).
-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.
--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 display sharing to run GUI-based applications. (default: Enabled)
--disable-my-resources	Disable the mount of the shared resources (/opt/my-resources) from the host (/home/dramelac/.exegol/my-resources) (default: Enabled)
--disable-exegol-resources	Disable the mount of the exegol resources (/opt/resources) from the host (/home/dramelac/Documents/tools/Exegol/exegol-resources) (default: Enabled)
--disable-shared-network	Disable the sharing of the host's network interfaces with exegol (default: Enabled)
--disable-shared-timezone	Disable the sharing of the host's time and timezone configuration with exegol (default: Enabled)

An additional feature of Exegol is the addition of a 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.

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.

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.

3.7.2 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

# Use a Proxmark
exegol start -d "/dev/ttyACM0"

# Use an HackRF One
exegol start -d "/dev/bus/usb/"
```

3.8 `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.

3.8.1 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)

3.8.2 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
```

3.9 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.

3.9.1 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.

3.9.2 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

# 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
```

3.10 `update` action

3.10.1 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.

3.10.2 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*.

3.10.3 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
```

3.11 `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.

3.11.1 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

3.11.2 Command examples

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

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

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

3.12 `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.

3.12.1 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).

3.12.2 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"
```

3.13 `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.

3.13.1 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.

3.13.2 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"
```

3.14 `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.

3.14.1 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)

3.14.2 Command examples

```
# Show version information
exegol version

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

3.15 Advanced uses

Contents

3.15.1 Exegol home directory

The `~/.exegol` folder exists in the user's home folder to centralize:doc:*exegol resources </the-exegol-project/offline-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/`.

3.15.2 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

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`)

3.15.3 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.

3.15.4 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.

3.16 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

3.17 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. 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 store add 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.

Contents

3.17.1 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
```

3.17.2 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

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.

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*.

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*.

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*.

3.17.3 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.

3.17.4 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!

3.18 Intro

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`).

3.19 Users

3.19.1 Roadmap

The roadmap is available on the discord server, feel free to visit it and upvote your favorite features

3.19.2 Discord

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

3.19.3 Changelogs

WIP

3.20 Contributors

3.20.1 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).

3.20.2 Opening pull request

Every PRs are welcome!

Describe your addition / bug fix and configure your PR to the **dev** branch.

3.20.3 Dev docs

WIP : Coming 'soon'

3.21 Maintainers

This part of the documentation is meant for Exegol maintainers.

3.21.1 Exegol Release checklist

Preparation

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
```

Configs

- 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

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

Upload

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`

3.22 Sponsors

Dramelac and I work at **Capgemini** and we thank them for allocating some time for us to develop and maintain Exegol! Visit Capgemini website at https://www.capgemini.com/.

We also 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/ !

3.23 Credits

Credits and thanks go to every infosec addicts that contribute and share but most specifically to

- dramelac_ for his incredible help on the whole Exegol project, especially on the wrapper.
- LamaBzh for working on Exegol-images
- th1b4ud for the inspiration "Kali Linux in 3 seconds with Docker".
- all behind Docker, Debian, Python, the various requirements to the project, all the tools and resources loaded in Exegol, and all Exegol users for their trust and patience!