



## Overview

**ZAP** is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing as well as experienced security professionals.

**ZAP** provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

You should only use **ZAP** to attack an application that you have permission to test. If you are worried about using **ZAP** then switch to using the 'Safe mode' via the top level toolbar: this will significantly reduce **ZAP**'s functionality but will prevent you from causing any damage.

You can download **ZAP** from <https://code.google.com/p/zaproxy/>  
**ZAP** runs on Windows, Linux and Mac OS: the only dependency is Java 7.

## The User Interface

By default **ZAP** displays:

- A **top level menu** which gives access to many of the automated and manual tools
- A **top level toolbar** which includes buttons for commonly used features
- A **'tree' window** on the left hand side which displays the Sites tree and the Scripts tree
- A **'workspace'** window on the top right hand side, which allows you to display and edit requests, responses and scripts
- An **'information' window** underneath workspace window, which displays details of the automated and manual tools
- A **footer** which displays a summary of the alerts found and the status of the main automated tools

**In order to reduce the complexity of the UI many of ZAPs features are only shown via context sensitive right-click menus.**

There are right-click menus throughout ZAP: on the nodes of the Sites and Script trees, on the tables displayed in the information tabs and in the workspace tabs. Some menus are only displayed when you highlight text - for example the 'Fuzz...' menu is only shown if you right click a highlighted string in the Request tab.

**When you start using ZAP it is worth right-clicking everywhere until you get used to the UI.**

All of the menus can also be accessed via keyboard shortcuts. Default shortcuts are defined for most of the menus, but they are all user configurable via the Tools / Options... menu. You can even generate and print off a cheatsheet for the shortcuts you have configured via the Keyboard Options page.

## Quick Start - Attack

The quickest way to get started with **ZAP** is to use the Quick Start tab, which allows you to enter a single URL that **ZAP** will attack. **ZAP** will use its spider to crawl the application, which will automatically passively scan all of the pages discovered. It

will then use the active scanner to attack all of the pages. This is a useful way to perform an initial assessment of an application, but it does have some drawbacks.

Firstly **ZAP** will not handle any authentication, which means that it will not discover any pages protected by a login page. Secondly you have much less control of the exploring and attacking phases, its a quick way to get started, but there's much more to ZAP than this.

## The Spiders

**ZAP** can only attack the pages of an application that it is aware of, which means that you must explore your application in some way. The Quick Start tab uses the 'traditional' spider which discovers links by examining the HTML in the application responses. You can run this spider independently via the Spider tab or the right click Attack menu.

This spider is very fast, but it is not always effective when exploring an AJAX application that generates links using JavaScript.

For AJAX applications the AJAX spider is likely to be more effective. This spider explores the application by invoking browsers which then follow the links which have been generated.

The AJAX spider is slower than the tradition spider and cannot be used in a 'headless' environment.

## Proxying your browser

The spiders are an effective way of exploring an application but ideally they should be combined with manual exploration. The spiders will only enter basic default data into forms, whereas a user can enter more relevant information. This may well open up more of the application. For example, a 'New user' form may ask for a username, and entering a value like 'aaa' in the username field causes an error to be displayed indicating that the username should be a valid email address. Neither of the spiders will be able to act on this information, while a human will be able to react to the error, supply a valid username and then open up more of the application's functionality.

In order to explore an application manually you should configure your browser to proxy via **ZAP**. If you are using a recent version of Firefox then the easiest way to do this is via the 'Plug-n-Hack' button on the Quick Start tab. You can also manually configure all modern browser to use **ZAP** as a proxy. See the **ZAP** help file or your browser's documentation for more details.

## Active and passive scanning

**ZAP** passively scans all of the requests and responses that it discovers via the spiders or that are proxied through it from your browser. Passive scanning does not change the responses in any way and is therefore always safe to use. Scanned is performed in a background thread to ensure that it does not slow down the exploration of an application. Passive scanning is good for finding a limited number of potential vulnerabilities, such as missing security related HTTP headers. It can be an effective way to get a sense of the state of security in a given web application, and clues for where to focus more invasive manual testing.

Active scanning attempts to find potential vulnerabilities by using known attacks against the selected targets. As active scanning is an attack on those targets it is completely under user control and should only be used against applications that you have permission to test. Active scanning can be started via the Active Scan tab or the right click 'Attack' menu.

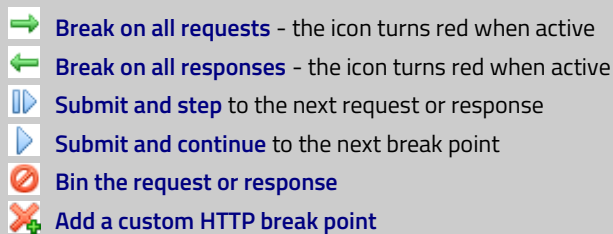
**ZAP** has very fine grained control of exactly which active and passive scan rules are run and how thoroughly they check for vulnerabilities. this allows you to tune **ZAP** to look more effectively for vulnerabilities common to the applications you are testing and to avoid false positives.

All of the potential vulnerabilities are shown in the Alerts tab. You can generate HTML and XML reports including all vulnerabilities via the Reports menu.

## Changing requests and responses

**ZAP** is an intercepting proxy, which means you can change requests and responses on the fly. All data that passes through **ZAP** can be changed, including HTTP, HTTPS, WebSockets and postMessage traffic.

The following toolbar buttons are used to control 'break points' in the traffic:



Intercepted messages are displayed in the Break tab and can be changed prior to submitting. Custom break points can be used to intercept messages based on the criteria you specify - this can be useful when dealing with an application that makes lots of requests. You can also trigger break points from Proxy Scripts which allows you to specify as complex a criteria as you need.

## Using ZAP in Continuous Integration

**ZAP** is an ideal tool to use for security testing in a Continuous Integration environment (CI), allowing you to find vulnerabilities soon after code has been checked into source control. You can run **ZAP** in headless mode using the '-daemon' command line option and then controlling it using the built in REST API.

A high level approach could be:

1. **Proxy browser based regression tests** (eg using Selenium) through **ZAP** in order to explore the application in a realistic way
2. **Use the spiders** to discover content not covered by regression tests
3. **Run the active scanner** to attack the application
4. **Read the alerts** found and report any new vulnerabilities

Tools like the spiders and active scanners are asynchronous, so you will need to invoke them using the relevant 'scan' action and then poll the 'status' view until they complete. **ZAP** supports an HTML interface to the API which you can use to explore it manually - just proxy your browser via **ZAP** and visit <http://zap/> or select the Tools / Browse API menu.

## The Marketplace

When you start **ZAP** the second time it will ask if you want to automatically check for updates. This is recommended but you can choose not to turn this on and check manually. **ZAP** is made up of a set of 'core' features that can only be updated by 'full' **ZAP** releases. However many features are implemented as 'add-ons' that can be updated independently without

requiring a new **ZAP** release. There are also a wide range of add-ons that are not included by default but can be downloaded and installed from within **ZAP**. These add-ons can add minor new features or major new functionality.

Add-ons are assigned a status which may be one of:

- **Release** which indicates they are of high quality and fit for purpose
- **Beta** which indicates they are of reasonable quality but may be incomplete or need further testing
- **Alpha** which indicates they are at an early stage of development


The latest versions of all official **ZAP** add-ons are available via the **ZAP** Marketplace, which can be accessed via the Help / Check for Updates... menu or the Manage Add-ons button: 

Many add-ons can be installed dynamically and are available to use without needing to restart **ZAP**. You can configure **ZAP** to alert you when new release, beta or alpha add-ons become available.

## Saving ZAP sessions

You can save **ZAP** sessions so that all of the data recorded is available to you later. **ZAP** actually stores all of the messages in a temporary database on file store, but this database is deleted when you stop **ZAP** unless you choose to persist it. If you think you'll want to save your session then we recommend that you persist the session at the start, before you start your testing. This will copy the temporary database into a permanent one. You can persist your session at any stage in your testing, but persisting a database with lots of data will take significantly longer than one that is empty. You only need to persist your session once, **ZAP** will keep updating the permanent database as you carry on testing. You can take 'snapshots' of the session database if you want to record the state at a particular time. These snapshots are not updated unless you open them and resume testing.

## More information

**ZAP** includes a context sensitive help file which includes information about all of the available functionality, which you can access via the F1 key, the Help / OWASP **ZAP** User Guide menu or the various help buttons on the toolbars: 

On the **ZAP** Google Code pages (<http://code.google.com/p/zaproxy>) you will find tutorial videos, conference videos and a wiki with a wealth of information.

There are also online groups which you can join to ask questions about **ZAP**:

<http://groups.google.com/group/zaproxy-users> - the user group, for questions about using **ZAP**

<http://groups.google.com/group/zaproxy-develop> - the developer group, for questions about enhancing **ZAP**

All of these resources can be accessed via the **ZAP** Online menu.

You can also follow [@zaproxy](#) on twitter for official **ZAP** announcements.