gvm-tools Documentation

Release 22.9.1.dev1

Greenbone Networks GmbH

Dec 09, 2022

CONTENTS

The Greenbone Vulnerability Management Tools, or **gvm-tools** in short, are a collection of tools that help with controlling a Greenbone Enterprise Appliance and Greenbone Community Edition installations remotely.

Essentially, the tools aid accessing the communication protocols *Greenbone Management Protocol (GMP)* and *Open Scanner Protocol (OSP)*.

Note: **gvm-tools** requires at least Python 3.7. Python 2 is not supported.

CHAPTER

ONE

INSTALLATION OF GVM-TOOLS

The current universally applicable installation process for Python is using the pip installer tool in conjunction with the pypi package repository.

1.1 Installing the Latest Stable Release of gvm-tools

For installing the latest stable release of **gvm-tools** from the Python Package Index, pip or poetry can be used.

1.1.1 Using pip

The following command installs **gvm-tools** system wide:

```
python3 -m pip install gvm-tools
```

A system wide installation usually requires admin permissions. Therefore, **gvm-tools** may only be installed for the current user via:

```
python3 -m pip install --user gvm-tools
```

For further details and additional installation options, please take a look at the documentation of pip.

1.1.2 Using poetry

To avoid polluting the system and user namespaces with Python packages and to allow installing different versions of the same package at the same time, python virtual environments have been introduced.

poetry is a tool combining the use of virtual environments and handling dependencies elegantly.

Please follow the poetry documentation to install the tool.

To install **gvm-tools** into a virtual environment, defaulting into the folder *.venv*, the following command need to be executed:

```
poetry install
```

Afterwards, the environment containing the installed **gvm-tools** can be activated by running:

```
poetry shell
```

It is also possible to run single commands within the virtual environment:

```
poetry run gvm-cli -h
```

1.2 Getting the Source

The source code of **python-gvm** can be found at GitHub.

To clone this public repository and install from source for the current user run the following commands:

```
git clone git://github.com/greenbone/gvm-tools.git && cd gvm-tools
python3 -m pip install -e .
```

CHAPTER

TWO

CONNECTION TYPES

Before being able to talk to a remote *GMP* or *OSP* server using one of the *provided command line clients*, the user has to choose a connection type for establishing a communication channel. Currently three different connection types are supported for being used as transport protocol:

- *TLS – tls*
- *SSH – ssh*
- *Unix Domain Socket – socket*

For the most common use case (querying *gvmd* via *GMP* on the same host) the *socket connection* should be chosen. The other connection types require some setup and possible adjustments at the server side, if no *Greenbone OS* based system is used.

2.1 Using a Unix Domain Socket

The Unix Domain Socket is the default connection type of *gvmd* in the *Greenbone Community Edition*. It is only usable when running the client tool on the same host as the daemon.

The location and name of the Unix Domain Socket provided by *gvmd* highly depends on the environment and *Greenbone Community Edition* installation.

For current releases of the Greenbone Community Edition 21.4 and 22.4 the socket should be found at `/run/gvmd/gvmd.sock`.

For *GOS 4* the path is either `/run/openvas/openvasmd.sock` or `/usr/share/openvas/gsa/classic/openvasmd.sock` and for *GOS 5* and later the path is either `/run/gvm/gvmd.sock` or `/usr/share/gvm/gsad/web/gvmd.sock`.

OSPd based scanners may be accessed via Unix Domain Sockets as well. The location and name of these sockets is configurable and depends on the used OSPd scanner implementation.

Warning: Accessing a Unix Domain Socket requires sufficient Unix file permissions for the user running the *command line interface tool*.

Please do not start a tool as **root** user via **sudo** or **su** only to be able to access the socket path. Instead, adjust the socket file permissions, e.g. by setting the **--listen-owner**, **--listen-group** or **--listen-mode** arguments of *gvmd*.

2.2 Using TLS

The TLS connection type was the default connection type for remote and local communication in *GOS 3.1* and before. It is used to secure the transport protocol connection of *GMP* or *OSP*. It requires to provide a TLS certificate file, TLS key file and TLS certificate authority file.

2.3 Using SSH

Since *GOS 4*, SSH is the default connection type for secure remote communication with the manager daemon via *GMP*. The *Greenbone Management Protocol* is tunneled through SSH and forwarded to *gvmd*.

CHAPTER

THREE

PROVIDED TOOLS

Currently, **gvm-tools** comes with three command line interface programs:

- *gvm-cli*
- *gvm-script*
- *gvm-pyshell*

All of these programs are clients communicating either via *Greenbone Management Protocol (GMP)* or *Open Scanner Protocol (OSP)*. The *connection* is established using a *TLS*, *SSH* or *Unix Domain Socket* communication channel.

All tools take several arguments and parameters. **gvm-tools** allows setting defaults for most of these in a configuration file. See *Configuration* for details about the possible settings and capabilities.

3.1 gvm-cli

gvm-cli is a low level tool which offers sending and receiving commands and responses for the XML-based *GMP* and *OSP* directly via the command line. It is intended for *simple scripting* via shell.

```
> gvm-cli --help
usage: gvm-cli [-h] [-c [CONFIG]]
               [--log [{DEBUG,INFO,WARNING,ERROR,CRITICAL}]]
               [--timeout TIMEOUT] [--gmp-username GMP_USERNAME]
               [--gmp-password GMP_PASSWORD] [-V]
               CONNECTION_TYPE ...

optional arguments:
  -h, --help            show this help message and exit
  -c [CONFIG], --config [CONFIG]
                        Configuration file path (default: ~/.config/gvm-
                        tools.conf)
  --log [{DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                        Activate logging (default level: None)
  --timeout TIMEOUT     Response timeout in seconds, or -1 to wait
                        indefinitely (default: 60)
  --gmp-username GMP_USERNAME
                        Username for GMP service (default: '')
  --gmp-password GMP_PASSWORD
                        Password for GMP service (default: '')
  -V, --version         Show version information and exit
```

(continues on next page)

(continued from previous page)

```
connections:
  valid connection types

  CONNECTION_TYPE       Connection type to use
    ssh                 Use SSH to connect to service
    tls                 Use TLS secured connection to connect to service
    socket              Use UNIX Domain socket to connect to service
```

Examples:

```
> gvm-cli socket --xml "<get_version/>"
<get_version_response status="200" status_text="OK"><version>7.0</version></get_version_
↪response>

> gvm-cli socket --xml "<get_tasks/>"
<get_tasks_response status="200" status_text="OK">
...
</get_tasks_response>

> gvm-cli socket < commands.xml
```

3.2 gvm-script

New in version 2.0.

gvm-script allows running *gvm scripts* which are Python based scripts calling the python-gvm API. Depending on the **--protocol** argument a global gmp or osp object is passed to the script.

Note: **gvm-script** is only available with **gvm-tools** version 2.0 and beyond.

```
usage: gvm-script [-h] [-c [CONFIG]]
                  [--log [{DEBUG,INFO,WARNING,ERROR,CRITICAL}]]
                  [--timeout TIMEOUT] [--gmp-username GMP_USERNAME]
                  [--gmp-password GMP_PASSWORD] [-V] [--protocol {GMP,OSP}]
                  CONNECTION_TYPE ...

optional arguments:
  -h, --help            show this help message and exit
  -c [CONFIG], --config [CONFIG]
                        Configuration file path (default: ~/.config/gvm-
                        tools.conf)
  --log [{DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                        Activate logging (default level: None)
  --timeout TIMEOUT     Response timeout in seconds, or -1 to wait
                        indefinitely (default: 60)
  --gmp-username GMP_USERNAME
                        Username for GMP service (default: '')
  --gmp-password GMP_PASSWORD
                        Password for GMP service (default: '')
```

(continues on next page)

(continued from previous page)

```
  -V, --version         Show version information and exit
  --protocol {GMP,OSP}  Service protocol to use (default: GMP)

connections:
  valid connection types

  CONNECTION_TYPE       Connection type to use
    ssh                 Use SSH to connect to service
    tls                 Use TLS secured connection to connect to service
    socket              Use UNIX Domain socket to connect to service
```

3.3 gvm-pyshell

gvm-pyshell is a tool to use the python-gvm API interactively. Running the tool will open a Python interpreter in the interactive mode providing a global gmp or osp object depending on the **--protocol** argument.

The interactive shell can be exited with:

- Ctrl+D on Linux or
- Ctrl+Z on Windows

```
> gvm-pyshell --help
usage: gvm-pyshell [-h] [-c [CONFIG]]
                   [--log [{DEBUG,INFO,WARNING,ERROR,CRITICAL}]]
                   [--timeout TIMEOUT] [--gmp-username GMP_USERNAME]
                   [--gmp-password GMP_PASSWORD] [-V] [--protocol {GMP,OSP}]
                   CONNECTION_TYPE ...

optional arguments:
  -h, --help            show this help message and exit
  -c [CONFIG], --config [CONFIG]
                        Configuration file path (default: ~/.config/gvm-
                        tools.conf)
  --log [{DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                        Activate logging (default level: None)
  --timeout TIMEOUT     Response timeout in seconds, or -1 to wait
                        indefinitely (default: 60)
  --gmp-username GMP_USERNAME
                        Username for GMP service (default: '')
  --gmp-password GMP_PASSWORD
                        Password for GMP service (default: '')
  -V, --version         Show version information and exit
  --protocol {GMP,OSP}  Service protocol to use (default: GMP)

connections:
  valid connection types

  CONNECTION_TYPE       Connection type to use
    ssh                 Use SSH to connect to service
    tls                 Use TLS secured connection to connect to service
    socket              Use UNIX Domain socket to connect to service
```

Example:

```
> gvm-pyshell socket
GVM Interactive Console 2.0.0 API 1.0.0. Type "help" to get information about␣
↪functionality.
>>> gmp.get_protocol_version()
'7'
>>> gmp.get_version().get('status')
'200'
>>> gmp.get_version()[0].text
'7.0'
>>> [t.find('name').text for t in tasks.xpath('task')]
['Scan Task', 'Simple Scan', 'Host Discovery']
```

CHAPTER

FOUR

CONFIGURATION

Changed in version 2.0.

By default, **gvm-tools** *programs* are evaluating the `~/.config/gvm-tools.conf` ini style config file since version 2.0. The name of the used config file can be set using the **-c/--config** command line switch.

4.1 Settings

The configuration file consists of sections, each led by a `[section]` header, followed by key/value entries separated by a = character. Whitespaces between key and value are ignored, i.e., `key = value` is the same as `key=value`.

Currently five sections are evaluated:

- *Main section*
- *GMP section*
- *Socket section*
- *TLS section*
- *SSH section*

Main Section

The main section allows changing the default connection timeout besides defining variables for *Interpolation*.

```
[main]
timeout = 60
```

GMP Section

The GMP section allows setting the default user name and password for Greenbone Management Protocol (GMP) based communication.

```
[gmp]
username=gmpuser
password=gmppassword
```

Socket Section

This section is only relevant if the *socket connection type* is used.

The socket section allows setting the default path to the Unix Domain socket of *gvmd*. It must not be confused with the socket path to the redis server used by *openvas*.

```
[unixsocket]
socketpath=/run/gvmd/gvmd.sock
```

TLS Section

This section is only relevant if the *TLS connection type* is used.

The TLS section allows setting the default port, TLS certificate file, TLS key file and TLS certificate authority file.

```
[tls]
port=1234
certfile=/path/to/tls.cert
keyfile=/path/to/tls.key
cafile=/path/to/tls.ca
```

SSH Section

This section is only relevant if the *SSH connection type* is used.

The SSH section allows setting the default SSH port, SSH user name and SSH password.

```
[ssh]
username=sshuser
password=sshpassword
port=2222
```

Comments

Configuration files may also contain comments by using the special character `#`. A comment should be placed on a separate line above or below the setting.

```
[main]
# connection timeout of 120 seconds
timeout=120
```

Interpolation

The configuration file also supports the interpolation of values. It is possible to define values in the `[main]` section and reference them via a `%(<variablename>)s` syntax. Additionally, values of the same section can be referenced.

```
[main]
my_first_name=John

[gmp]
```

(continues on next page)

(continued from previous page)

```
my_last_name=Smith
username=%(my_first_name)s%(my_last_name)s
```

Using this syntax will set the gmp user name setting to *JohnSmith*.

4.2 Example

Full example configuration:

```
[main]
# increased timeout to 5 minutes
timeout = 300
tls_path=/data/tls
default_user=johnsmith

[gmp]
username=%(default_user)s
password=choo4Gahdi2e

[unixsocket]
socketpath=/run/gvmd/gvmd.sock

[tls]
port=1234
certfile=%(tls_path)s/tls.cert
keyfile=%(tls_path)s/tls.key
cafile=%(tls_path)s/tls.ca

[ssh]
username=%(default_user)s
password=Poa8Ies1iJee
```

CHAPTER

FIVE

SCRIPTING

5.1 XML Scripting

Note: XML scripting via **gvm-cli** should only be considered for simpler use cases. *Greenbone Management Protocol (GMP) or Open Scanner Protocol (OSP) scripts* are often more powerful and easier to write.

Scripting via **gvm-cli** is directly based on GMP and OSP. Both protocols make use of XML command requests and corresponding responses.

A typical example for using GMP is the automatic scan of a new system. In the example below, it is assumed that an Intrusion Detection System (IDS) that monitors the systems in the Demilitarized Zone (DMZ) and immediately discovers new systems and unusual, new TCP ports is in use. If such an event is being discovered, the IDS should automatically initiate a scan of the new system. This can be done with the help of a script.

1. Starting point is the IP address of the new suspected system. For this IP address, a target needs to be created on the *Greenbone Enterprise* Appliance.

   If the IP address is saved in the environment variable `IPADDRESS` by the IDS, the respective target can be created:

```
> gvm-cli socket --xml "<create_target><name>Suspect Host</name><hosts>"$IPADDRESS"</
↪hosts></create_target>"
<create_target_response status="201" status_text="OK, resource created" id="e5adc10c-
↪71d0-49fe-aacf-a442ee31d387"/>
```

See **create_target** command for all details.

2. Create a task using the default *Full and Fast* scan configuration with UUID `daba56c8-73ec-11df-a475-002264764cea` and the previously generated target:

```
> gvm-cli socket --xml "<create_task><name>Scan Suspect Host</name><target id=\"e5adc10c-
↪71d0-49fe-aacf-a442ee31d387\"/><config id=\"daba56c8-73ec-11df-a475-002264764cea\"/>
↪<scanner id=\"08b69003-5fc2-4037-a479-93b440211c73\"/></create_task>"
<create_task_response status="201" status_text="OK, resource created" id="7249a07c-03e1-
↪4197-99e4-a3a9ab5b7c3b"/>
```

See **create_task** command for all details.

3. Start the task using the UUID return from the last response:

```
> gvm-cli socket --xml "<start_task task_id=\"7249a07c-03e1-4197-99e4-a3a9ab5b7c3b\"/>"
<start_task_response status="202" status_text="OK, request submitted"><report_id>
↪0f9ea6ca-abf5-4139-a772-cb68937cdfbb</report_id></start_task_response>
```

See **start_task** command for all details.

→ The task is running. The response returns the UUID of the report which will contain the results of the scan.

4. Display the current status of the task:

```
> gvm-cli socket --xml "<get_tasks task_id=\"7249a07c-03e1-4197-99e4-a3a9ab5b7c3b\"/>"
<get_tasks_response status="200" status_text="OK">
...
<status>Running</status><progress>98 ... </progress>
...
<get_tasks_response/>
```

See **get_tasks** command for all details.

→ As soon as the scan is completed, the full report is available and can be displayed.

5. Display the full report:

```
> gvm-cli socket --xml "<get_reports report_id=\"0f9ea6ca-abf5-4139-a772-cb68937cdfbb\"/>
↪"
<get_reports_response status="200" status_text="OK"><report type="scan" id="0f9ea6ca-
↪abf5-4139-a772-cb68937cdfbb" format_id="a994b278-1f62-11e1-96ac-406186ea4fc5"␣
↪extension="xml" content_type="text/xml">
...
</get_reports_response>
```

See **get_reports** command for all details.

6. Additionally, the report can be downloaded in a specific report format instead of plain XML.

   List all report formats:

```
> gvm-cli socket --xml "<get_report_formats/>"
<get_report_formats_response status="200" status_text="OK"><report_format id="5057e5cc-
↪b825-11e4-9d0e-28d24461215b">
...
</get_report_formats_response>
```

See **get_report_formats** command for all details.

7. Download the report in the desired format.

   Example: download the report as a PDF file:

```
> gvm-cli socket --xml "<get_reports report_id=\"0f9ea6ca-abf5-4139-a772-cb68937cdfbb\"␣
↪format_id=\"c402cc3e-b531-11e1-9163-406186ea4fc5\"/>"
```

Note: Please be aware that the PDF is returned as base64 encoded content of the *<get_report_response><report>* element in the XML response.

5.2 GVM Scripts

Changed in version 2.0.

Scripting of *Greenbone Management Protocol (GMP)* and *Open Scanner Protocol (OSP)* via **gvm-script** or interactively via **gvm-pyshell** is based on the python-gvm library. Please take a look at python-gvm for further details about the API.

Note: By convention, scripts using *GMP* are called *GMP scripts* and are files with the ending `.gmp.py`. Accordingly, *OSP scripts* with the ending `.osp.py` are using *OSP*. Technically both protocols could be used in one single script file.

The following sections are using the same example as it was used in *XML Scripting* where it was assumed that an Intrusion Detection System (IDS) that monitors the systems in the Demilitarized Zone (DMZ) and immediately discovers new systems and unusual, new TCP ports is in use. The IDS will provide the IP address of a new system to the GMP script.

1. Define the function that should be called when the script is started by adding the following code to a file named `scan-new-system.gmp.py`:

```
if __name__ == '__gmp__':
  main(gmp, args)
```

→ The script is only called when being run as a GMP script. The *gmp* and *args* variables are provided by **gvm-cli** or **gvm-pyshell**. *args* contains arguments for the script, e.g., the user name and password for the GMP connection. The most important aspect about the example script is that it contains the *argv* property with the list of additional script specific arguments. The *gmp* variable contains a connected and authenticated instance of a Greenbone Management Protocol class.

2. The main function begins with the following code lines:

```
def main(gmp: Gmp, args: Namespace) -> None:
  # check if IP address is provided to the script
  # argv[0] contains the script name
  if len(args.argv) <= 1:
    print('Missing IP address argument')
    return 1

  ipaddress = args.argv[1]
```

→ The main function stores the first argument passed to the script as the `ipaddress` variable.

3. Add the logic to create a target, create a new scan task for the target, start the task and print the corresponding report ID:

```
ipaddress = args.argv[1]

target_id = create_target(gmp, ipaddress)

full_and_fast_scan_config_id = 'daba56c8-73ec-11df-a475-002264764cea'
openvas_scanner_id = '08b69003-5fc2-4037-a479-93b440211c73'
task_id = create_task(
    gmp,
    ipaddress,
    target_id,
```

(continues on next page)

(continued from previous page)

```
    full_and_fast_scan_config_id,
    openvas_scanner_id,
)

report_id = start_task(gmp, task_id)

print(
    f"Started scan of host {ipaddress}. Corresponding report ID is {report_id}"
)
```

For creating the target from an IP address (DNS name is also possible), the following is used. Since target names must be unique, the current date and time in ISO 8601 format (YYYY-MM-DDTHH:MM:SS.mmmmmm) is added:

```
def create_target(gmp, ipaddress):
    import datetime

    # create a unique name by adding the current datetime
    name = f"Suspect Host {ipaddress} {str(datetime.datetime.now())}"
    response = gmp.create_target(name=name, hosts=[ipaddress])
    return response.get('id')
```

The function for creating the task is defined as:

```
def create_task(gmp, ipaddress, target_id, scan_config_id, scanner_id):
    name = f"Scan Suspect Host {ipaddress}"
    response = gmp.create_task(
        name=name,
        config_id=scan_config_id,
        target_id=target_id,
        scanner_id=scanner_id,
    )
    return response.get('id')
```

Finally, the function to start the task and get the report ID:

```
def start_task(gmp, task_id):
    response = gmp.start_task(task_id)
    # the response is
    # <start_task_response><report_id>id</report_id></start_task_response>
    return response[0].text
```

For getting a PDF document of the report, a second script `pdf-report.gmp.py` can be used:

```
from base64 import b64decode
from pathlib import Path


def main(gmp: Gmp, args: Namespace) -> None:
    # check if report id and PDF filename are provided to the script
    # argv[0] contains the script name
    if len(args.argv) <= 2:
        print('Please provide report ID and PDF file name as script arguments')
        return 1
```

(continues on next page)

(continued from previous page)

```
    report_id = args.argv[1]
    pdf_filename = args.argv[2]

    pdf_report_format_id = "c402cc3e-b531-11e1-9163-406186ea4fc5"
    response = gmp.get_report(
        report_id=report_id, report_format_id=pdf_report_format_id
    )

    report_element = response[0]
    # get the full content of the report element
    content = "".join(report_element.itertext())

    # convert content to 8-bit ASCII bytes
    binary_base64_encoded_pdf = content.encode('ascii')
    # decode base64
    binary_pdf = b64decode(binary_base64_encoded_pdf)

    # write to file and support ~ in filename path
    pdf_path = Path(pdf_filename).expanduser()
    pdf_path.write_bytes(binary_pdf)

    print('Done.')


if __name__ == '__gmp__':
    main(gmp, args)
```

5.3 Example Scripts

All example scripts can be found at GitHub.

CHAPTER

SIX

GLOSSARY

gvmd
: Management daemon shipped with *GVM 10* and later. Abbreviation for **G**reenbone **V**ulnerability **M**anager **D**aemon.

openvassd
: Scanner daemon used by *GVM 10* and before. It listens for incoming connections via OTP and starts scan processes to run the actual vulnerability tests. It collects the results and reports them to the management daemon. With *GVM 11* it has been converted into the *openvas* application by removing the daemon and OTP parts. Abbreviation for **OpenVAS S**canner **D**aemon.

openvas
: Scanner application executable to run vulnerability tests against targets and to store scan results into a redis database. Used in *GVM 11* and later. It has originated from the *openvassd* daemon.

OSPd
: A framework for several scanner daemons speaking the *Open Scanner Protocol (OSP)*.

ospd-openvas
: A *OSP* scanner daemon managing the *openvas* executable for reporting scan results to the management daemon *gvmd*. Used in *GVM 11* and later.

GOS
: Greenbone Operating System, the operating system of the *Greenbone Enterprise* Appliance. It provides the commercial version of the *Greenbone Community Edition* with enterprise support and features.

GSM
: Greenbone Security Manager (GSM) is the former name of our commercial product line *Greenbone Enterprise* as hardware or virtual appliances.

GMP
: The Greenbone Management Protocol (GMP) is an XML-based communication protocol provided by *gvmd*. It provides an API to create, read, update and delete scans and vulnerability information.

OSP
: The Open Scanner Protocol is an XML-based communication protocol provided by *ospd-openvas*. It provides an API to start scans, get *VT* information and to receive scan results.

GVM
: The *Greenbone Community Edition* consists of several services. This software framework has been named Greenbone Vulnerability Management (GVM) in the past.

Greenbone Community Edition
: The Greenbone Community Edition covers the actual releases of the Greenbone application framework for vulnerability scanning and vulnerability management provided as open-source software to the community. The Greenbone Community Edition is adopted by external third parties, e.g., if the software framework is provided

by a Linux distribution, it is build from the Greenbone Community Edition. It is developed as part of the commercial *Greenbone Enterprise* product line. Sometimes referred to as the OpenVAS framework.

GVM9
Version 9 of the *Greenbone Community Edition*. Also known as **OpenVAS 9**. Used in the *GOS 4* series.

GVM10
Version 10 of the *Greenbone Community Edition*. Used in *GOS 5*.

GVM11
Version 11 of the *Greenbone Community Edition*. Used in *GOS 6*.

GVM20.08
Version 20.08 of the *Greenbone Community Edition*. Used in *GOS 20.08*. First version using Calendar Versioning

GVM21.4
Version 21.4 of the *Greenbone Community Edition*. Used in *GOS 21.04*.

GVM22.4
Version 22.4 of the *Greenbone Community Edition*. Used in *GOS 22.04*.

Greenbone Enterprise
Greenbone Enterprise is the Greenbone product line for on-premises solutions. Included are virtual or hardware Greenbone Enterprise Appliances with the *Greenbone Operating System (GOS)*, the *Greenbone Community Edition* framework, and the *Greenbone Enterprise Feed*.

Greenbone Community Feed
The Greenbone Community Feed is the freely available feed for vulnerability information licensed as open-source. It contains basic scan configurations, report formats, port lists and the most important vulnerability tests. The provided data is updated on a daily basis with no warranty or promises for fixes or completeness.

Greenbone Enterprise Feed
The Greenbone Enterprise Feed is the commercial feed provided by Greenbone Networks containing additional enterprise features like vulnerability tests for enterprise products, policy and compliance checks, extensive reports formats and special scan configurations. The feed comes with a service-level agreement ensuring support, quality assurance and availability.

VT
Vulnerability Tests (VTs), also known as Network Vulnerability Tests (NVTs), are scripts written in the NASL programming language to detect vulnerabilities at remote hosts.

INDEX