anteater Documentation

Luke Hinds

Jan 09, 2019

Contents

This is the documentation of Anteater - CI/CD Gate Check Framework .

Anteater is an application that is run as a gate / build check within a continous Integration / DevOps deployment scenario.

It's main function is to block content based on regular expressions.

You can use it to protect against security risks, or automate a way of letting developers know that their pull request contains content that is depreciated or in some way no longer accepted by your project.

The tool can be run locally, or as a part of DevOps CI flow with systems such as *Travis CI Integration* or *CircleCI*, Jenkins etc.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Operating System Requirements

This tool is best run on a Linux distribution, it may work on Mac, but has not been tested as yet. The recommended way is using docker, as that way it will not interfere with your local systems package installations.

The main OS package requirements are listed below.

Note: If you only intend to use anteater as part of a Travis CI / CircleCI gate, then you can likely bypass this chapter which is more centered on installation for self hosted CI enviroments. See *Travis CI Integration* or *CircleCI* for setup details. details.

1.1.2 Docker

Get the latest Dockerfile (read the Dockerfile first before running!):

```
wget https://raw.githubusercontent.com/lukehinds/anteater/master/docker/Dockerfile
```

Build the Image:

```
docker build -t anteater .
```

Run an instance:

```
docker run -t -i anteater /bin/bash
```

Or to run from a job:

```
PROJECT="myrepo"
git diff --name-only HEAD^ > /tmp/patch
vols="-v /home/user/repos/$PROJECT"
docker run -i $vols ~/venv/bin/anteater --project $PROJECT --patchset /tmp/patch"
```

1.1.3 Install Anteater

The best method to install anteater, is via pip:

```
pip install anteater
```

1.1.4 Contribute

All contributions must be made as pull requests from your forked repository of anteater.

To install from source (recommend a virtualenv for isolation / non root use):

Install requirements:

```
pip install -r requirements.txt
```

Install anteater:

```
python setup.py install
```

1.2 User Guide

1.2.1 Configuration

Anteaters configuration exists witin `anteater.conf`:

```
[config]
anteater_files = anteater_files/
reports_dir = %(anteater_files)s.reports/
anteater_log = %(anteater_files)s/.reports/anteater.log
flag_list =  %(anteater_files)s/flag_list.yaml
ignore_list = %(anteater_files)s/ignore_list.yaml
vt_rate_type = public
```

- `anteater_files`: Main location to store anteater `flag_list`, `ignore_list` and reports. This location is ignored by anteater when performing scans.
- `reports_dir`: location for anteater to send reports
- `anteater_log`: anteater application logging output file.
- `flag_list`: Regular Expressions to flag. See RegExp Framework.
- `ignore_list`: Regular Expressions to overwrite / cancel `flag_list`.
- `vt_rate_type`: `public` or `private` VirusTotal API limiting.

The `anteater.conf` file should always be in the directory from where the anteater command is run from. `anteater` will look for `anteater.conf` in the present working directory.

1.2.2 Methods of Operation

Anteater uses a simple argument system in the standard POSIX format.

The main usage parameters are `--project` and either `---path` or `--patchset`.

Optional parameters are `--binaries` which is the binary check system. When this argument is passed, all binaries / blobs will result in a VirusTotal scan - unless a sha256 checksum of the binary is listed in one of the exeception files (`ignore_list` or a `project_exceptions` file. `--ips` peforms a scan of IP addresses, and `--urls` for any URL's found within file contents.

Refer to *binary exceptions* for more details on the binary blocking feature of anteater.

1.2.3 The `--project` argument

Anteater always requires a project name passed with the `--project` argument. This should be the same as the name as your repository. So for example, if your git repository and its root folder are named 'acme', then you pass `--project acme`.

Having a project parameter allows for a scenario of multiple projects (for example when using gerrit).

The `--project` parameter maps to several areas:

- Reports naming convention (for example `contents_<project>.log`)
- dealing with a relative path (we strip out the full path, to allow people to enter filenames with a path relative to the repository). This is useful for when running locally (where every user will have their own unique `$HOME`).
- project exceptions:

```
project_exceptions:
  - myrepo: anteater_files/myrepo.yaml
```

Note: See *Exceptions* for more details.

1.2.4 The `--patchset` and `--path` arguments

Anteater can be run with two methods, `--patchset` or `--path`.

When `--patchset` is passed as an argument, it is expected that a text file be provided that consists of a list of files, using a relative or full path. Anteater will then iterate scans over each file, with the files seperated by a new line. For example:

```
% cat /tmp/patchset
/path/to/repos/myrepo/fileone.sh
/path/to/repos/myrepo/filetwo.sh
/path/to/repos/myrepo/filethree.txt
```

The patchset is typically generated by another system, with git being a good example and allowing a complete pull request to be iterated over:

```
git diff --name-only HEAD^ > /tmp/patchset
```

This would then be called with:

```
anteater --project myrepo --patchset /tmp/patchset
```

When `--path` is provided, the argument should be a single relative or full path to your repositories folder. Anteater will then perform a recursive walk through all files in the respository folder. For example:

```
anteater --project myrepo --path /path/to/repos/myrepo
```

Having these two methods allows anteater to scan individual pull requests / patch sets or perform a complete audit on existing files.

RegExp Framework

The RegExp Framework is set of a YAML formatted files which are declared in `anteater.conf` under the directives `flag_list` and `ignore_list`, as well as `project_exceptions` embedded within `ignore_list`.

There is a simple hierarchy with these files, with `ignore_list` and the contents within `project_exceptions` "stacking" on top.

All RegExp files should be stored in the set location of `anteater_files` that is declared in `anteater.conf` - this is important, as `anteater_files` is ignored by anteater during all scanning operations, thereby stopping anteater falsely flagging its own strings set within `flag_list`.

1.2.5 flag_list

`flag_list` is a complete list of all regular expressions, that if matched within any file content or binary / file name, will cause anteater to exit with a sys code of `1`, thereby causing a build failure within a CI system (such as jenkins / Travis CI).

`flag_list` should be considered a list of strings or object namings that you do not want anyone to merge into a repository, a blacklist essentially. This could include security objects such as private keys, binaries or depreciated functions, modules, libaries. Basically anything that can be matched using standard regular expression syntax.

Within `flag_list` are several parameters set within YAML list formats.

1.2.6 file_names

`file_names` is a list of full file names to flag. For example, the following would flag someone's shell history if included in a pull request / patch:

```
file_audits:
    file_names:
      - (irb|plsq|mysql|bash|zsh)_history
```

So if a user then accidentally checks in a `zsh_history` then anteater will flag this, the build will fail and prevent an oversight from happening and the file being merged into main branches.

1.2.7 file_contents

`file_contents` is a list of regular expression strings that will be searched for within any file that is not a binary / blob - this could be text files, documentation, shell scripts, source code etc.

The structure of the file is as follows:

```
file_audits:
  file_contents:
    unique_name:
        regex: <Regular Expression to Match>
        desc: <Line of text to describe the rationale for flagging the string>
```

The following would be examples for ensuring no insecure cryptos are used and a depreciated function is also flagged:

```
file_contents:
  md245:
    regex: md[245]
    desc: "Insecure hashing algorithm"

  depreciated_function:
    regex: depreciated_function\(.*\)
    desc: This function was depreciated in release X, use Y function.
```

So the above would match and flag the following lines:

```
hashlib.md5(password)

dothis = thing.depreciated_function(some_value):
```

1.2.8 Exceptions

Exceptions are essentially a regular expression that provides a waiver to strings that are flagged as false postives.

Exceptions can be made in two locations `ignore_list` or `project_exceptions` set within `ignore_list` and allows you to overule a string set within the `flag_list` file with a more unique regular expression.

There are main three sections within `ignore_list.yaml` and `project_exceptions`

- `file_contents` - ignore matching regex if matched in a certain file.
- `file_names` - ignore matching regex when it matches a file name.
- `binaries` - allow binaries, when they have a matching sha256 checksum set.

1.2.9 Project Exceptions

If you're a single project, then you can place all of the above three sections into `ignore_list.yaml`. If you have to manage multiple projects, then use `ignore_list.yaml` as a global master list, and use a `project_exceptions` entry for each individual project. For example, within your `ignore_list.yaml` you can declare each projects exeception list as follows:

```
project_exceptions:
  - acme:   anteater_files/acme.yaml
  - bravo   anteater_files/bravo.yaml
  - charlie anteater_files/charlie.yaml
```

1.2.10 file_contents exceptions

`file_contents` exceptions are used to cancel out a `flag_list` entry by using a regular expression that matches a unique string that has been incorrectly flagged and is a false positive.

Let's say we wish to have some control over git repositories that can be cloned in shell scripts present in out repository and used to automate our builds.

First we make an entry in the `flag_list` around git clone:

```
file_contents:
  clone:
    regex: git.*clone
    desc: "Clone blocked as using an non approved external source"
```

The above would flag any instance of a clone, for example:

```
git clone http://github.com/no_longer_around/some_unmaintained_repo.git
```

Now let's assume we want to allow all clones from a specific github org called 'acme' which we trust, but no other github repositories.

We could do this by using the following Exception:

```
file_contents:
  - git clone https:\/\/github\.com\\acme\\.+
```

This would then allow the following strings:

```
git clone https://github.com/acme/repository
git clone https://github.com/acme/another_repository
```

Let's look at an example again using the md5 flag:

```
file_contents:
  md245:
    regex: md[245]
    desc: "Insecure hashing algorithm"
```

The above `file_contents` expression would incorrectly match the following string:

```
mystring = int(md500) * 4
```

In this case `md500` is incorrectly matched against ``md5`.

We can cancel out this false postive with a regular expression unique to the incorrectly flagged false positive:

```
file_contents:
  - mystring.=.int\(md500\).*
```

Note: You can test strings out on an regex site such as https://regex101.com

1.2.11 file_names exceptions

As with `file_contents`, `file_names` incorrectly flagged as false postives may also be removed using a regular expression.

1.2.12 Public IP Addresses

If *–ips* is passed as arguments, anteater will perform a scan for public / external IP Addresses. Once an address is found, the IP is sent to the Virus Total API and if the IP Address has past assocations with malicous or malware hosting domains, a failure is registered and a report is provided.

An example report can be seen here.

1.2.13 URLs

If `--urls` is passed as arguments, anteater will perform a scan for URL's. If an URL is found, the URL is sent to the Virus Total API which then compares the URL to a large list of URL blacklisting services.

An example report can be seen here.

1.2.14 binary exceptions

If the `--binaries` argument is passed to anteater, anteater blocks (CI build failure) all binary files unless a sha256 checksum of the file is entered as an exeception. If no checksum is present, the binary (hash) is also sent to the VirusTotal API.

This is done using the relative path from the root of the respository.

For example:

```
media/images/weather-storm.png:
  - 48f38bed00f002f22f1e61979ba258bf9006a2c4937dde152311b77fce6a3c1c
media/images/stop_light.png:
  - 5a1101e8b1796f6b40641b90643d83516e72b5b54b1fd289cf233745ec534ec9
```

Examples of files can be found **here**_. .. _here: https://github.com/anteater/tree/master/examples

1.3 Virus Total API

1.3.1 API Key

In order to use the VirusTotal API, you will first require an API key. These are free to get and can be obtained by signing up to the service here.

Once you have your key, it needs to be set as an environment variable.

If you're using CI, then see refer to the relevant CI document section in these docs for examples of how to achieve this.

If either *–ips*, `--urls` or `--bincheck` are called as arguments (in any combination including all three at once), then the VirusTotal API will be queried for information on the following:

1.3.2 Public IP Addresses

If *–ips* is passed as arguments, anteater will perform a scan for public / external IP Addresses. Once an address is found, the IP is sent to the Virus Total API and if the IP Address has past assocations with malicous or malware hosting domains, a failure is registered and a report is provided.

An example report can be seen here.

If you wish to whitelist an IP address, make an entry into your *ignore_list* or project specific ignore_list:

ip_ignore:

- '173.217.16.206'
- '92.47.16.221'

1.3.3 URLs

If `--urls` is passed as arguments, anteater will perform a scan for URL's. If an URL is found, the URL is sent to the Virus Total API which then compares the URL to a large list of URL blacklisting services.

An example report can be seen here.

If you wish to whitelist an IP address, make an entry into your *ignore_list* or project specific ignore_list:

url_ignore:

- 'http://www.apache.org'
- 'https://github.com'

1.3.4 Binaries

If `--bincheck` is passed as arguments, anteater will send a hash of the binary to the Virus Total API which then compares the binary to an aggregation of Virus Scanner results. If no existing report is available, anteater will send the complete binary file to Virus Total for a new scan.

If you wish to whitelist an binary, make an entry into your *ignore_list* or project specific ignore_list:

binaries:

path/to/example.png:

- 609feaed93afbea14c6b10c6effc986f39d1deb0a372ac088129bb22bbca8834

- Note: The sha256 checksum showed above, will be outputed in anteaters logs when it finds a binary.

1.3.5 Rate limit

Use of the public Virus Total API requires a rate limit of no more than three requests per minute, unless you have use of a private API account.

Public or Private can be set within the *anteater.conf* file, and anteater will then use the appropriate rate limit:

```
vt_rate_type = public
```

The values are `public` for the public API, and `private` for the private API.

Redis is requried for rate litmiting as means to track global rate requests.

All that is required for the Redis set up, is the installation of Redis and running redis with its default values.

The Dockerfile will deploy redis for you. Refer to **'installation'_** for more details.

1.4 Travis CI Integration

1.4.1 Set up steps

First create an `anteater.conf in the root directory of your repository:

```
[config]
anteater_files = anteater_files/
reports_dir = %(anteater_files)s.reports/
anteater_log = %(anteater_files)s/.reports/anteater.log
flag_list =  %(anteater_files)s/flag_list.yaml
ignore_list = %(anteater_files)s/ignore_list.yaml
```

1.4.2 anteater_files

anteater_files is a location which anteater **wil not** scan.

The rationale about hiding this folder from anteater, is for the simple fact anteater will report on the strings it uses itself as a guide for what to search for.

1.4.3 reports_dir & anteater_log

You can leave these as is, its a logging location used for when running the tool locally.

1.4.4 flag_list & ignore_list

flag_list.yaml is where all regular expressions are set, that if matched will fail the build, thereby marking a failure on the pull request page.

Some examples can be found here.

For information on flag_list, please consult the *User Guide*

1.4.5 Travis Integration

All that is required now is to make the following entries to your yaml file:

```
language: python

python:
  - "2.7"

install:
  - pip install anteater

before_script:
  - git diff --name-only HEAD^ > ./patch
script:
  - anteater --project antest --patchset ./patch
```

Note: Should you be using another language other then python (for example ruby), you can use matrix:include

```
matrix:
  include:
    - language: python
      python:
        - "2.7"
        - "3.6"

      install:
        - pip install anteater

      before_script:
        - git diff --name-only HEAD^ > ./patch

      script:
        - anteater --project antest --patchset ./patch

    - language: ruby
    # your project travis elements go here.
```

An example .travis.yml can be found here.

1.4.6 Virus Total API KEY

Should you wish to use any of the Virus Total functionailty such as URL scanning, then please set your Virus Total Key as the environment variable `VT_KEY` in the "Environment Variables" section of your Travis CI job, see here. for complete details.

1.4.7 Developer Workflow

1. Contributor forks , creates a branch

```
git checkout -b mypullrequest
```

2. Contributor commits and makes pull request

```
git commit -m "My Pull Request" git push origin mypullrequest
```

3. A pull request is then made on the Contributors github page.
4. Travis CI runs anteater checks, checks fail.
5. Contributor addresses the failure.

```
git commit -va -m "Correcting for anteater failures"
```

6. Travis CI runs anteater again, and marks build as Passed.
7. Main developer see's test has passed, and merges Contributors pull request.

1.5 CircleCI

1.5.1 Set up steps

First create an `anteater.conf in the root directory of your repository:

```
[config]
anteater_files = anteater_files/
reports_dir = %(anteater_files)s.reports/
anteater_log = %(anteater_files)s/.reports/anteater.log
flag_list =  %(anteater_files)s/flag_list.yaml
ignore_list = %(anteater_files)s/ignore_list.yaml
```

1.5.2 anteater_files

`anteater_files` is a location which anteater **wil not** scan.

The rationale about hiding this folder from anteater, is for the simple fact anteater will report on the strings it uses itself as a guide for what to search for.

1.5.3 reports_dir & anteater_log

You can leave these as is, its a logging location used for when running the tool locally.

1.5.4 flag_list & ignore_list

`flag_list.yaml` is where regular expressions are set, that if matched will fail the build, thereby marking a failure on the github pull request page.

More examples can be found here.

For information on `flag_list`, please consult the *User Guide*

1.5.5 CircleCI Integration

All that is required now is to make the following entries to your CircleCI configuration file `.circleci/config.yml`:

```
# Python CircleCI 2.0 configuration file
#
# Check https://circleci.com/docs/2.0/language-python/ for more details
#
version: 2
jobs:
  build:
    docker:
      # specify the version you desire here
      # use `-browsers` prefix for selenium tests, e.g. `3.6.1-browsers`
      - image: circleci/python:2.7

      # Specify service dependencies here if necessary
      # CircleCI maintains a library of pre-built images
      # documented at https://circleci.com/docs/2.0/circleci-images/
      # - image: circleci/postgres:9.4

    working_directory: ~/repo

    steps:
      - checkout
```

(continues on next page)

(continued from previous page)

```
    # Download and cache dependencies
    - restore_cache:
        keys:
        - v1-dependencies-{{ checksum "requirements.txt" }}
        # fallback to using the latest cache if no exact match is found
        - v1-dependencies-

    - run:
        name: install dependencies
        command: |
          virtualenv ~/venv
          . ~/venv/bin/activate
          pip install -r requirements.txt
          git diff --name-only HEAD^ > ~/repo/patchset

    - save_cache:
        paths:
          - ./venv
        key: v1-dependencies-{{ checksum "requirements.txt" }}

    # run tests!
    - run:
        name: run tests
        command: |
          . ~/venv/bin/activate
          anteater --project ci-circle --patchset ~/repo/patchset

    - store_artifacts:
        path: test-reports
        destination: test-reports
```

An example config.yml can be found here.

1.5.6 Virus Total API KEY

Should you wish to use any of the Virus Total functionailty such as URL scanning, then please set your Virus Total Key as the environment variable `VT_KEY` in the "Environment Variables" section of your Circle CI app, see here. for complete details.

1.5.7 Developer Workflow

1. Contributor forks , creates a branch

```
git checkout -b mypullrequest
```

2. Contributor commits and makes pull request

```
git commit -m "My Pull Request" git push origin mypullrequest
```

3. A pull request is then made on the Contributors github page.
4. Travis CI runs anteater checks, checks fail.
5. Contributor addresses the failure.

```
git commit -va -m "Correcting for anteater failures"
```

6. Travis CI runs anteater again, and marks build as Passed.
7. Main developer see's test has passed, and merges Contributors pull request.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search