
tartufo

Release 1.1.2

GoDaddy.com, LLC

May 29, 2020

MORE INFORMATION

1	Example	3
2	Quick start	5
3	Attributions	7

tartufo searches through git repositories for secrets, digging deep into commit history and branches. This is effective at finding secrets accidentally committed. *tartufo* also can be used by git pre-commit scripts to screen changes for secrets before they are committed to the repository.

This tool will go through the entire commit history of each branch, and check each diff from each commit, and check for secrets. This is both by regex and by entropy. For entropy checks, tartufo will evaluate the shannon entropy for both the base64 char set and hexadecimal char set for every blob of text greater than 20 characters comprised of those character sets in each diff. If at any point a high entropy string > 20 characters is detected, it will print to the screen.

EXAMPLE

```
Date: 2014-04-21 18:46:21
Branch: master
Commit: Removing aws keys

@@ -57,8 +57,8 @@ public class EurekaEVCacheTest extends AbstractEVCacheTest {
    //
    props.setProperty("datacenter", "cloud");
-   props.setProperty("awsAccessId", "<aws access id>");
-   props.setProperty("awsSecretKey", "<aws secret key>");
+   props.setProperty("awsAccessId", "AKIAJCK2WUJ2653GNBQ");
+   props.setProperty("awsSecretKey", "7JyrN0rk23B7bErD88eg8IfhYjAYdFJlhCbKEo6A");
    props.setProperty("appinfo.validateInstanceId", "false");

    props.setProperty("discovery.us-east-1.availabilityZones", "us-east-1c,us-east-1d,us-east-1e");
```


QUICK START

Getting started is easy!

1. Install tartufo from the [tartufo page on the Python Package Index](#), by using `pip` or using `docker` to pull the `tartufo` image from Docker Hub.

Install using `pip`:

```
$ pip install tartufo
```

Install using `docker`:

```
$ docker pull godaddy/tartufo
```

For more detail, see [Installation](#).

2. Use `tartufo` to scan your repository and find any secrets in its history!

```
# You can scan a remote git repo
$ tartufo git@github.com:my_user/my_repo.git

# Or, scan a local clone of a repo!
$ tartufo --repo-path /path/to/your/git/repo
```

```
# Scan a remote repo using docker
$ docker run --rm godaddy/tartufo git@github.com:my_user/my_repo.git

# Mount a local clone of a repo and scan it using docker!
$ docker run --rm -v "/path/to/your/git/repo:/git" godaddy/tartufo
```

For more detail on usage and options, see [usage](#).

ATTRIBUTIONS

This project was inspired by and built off of the work done by [Dylan Ayrey](#) on the [truffleHog](#) project.

3.1 Installation

You can install `tartufo` in the usual ways you would for a Python Package, or using `docker` to pull the latest `tartufo` `docker` image from Docker Hub.

Installation with `pip`:

```
$ pip install tartufo
```

Installation with `docker`:

```
$ docker pull godaddy/tartufo
```

If you would like to install the latest in-development version of `tartufo`, this can also be done with `pip`.

```
$ pip install -e git+ssh://git@github.com/godaddy/tartufo.git#egg=tartufo
```

Note: Installing the in-development version is NOT guaranteed to be stable. You will get the latest set of features and fixes, but we CAN NOT guarantee that it will always work.

3.1.1 Checking the installation

When `tartufo` is installed, it inserts an eponymous command into your path. So if everything went well, the easiest way to verify your installation is to simply run that command:

Checking the `pip` installation:

```
$ tartufo --help
```

Checking the `docker` installation:

```
$ docker run godaddy/tartufo --help
```

3.2 Features

3.2.1 Regex Checking

tartufo can scan for a pre-built list of known signatures for things such as SSH keys, EC2 credentials, etc. These scans are activated by use of the `--regex` flag on the command line. They will be reported with an issue type of `Regular Expression Match`, and the issue detail will be the name of the regular expression which was matched.

Customizing

Additional rules can be specified in a JSON file, pointed to on the command line with the `--rules` argument. The file should be in the following format:

```
{
  "RSA private key": "-----BEGIN EC PRIVATE KEY-----"
}
```

Things like subdomain enumeration, s3 bucket detection, and other useful regexes highly custom to the situation can be added.

If you would like to deactivate the default regex rules, using only your custom rule set, you can use the `--no-default-regexes` flag.

Feel free to also contribute high signal regexes upstream that you think will benefit the community. Things like Azure keys, Twilio keys, Google Compute keys, are welcome, provided a high signal regex can be constructed.

tartufo's base rule set sources from <https://github.com/dxa4481/truffleHogRegexes/blob/master/truffleHogRegexes/regexes.json>

3.2.2 High Entropy Checking

tartufo calculates the [Shannon entropy](#) of each commit, finding strings which appear to be generated from a stochastic source. In short, it looks for pieces of data which look random, as these are likely to be things such as cryptographic keys. These scans are activated by usage of the `--entropy` command line flag.

3.2.3 Limiting Scans by Path

With the `--include-paths` and `--exclude-paths` options, it is also possible to limit scanning to a subset of objects in the Git history by defining regular expressions (one per line) in a file to match the targeted object paths. To illustrate, see the example include and exclude files below:

Listing 1: include-patterns.txt

```
src/
# lines beginning with "#" are treated as comments and are ignored
gradle/
# regexes must match the entire path, but can use python's regex syntax for
# case-insensitive matching and other advanced options
(?i).*\.(properties|conf|ini|txt|y(a)?ml)$
(.*\/)?id_[rd]sa$
```

Listing 2: exclude-patterns.txt

```
(.*/)?\.classpath$
.*\.jmx$
(.*/)?test/(.*/)?resources/
```

These filter files could then be applied by:

```
tartufo --include-paths include-patterns.txt --exclude-paths exclude-patterns.txt_
↪file://path/to/my/repo.git
```

With these filters, issues found in files in the root-level `src` directory would be reported, unless they had the `.classpath` or `.jmx` extension, or if they were found in the `src/test/dev/resources/` directory, for example. Additional usage information is provided when calling `tartufo` with the `-h` or `--help` options.

These features help cut down on noise, and makes the tool easier to shove into a devops pipeline.

3.3 Usage

3.3.1 History Scan

By default, *tartufo* will scan the entire history of a git repo. The repo to be scanned can be specified in one of two ways. The first, default behavior, is by passing a git URL to *tartufo*. For example:

```
$ tartufo https://github.com/godaddy/tartufo.git
```

For docker:

```
$ docker run --rm godaddy/tartufo https://github.com/godaddy/tartufo.git
```

When used this way, *tartufo* will clone the repository to a temporary directory, scan the local clone, and then delete it.

Alternatively, if you already have a local clone, you can scan that directly without the need for the temporary clone:

```
$ tartufo --repo-path /path/to/my/repo
```

For docker, mount the local clone to the `/git` folder in the docker image:

```
$ docker run --rm -v "/path/to/my/repo:/git" godaddy/tartufo
```

When scanning private repositories, the `docker` runtime needs to have access to SSH keys for authorization. Make sure `ssh-agent` is running on your host machine and has the key added. (Verify using `ssh-add -L` on host machine).

For Docker for Linux, mount the location of `SSH_AUTH_SOCK` to a location in the docker container, and point the environment variable `SSH_AUTH_SOCK` to the same location:

```
$ docker run --rm -v "/path/to/my/repo:/git" -v $SSH_AUTH_SOCK:/agent -e SSH_AUTH_
↪SOCK=/agent godaddy/tartufo
```

If using Docker Desktop for Mac, use `/run/host-services/ssh-auth.sock` both as source and target, and point the environment variable `SSH_AUTH_SOCK` to the same location:

```
$ docker run --rm -v "/path/to/my/repo:/git" -v /run/host-services/ssh-auth.sock:/run/
↪host-services/ssh-auth.sock -e SSH_AUTH_SOCK="/run/host-services/ssh-auth.sock"_
↪godaddy/tartufo
```

(continues on next page)

3.3.2 Pre-commit

The `--pre-commit` flag instructs tartufo to scan staged, uncommitted changes in a local repository. The repository location can be specified using `--repo-path`, but it is legal to not supply a location; in this case, the caller's current working directory is assumed to be somewhere within the local clone's tree and the repository root is determined automatically.

The following example demonstrates how tartufo can be used in `.git/hooks/pre-commit` to verify that secrets will not be committed to a git repository in error:

```
#!/bin/sh

# Redirect output to stderr.
exec 1>&2

# Check for suspicious content.
tartufo --pre-commit --regex --entropy
```

Git will execute tartufo before committing any content. If problematic changes are detected, they are reported by tartufo and git aborts the commit process. Only when tartufo returns a success status (indicating no potential secrets were discovered) will git commit the staged changes.

Note that it is always possible, although not recommended, to bypass the pre-commit hook by using `git commit --no-verify`.

If you would like to automate these hooks, you can use either the `Python` or `Docker` approach to setting up tartufo as a pre-commit hook

Python pre-commit hook

Add a `.pre-commit-config.yaml` file to your repository. You can copy and paste the following to get you started:

```
- repo: https://github.com/godaddy/tartufo
  rev: master
  hooks:
    - id: tartufo
```

That's it! Now your contributors only need to run `pre-commit install --install-hooks`, and `tartufo` will automatically be run as a pre-commit hook.

Warning: You probably don't actually want to use the `master` rev. This is the active development branch for this project, and can not be guaranteed stable. Your best bet would be to choose the latest version, currently 1.1.2.

Docker pre-commit hook

Use the docker image as pre-commit hook by adding the docker run command to `.git/hooks/pre-commit`:

```
docker pull godaddy/tartufo
cat <<EOF > .git/hooks/pre-commit
docker run -t --rm -v "$PWD:/git" godaddy/tartufo --pre-commit
EOF
```

3.3.3 Temporary File Cleanup

tartufo stores the results in temporary files, which are left on disk by default, to allow inspection if problems are found. To automatically delete these files when *tartufo* completes, specify the `--cleanup` flag:

```
tartufo --cleanup
```

3.4 Configuration

tartufo has a number of configuration options to customize its operation to your specific needs. These various options can be specified both on the command line, and in a configuration file, based on your needs.

3.4.1 Command Line

The basic usage of the command can be seen via the `-h` or `--help` command line switch, as seen here:

```
$ tartufo --help
Usage: tartufo [OPTIONS] [GIT_URL]

Find secrets hidden in the depths of git.

Tartufo will, by default, scan the entire history of a git repository for
any text which looks like a secret, password, credential, etc. It can also
be made to work in pre-commit mode, for scanning blobs of text as a pre-
commit hook.

Options:
  --json / --no-json           Output in JSON format.
  --rules FILENAME             Path(s) to regex rules json list file(s).
  --default-regexes / --no-default-regexes
                                Whether to include the default regex list
                                when configuring search patterns. Only
                                applicable if --rules is also specified.
                                [default: --default-regexes]
  --entropy / --no-entropy     Enable entropy checks. [default: True]
  --regex / --no-regex         Enable high signal regexes checks. [default:
                                False]
  --since-commit TEXT          Only scan from a given commit hash.
  --max-depth INTEGER          The max commit depth to go back when
                                searching for secrets. [default: 1000000]
  --branch TEXT                Specify a branch name to scan only that
                                branch.
  -i, --include-paths FILENAME File with regular expressions (one per
```

(continues on next page)

(continued from previous page)

	line), at least one of which must match a Git object path in order for it to be scanned; lines starting with '#' are treated as comments and are ignored. If empty or not provided (default), all Git object paths are included unless otherwise excluded via the <code>--exclude-paths</code> option.
<code>-x, --exclude-paths FILENAME</code>	File with regular expressions (one per line), none of which may match a Git object path in order for it to be scanned; lines starting with '#' are treated as comments and are ignored. If empty or not provided (default), no Git object paths are excluded unless effectively excluded via the <code>--include-paths</code> option.
<code>--repo-path DIRECTORY</code>	Path to <code>local</code> repo clone. If provided, <code>git_url</code> will not be used.
<code>--cleanup / --no-cleanup</code>	Clean up all temporary result files. [default: False]
<code>--pre-commit</code>	Scan staged files in <code>local</code> repo clone.
<code>--git-rules-repo TEXT</code>	A file path, or git URL, pointing to a git repository containing regex rules to be used for scanning. By default, all <code>.json</code> files will be loaded from the root of that repository. <code>--git-rules-files</code> can be used to override this behavior and load specific files.
<code>--git-rules-files TEXT</code>	Used in conjunction with <code>--git-rules-repo</code> , specify glob-style patterns for files from which to load the regex rules. Can be specified multiple times.
<code>--config FILE</code>	Read configuration from specified file. [default: <code>pyproject.toml</code>]
<code>-h, --help</code>	Show this message and exit.

3.4.2 Configuration via File

tartufo looks for configuration in two files in your current directory: `tartufo.toml`, and `pyproject.toml`. The latter is searched for as a matter of convenience for Python projects, such as *tartufo* itself. Within these files, *tartufo* will search for a section labeled `[tool.tartufo]` for its configuration.

This file should be written in the **TOML** format. All command line options can be specified in the configuration file, with or without the leading dashes, and using either dashes or underscores for word separators. When the configuration is read in, this will all be normalized automatically. For example, the configuration for *tartufo* itself looks like this:

```
[tool.tartufo]
repo-path = "."
json = false
cleanup = true
regex = true
entropy = true
```

Note that all options specified in a configuration file are treated as defaults, and will be overridden by any options specified on the command line.

3.5 Contributing

Everyone is welcome to contribute to GoDaddy's Open Source Software. Contributing doesn't just mean submitting pull requests. You can also get involved by reporting/triaging bugs, or participating in discussions on the evolution of each project.

No matter how you want to get involved, we ask that you first learn what's expected of anyone who participates in the project by reading these Contribution Guidelines.

Please Note: GitHub is for bug reports and contributions primarily - if you have a support question head over to [GoDaddy's Open Source Software Slack](#).

3.5.1 Answering Questions

One of the most important and immediate ways you can support this project is to answer questions on [Slack](#) or [Github](#). Whether you're helping a newcomer understand a feature or troubleshooting an edge case with a seasoned developer, your knowledge and experience with Python or security can go a long way to help others.

3.5.2 Reporting Bugs

Do not report potential security vulnerabilities here. Refer to [our security policy](#) for more details about the process of reporting security vulnerabilities.

Before submitting a ticket, please be sure to have a simple replication of the behavior. If the issue is isolated to one of the dependencies of this project, please create a Github issue in that project. All dependencies are open source software and can be easily found through [PyPI](#).

Submit a ticket for your issue, assuming one does not already exist:

- Create it on our [Issue Tracker](#)
- Clearly describe the issue by following the template layout
 - Make sure to include steps to reproduce the bug.
 - A reproducible (unit) test could be helpful in solving the bug.
 - Describe the environment that (re)produced the problem.

For a bug to be actionable, it needs to be reproducible. If you or contributors can't reproduce the bug, try to figure out why. Please take care to stay involved in discussions around solving the problem.

3.5.3 Triaging bugs or contributing code

If you're triaging a bug, try to reduce it. Once a bug can be reproduced, reduce it to the smallest amount of code possible. Reasoning about a sample or unit test that reproduces a bug in just a few lines of code is easier than reasoning about a longer sample.

From a practical perspective, contributions are as simple as:

- Forking the repository on GitHub.
- Making changes to your forked repository.
- When committing, reference your issue (if present) and include a note about the fix.
- If possible, and if applicable, please also add/update unit tests for your changes.
- Push the changes to your fork and submit a pull request to the 'master' branch of the projects' repository.

If you are interested in making a large change and feel unsure about its overall effect, please make sure to first discuss the change and reach a consensus with core contributors through [slack](#). Then ask about the best way to go about making the change.

3.5.4 Code Review

Any open source project relies heavily on code review to improve software quality:

All significant changes, by all developers, must be reviewed before they are committed to the repository. Code reviews are conducted on GitHub through comments on pull requests or commits. The developer responsible for a code change is also responsible for making all necessary review-related changes.

Sometimes code reviews will take longer than you would hope for, especially for larger features. Here are some accepted ways to speed up review times for your patches:

- Review other people's changes. If you help out, others will be more willing to do the same for you. Good will is our currency.
- Split your change into multiple smaller changes. The smaller your change, the higher the probability that somebody will take a quick look at it.
- Ping the change on [slack](#). If it is urgent, provide reasons why it is important to get this change landed. Remember that you're asking for valuable time from other professional developers.

Note that anyone is welcome to review and give feedback on a change, but only people with commit access to the repository can approve it.

3.5.5 Attribution of Changes

When contributors submit a change to this project, after that change is approved, other developers with commit access may commit it for the author. When doing so, it is important to retain correct attribution of the contribution. Generally speaking, Git handles attribution automatically.

3.5.6 Writing Code

Setting Up A Development Environment

This project uses [Poetry](#) to manage its dependencies and do a lot of the heavy lifting. This includes managing development environments! If you are not familiar with this tool, we highly recommend checking out [their docs](#) to get used to the basic usage.

Now, setting up a development environment is super simple!

Step 1: [Install Poetry](#) Step 2: Run `poetry install`

Done!

Code Style

To make code formatting easy on developers, and to simplify the conversation around pull request reviews, this project has adopted the [black](#) code formatter. This formatter must be run against any new code written for this project. The advantage is, you no longer have to think about how your code is styled; it's all handled for you!

To make this easier on you, you can [set up most editors](#) to auto-run [black](#) for you. We have also set up a [pre-commit](#) hook to run automatically on every commit, which is detailed below!

3.5.7 Running tests

This project support multiple Python versions. Thus, we ask that you use the [tox](#) tool to test against them. In conjunction with poetry, this will look something like:

```
$ poetry run tox
.package recreate: /home/username/tartufo/.tox/.package
.package installdeps: poetry>=0.12
...
py35: commands succeeded
py36: commands succeeded
py37: commands succeeded
py38: commands succeeded
pypy3: ignored failed command
black: commands succeeded
mypy: commands succeeded
pylint: commands succeeded
vulture: commands succeeded
docs: commands succeeded
congratulations :)
$
```

If you do not have all the supported Python versions, that's perfectly okay. They will all be tested against by our CI process. But keep in mind that this may delay the adoption of your contribution, if those tests don't all pass.

Finally, this project uses multiple [pre-commit](#) hooks to help ensure our code quality. If you have followed the instructions above for setting up your virtual environment, [pre-commit](#) will already be installed, and you only need to run the following:

```
$ pre-commit install --install-hooks
pre-commit installed at .git/hooks/pre-commit
$
```

Now, any time you make a new commit to the repository, you will see something like the following:

```
Tartufo.....Passed
mypy.....Passed
black.....Passed
pylint.....Passed
```

3.5.8 Additional Resources

- [General GitHub Documentation](#)
- [GitHub Pull Request documentation](#)

3.6 Reporting Security Issues

We take security very seriously at GoDaddy. We appreciate your efforts to responsibly disclose your findings, and will make every effort to acknowledge your contributions.

3.6.1 Where should I report security issues?

In order to give the community time to respond and upgrade, we strongly urge you report all security issues privately.

To report a security issue in one of our Open Source projects email us directly at **oss@godaddy.com** and include the word “SECURITY” in the subject line.

This mail is delivered to our Open Source Security team.

After the initial reply to your report, the team will keep you informed of the progress being made towards a fix and announcement, and may ask for additional information or guidance.

3.7 Project History

3.7.1 v1.1.2 - 21 April 2020

- #48 (Backport of #45 & #46) * Documented Docker usage * Small fixes to Docker to allow SSH clones and avoid scanning tartufo itself
- Docs have been backported from the *master* branch.

3.7.2 v1.1.1 - 13 December 2019

- Fix the docs and pre-commit hook to use hyphens in CLI arguments, as opposed to underscores.

3.7.3 v1.1.0 - 27 November 2019

- Support reading config from *tartufo.toml* for non-Python projects
- #17 - A separate repository can be used for storing rules files
- #18 - Read the *pyproject.toml* or *tartufo.toml* from the repo being scanned

3.7.4 v1.0.2 - 19 November 2019

This release is essentially the same as the v1.0.0 release, but with a new number. Unfortunately, we had historical releases versioned as v1.0.0 and v1.0.1. Due to limitations in PyPI (<https://pypi.org/help/#file-name-reuse>), even if a previous release has been deleted, the version number may not be reused.

3.7.5 v1.0.0 - 19 November 2019

Version 1.0.0! Initial stable release!

- Finished the “hard fork” process, so that our project is now independent of *truffleHog*.
- #13 - Tests are now split into multiple files/classes
- #14 - *tartufo* is now configurable via *pyproject.toml*
- #15 - Code is fully type annotated
- #16 - Fully fleshed out “Community Health” files
- #20 - Code is now fully formatted by *black*

3.7.6 v0.0.2 - 23 October 2019

Automated Docker builds!

- Docker images are built and pushed automatically to <https://hub.docker.com/r/godaddy/tartufo>
- The version of these images has been synchronized with the Python version via the VERSION file
- Gave the Python package a more verbose long description for PyPi, straight from the README.

3.7.7 v0.0.1 - 23 October 2019

This is the first public release of *tartufo*, which has been forked off from *truffleHog*.

The primary new features/bugfixes include:

- Renamed everything to *tartufo*
- #1 - Additive whitelist/blacklist support
- #4 - *--pre_commit* support
- #6 - Documented the *-cleanup* switch which cleans up files in */tmp*
- #10 - Running *tartufo* with no arguments would produce an error
- Added support for <https://pre-commit.com/> style hooks