Wallarm FAST quick evaluation guide

Version 2

Introduction

Wallarm FAST is a tool for automatically generating and running security tests. It is well suited for security researchers in enterprise Red Teams as well as for teams in charge of test automation in CI/CD environments. The main goal of the tool is to help significantly increase security test coverage and to use Wallarm learned application context to help find application stack vulnerabilities and exploits both in newly developed code and in third party modules. Wallarm FAST also supports a variety of API protocols, including XML, Base64 encoded, and multi-level nested protocols, which allows FAST to find issues that are overlooked by other tools.
REST API is available for integration with CI and test-automation tools.

Assumptions and prerequisites

To quickly try Wallarm FAST in your own environment, you can use our fully configured Docker container on the same machine where you execute the browser-based or HTTP tests.

You need an environment capable of running Docker—we have tested our container with Linux and Windows laptops; both work fine.

Before you start the test, you will need to create an account on Wallarm Cloud. Create an account in fast.wallarm.com/signup and note your username and password—you will need it to configure your FAST instance.

Next, you will need to note the host name or IP address of the application you are planning to test. If you are running the proxy on a machine other than your local host, you will also need the public IP address of the proxy.

If you are running the tests and the proxy on the same machine, you may want to consider using the Firefox browser for your browser tests because this browser allows you to set up the proxy manually, just for the browser (more details, also remember to set the proxy for both http and https traffic) . Alternatively, you can install TunnelSwitch for Chrome to quickly switch proxies.

If you want to test the individual HTTP requests of the APIs, you can use the *curl* utility with the *--proxy* parameter or set an environment variable (in the case of UNIX-like systems like OS X and Linux) by using the following command:

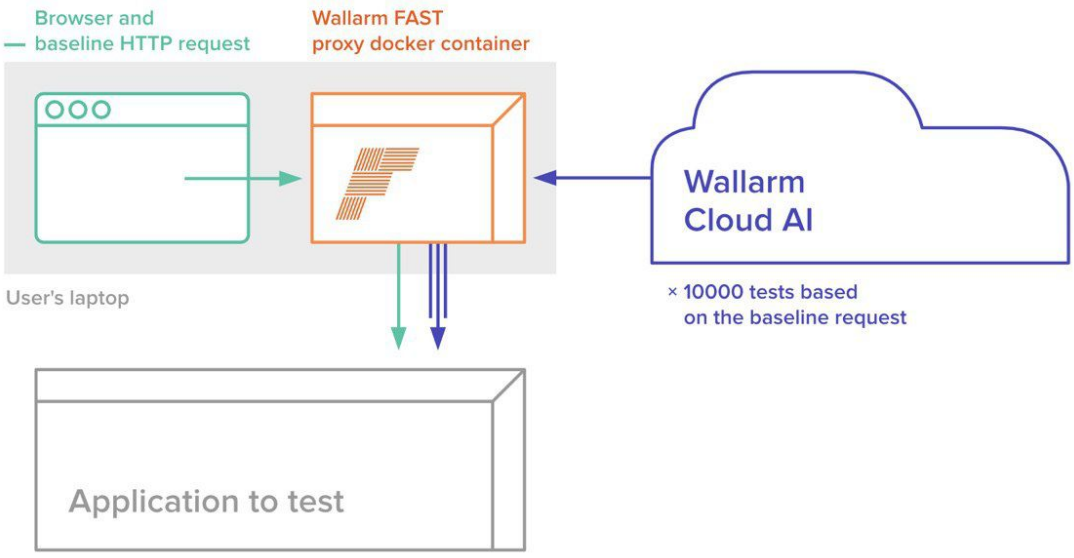
```
$ export HTTP_PROXY=http://127.0.0.1:8888
```

Just to get yourself familiar with deploying and running FAST, you may want to consider initially setting it up to test a well-known buggy application, such as http://google-gruyere.appspot.com/.

Initial deployment

In this example, we show how to deploy Wallarm FAST Proxy on a local machine to create security tests based on manually crafted HTTP/HTTPS requests. However, you can install Wallarm FAST Node inside any infrastructure; it doesn't have to live on your local machine.
The difference between Wallarm FAST Node and other Wallarm Nodes is the ability to use FAST node like an ordinary HTTP proxy (with the CONNECT method) for your browser, while a regular Wallarm node is a reverse proxy (it should be configured as a frontend with all the backends configured).

Below is a schematic installation diagram that explains the traffic flow:



NOTE: Running test sets directly from Wallarm Cloud without sending them through Wallarm FAST Proxy will only work if the application does not tie the sessions to the IP address of the client. You will need to either disable this feature in the application or stick with the FAST Proxy for traffic routing.

Test generation, execution, and terminology

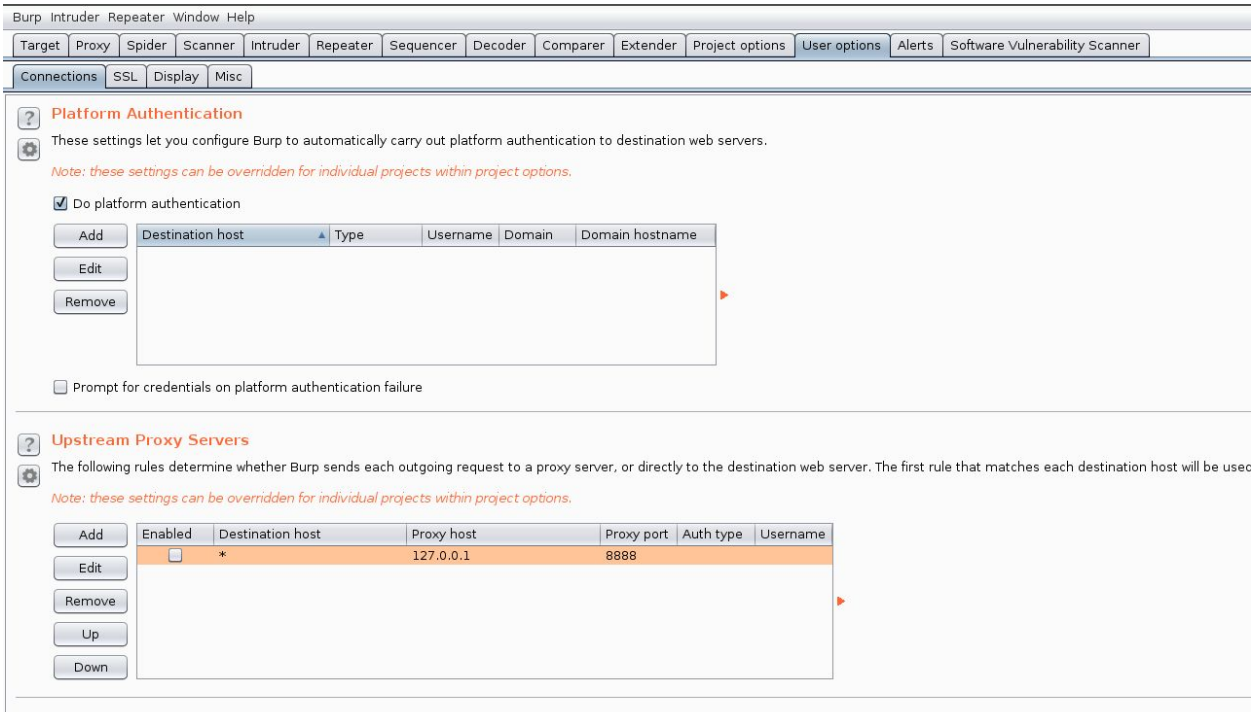
Wallarm FAST creates security tests by

1. Applying a fuzzing technique to generate payloads that are more likely to trigger an issue.
2. Using Wallarm's understanding of the application end-points to understand what should be tested and applying library payloads for known attack types.

To generate each set of tests, Wallarm takes a browser or API request to one of the application endpoints and applies test generation technique to it per the policy, including using the fuzzer to generate new payloads. The test that is used as a base for fuzzing is called the **baseline**. All the tests created on a basis of a single **baseline** are together called a **test set.** Multiple **test sets** can be executed together to achieve a more comprehensive test result. This group of tests is called a **test run**. To identify that an application request should be subject to fuzzing and that it belongs to a certain **test run**, a special header is added to this request called a "**test run ID**" (it may also be referred to as a "**test marke**r").

In our simplified setup, the **test run ID** is added by FAST Proxy. It is also possible to add this header using other techniques to achieve similar results.

For example, if you are using Burp Suite, you can configure another proxy as an upstream:



All the requests will be added to Wallarm as **baselines** in the **test run** referenced by the **test marker/test run ID**.

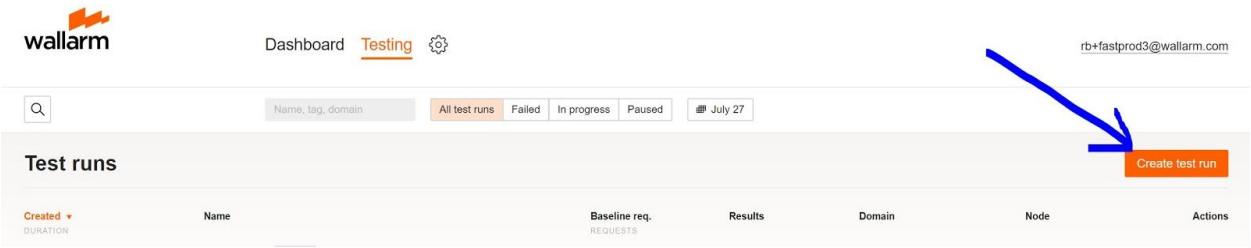
For regression tests and automation, you can also use a **tag** with your **test run** if, for example, you want to run a regression test with the same tests after a bug has been fixed.

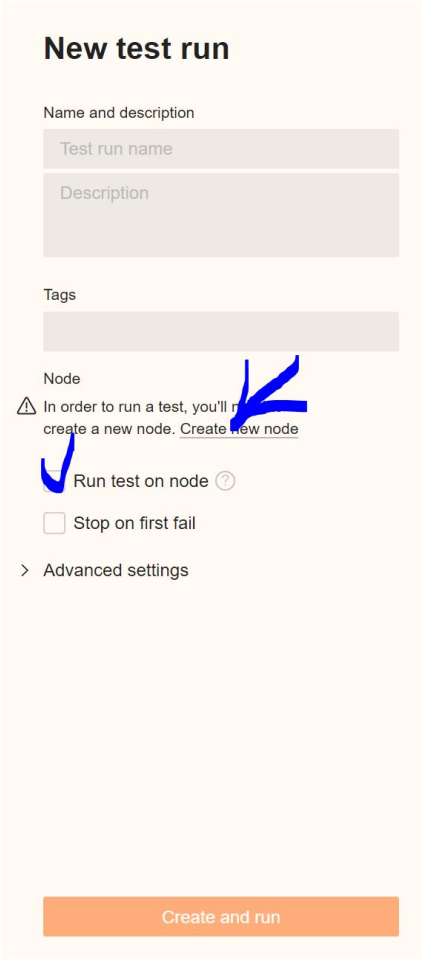
The tests are executed from Wallarm Cloud and are proxied through FAST Proxy. This means that both those applications behind the company perimeter and those applications with publicly resolvable hostnames can be tested.

Each **test set** is run in the session and authentication context of the original baseline request, so the testing is grey box rather than black box.

Creating a new test run in the Wallarm interface

Now that we understand the terminology, it's time to create a new test run to make the security testing happen. Just go to my.wallarm.com/testing and click on the orange "Create test run" button on the right





Then enter the name of **test run** and, optionally, a description and a tag. To record functional tests or baselines and to execute tests you will need a FAST node. You can select a node you have configured when you previously used FAST, or you can create a new node. When a new node is created, at the same time the system creates a *token* which you will need to connect a Docker container which actually executes the testing to your account. This *token* is automatically copied to the clipboard for you.

Make sure *Run tests on the node* option is selected to run the auto-generated tests on the same FAST node where baselines are captured.
Click *Create and Run* to create and launch the Test Run.

Your test run starts automatically right after you launch the Docker container and the first baseline request is executed (which means the test sets are generated per the fuzzing policy and immediately run) NOTE: there are alternative ways to generate and run tests with FAST using Wallarm FAST APIs and/or modifying http headers. These topics are advanced and will not be covered in this document.

Configuring and running FAST Proxy

When setting test run parameters for a Docker container running FAST Node is the *token* which connects the Docker container to your Wallarm account where your *Test Run* is defined.

Make sure your environment is configured and your Docker engine service is running.
Then run the following command from a terminal / shell window:

For Linux:

```
$docker run  -p 8888:8080 --rm --name your-docker-name -ti -e
WALLARM_API_TOKEN='base64-token-from-the-clipboard' -e
ALLOWED_HOSTS=app-to-test.yourdomain.com wallarm/fast
```

For Windows:

```
C:\>docker run  -p 8888:8080 --rm --name your-docker-name -ti -e
WALLARM_API_TOKEN="base64-token-from-the-clipboard" -e
ALLOWED_HOSTS=app-to-test.yourdomain.com wallarm/fast
```

The command options are as follows:

- -p defines proxy port options options
- --name defines the name of the container/image and should be unique among any running Docker containers on the same system
- -ti is a standard docker command for executing commands and setting options inside the container
- -e sets environmental variables inside the container
- WALLARM_API_TOKEN is a Wallarm specific environmental variable which defines a token used to connect the running Docker container with a logical Wallarm node. You get this token from the Wallarm console
- ALLOWED_HOSTS is a Wallarm specific environmental variable which defines which requests to use as baselines. Only requests targeted to the hosts listed in ALLOWED_HOSTS will be used as baselines to create test sets. Multiple hosts can be listed, column-separated. The hosts may be specified by their DNS name or by their IP addresses.
- wallarm/fast is the name of the Wallarm image in the Docker repository (used by the implicit Docker pull command.) This should not be changed.

If you intend on testing https applications, you can either choose to trust Wallarm certificate, or you can configure the Wallam proxy to install the certificates from your own environment.

In the former case it's as simple as going to http://wallarm.get/cert.der to install the Wallarm certificate into your browser.
To run the docker container with your own certificates, the docker launch commands becomes a bit more complicated and looks like this:

```
$docker run  -p 8888:8080 -v /home/path_to_certs:/tmp/inside --rm
--name your-docker-name -ti -e CA_CERT=/tmp/inside/cert
-e CA_KEY=/tmp/inside/key
-e WALLARM_API_TOKEN='base64-token-from-the-clipboard'
-e ALLOWED_HOSTS=app-to-test.yourdomain.com wallarm/fast
```

In this case `/tmp/inside/cert` is the local directory where the cert is stored; `/tmp/inside/key` is the local directory where the key is stored.

Alternatively, to make for a shorter command line, you can put put all the relevant environmental variables into a configuration file and provide it to the docker command as input as follows:

```
$docker run -p 8888:8080 --name --rm your-docker-name -e --env-file
config.file your-docker-name
```

After executing the `docker run` command you will see significant number of ongoing status messaging informing you of the container and Wallarm node operations.
You may want to open an additional terminal/shall window to execute additional commands.

Now that our Wallarm Node is running in the Docker container, all the requests that come through the proxy and match the ALLOWED_HOSTS destination will be used as baselines to generate test sets.

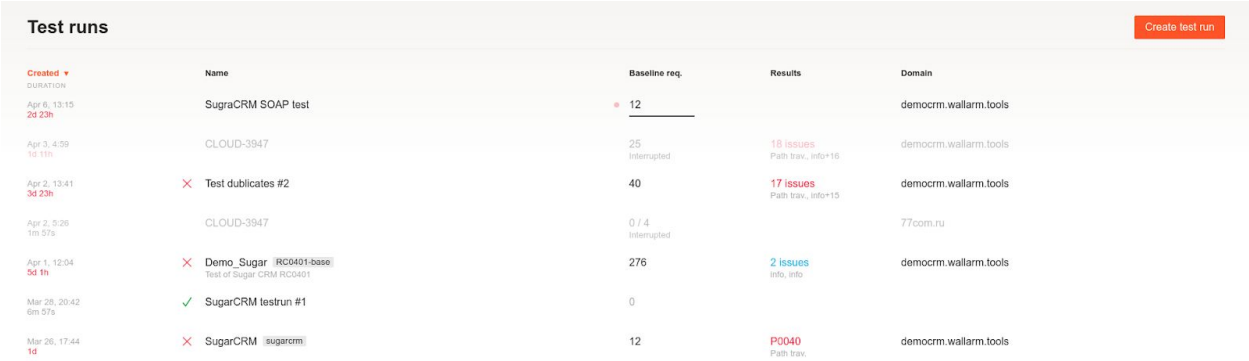
You can start creating security tests right away using the default test & fuzzing policy.
You can also customize the fuzzing policy if needed. In the example below, we have disabled all the checks for HTTP request headers and disabled the fuzzer altogether to keep only vulnerability checks, like XSS, SQLi, Path Traversal, etc., for a quick scan.

```
$ docker exec -ti fast-proxy set_policy 'type=!fuzzer;insertion=exclude:POST_(.*)'
```

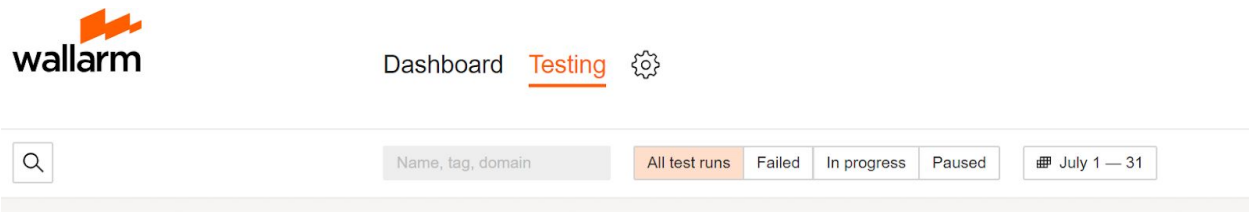
Set the proxy *localhost:8888* in your browser on both http and https ports.

Checking the results

Go to the my.wallarm.com/testing interface and take a look at your test run:

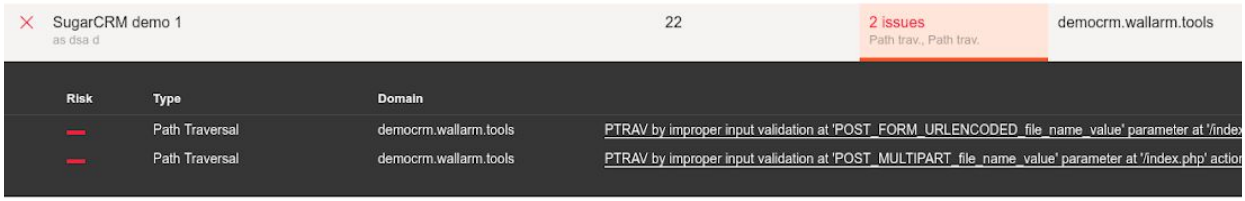


You can choose the status and dates, or use a search by tag names specified earlier when you created the test runs. By default, all the tests for today will be displayed; to filter the results by certain conditions, please use the toolbar panel at the top of the page:

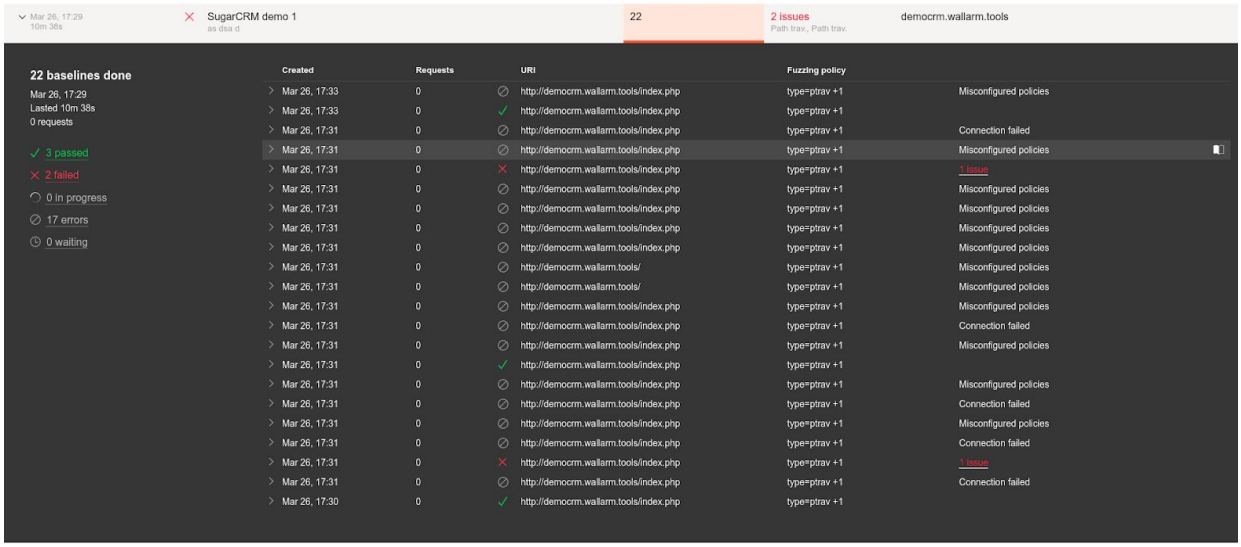


If some issues were found, they will be shown in the Results column, similar to the screenshot on the left:

Click an entry in this column to expand the list with all the issues discovered during the current test run:



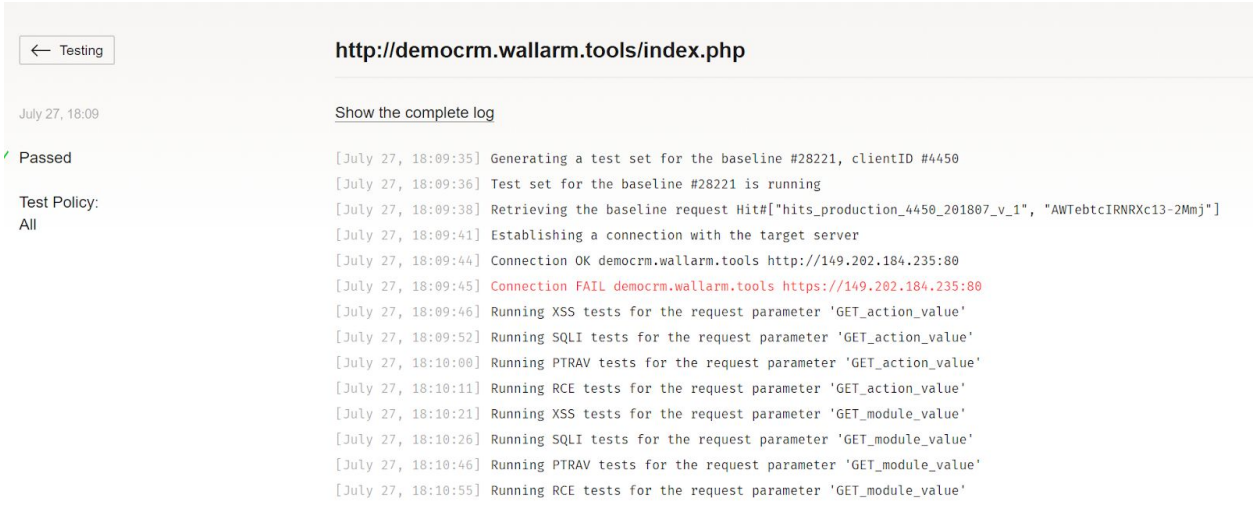
Click the vulnerability to expand review the details.
Click the “Baseline req.” column to look at the details of baselines received during this test run:



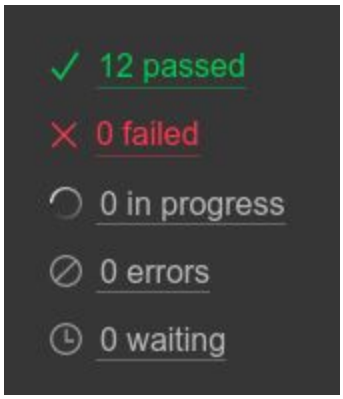
For each of the baselines , you will see the status, the number of tests generated and sent based on it, and some other information:

Created	Requests		URI	Fuzzing policy
Apr 9, 10:34	748	✓	http://democrm.wallarm.tools/service/v4_1/soap.php	type=fuzzer +1
Apr 7, 15:21	748	✓	http://democrm.wallarm.tools/service/v4_1/soap.php	type=fuzzer +1

On the very right, there is a link labeled `Details`. Clicking it will show the detailed list of operations performed for this test set, including all the security checks that were executed, similar to the screenshot below:



There are a few possible statuses for each of the test sets:



Passed means that the tests within the **test set** based on this baseline were all executed and no issues, vulnerabilities, or anomalies were found.

Failed means that one or more anomalies or vulnerabilities were detected during the execution of the **test set** based on the current baseline.

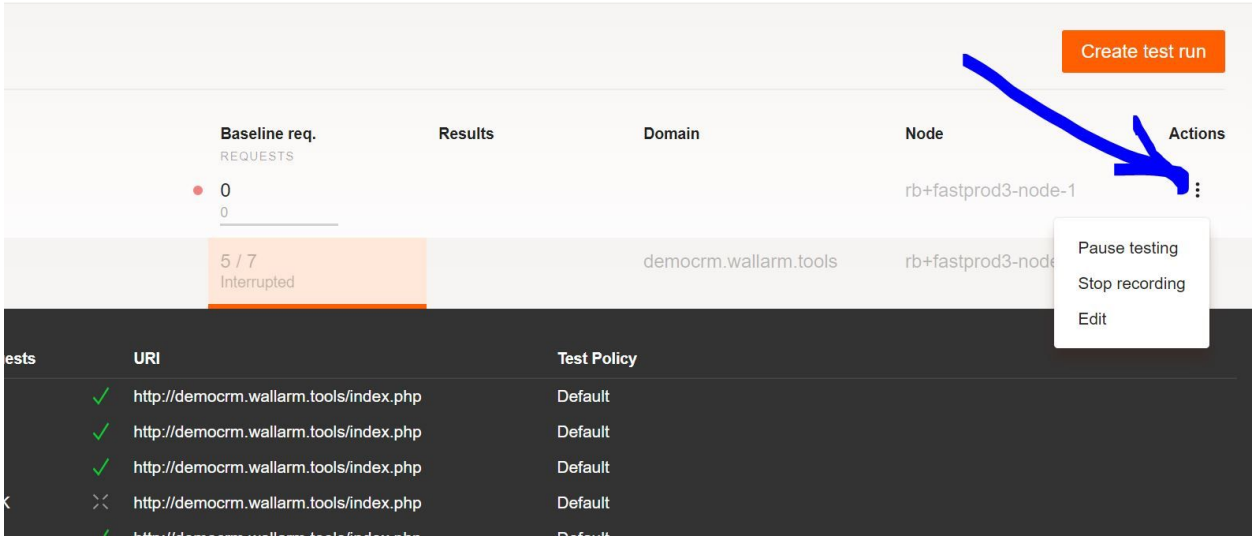
In progress is self-explanatory. This means that the **test set** is currently being executed.

Error status means that the **test set** was either not generated or its execution did not complete. The possible reasons might include a connection failure, the application under test being unresponsive, or a test policy that resulted in no tests being generated for a particular **baseline**.

Waiting status is assigned to the baselines when the **test sets** have not yet started and will execute after the previously queued **test sets** are completed.

Interrupted is also self-explanatory. It will be assigned to test sets that have experienced a hard stop or were stopped manually.

To control the operation of the current Test Run, expend the three dot menu on the top right :



To stop including any follow on baseline requests (recording of baseline requests) in the current **Test run**, select `Stop Recording`

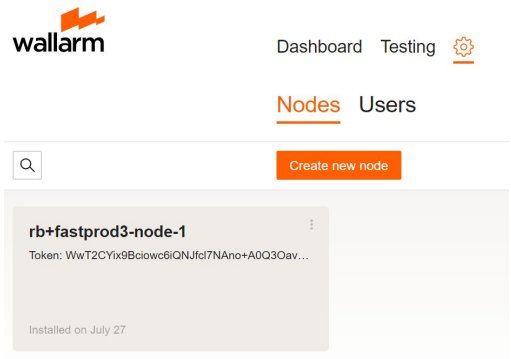
Be careful—it's impossible resume recording/adding baselines to the test run after you stop it; you will need to create a new test run instead.

To pause generating and running tests based on previously recorded baselines, select `Pause Testing`. to your application according to the current **test run,** use the pause/play button:

If you want to change some details of a **test run,** like a tag or a description, select `Edit` option.

Multiple Test Runs

When you created your first test run, in the background, Wallarm has automatically create a logical Fast Node. You can review this node under my.wallarm.com/settings



If needed, you can regenerate the authentication token from this interface.
Please, note, only one **test run** can run on a node at any time. If you create a new **test run** and select a node that is currently either recording baselines or running test sets, you will have to interrupt the execution of the previous **test run** to proceed. Alternatively, you can create a new node, which will then also appear in the settings interface.