tartufo

Release 4.0.0

GoDaddy.com, LLC

Jan 17, 2023

TABLE OF CONTENTS

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 hexidecimal 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.

CHAPTER

ONE

EXAMPLE

```
Reason: High Entropy
Filepath: tests.py
Signature: 978dc8605ef9bf471e467dd2d8570fd658f0a5f25378799a2ad8bb6ea480173d
Commit time: 2017-09-28 23:59:47
Commit message: adding some tests

Commit hash: e986c5ed6a0f0b357953c854bdc56f16914eb2ba
Branch: origin/master
@@ -0,0 +1,26 @@
+import unittest
+import os
+from truffleHog import truffleHog
+
+
+class TestStringMethods(unittest.TestCase):
+
+    def test_shannon(self):
+        random_stringB64 = "ZWVTjPQSdhwRgl204Hc51YCsritMIzn8B=/p9UyeX7xu6KkAGqfm3FJ+oObLDNEva"
+        random_stringHex = "b3A0a1FDfe86dcCE945B72"
+        self.assertGreater(truffleHog.shannon_entropy(random_stringB64, truffleHog.BASE64_CHARS), 4.5)
+        self.assertGreater(truffleHog.shannon_entropy(random_stringHex, truffleHog.HEX_CHARS), 3)
```

CHAPTER

TWO

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 scan-remote-repo git@github.com:my_user/my_repo.git

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

   ```
   # Scan a remote repo using docker
   $ docker run --rm godaddy/tartufo scan-remote-repo https://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 scan-local-repo /
   ↪git
   ```

   For more detail on usage and options, see *Usage* and *Features*.

CHAPTER

THREE

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 Usage

3.2.1 tartufo

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.

```
tartufo [OPTIONS] COMMAND [ARGS]...
```

Options

--default-regexes, --no-default-regexes

Whether to include the default regex list when configuring search patterns. Only applicable if –rules is also specified.

Default
True

--entropy, --no-entropy

Enable entropy checks.

Default
True

--regex, --no-regex

Enable high signal regexes checks.

Default
True

--scan-filenames, --no-scan-filenames

Check the names of files being scanned as well as their contents.

Default
True

-of, --output-format <output_format>

Specify the format in which the output needs to be generated *–output-format json/compact/text*. Either *json*, *compact* or *text* can be specified. If not provided (default) the output will be generated in *text* format.

Options
json | compact | text | report

-od, --output-dir <output_dir>

If specified, all issues will be written out as individual JSON files to a uniquely named directory under this one. This will help with keeping the results of individual runs of tartufo separated.

-td, --temp-dir <temp_dir>

If specified, temporary files will be written to the specified path

--buffer-size <buffer_size>

Maximum number of issue to buffer in memory before shifting to temporary file buffering

Default
10000

--git-rules-repo <git_rules_repo>

A file path, or git URL, pointing to a git repository containing regex rules to be used for scanning. By default, all .json files will be loaded from the root of that repository. –git-rules-files can be used to override this behavior and load specific files.

--git-rules-files <git_rules_files>

Used in conjunction with –git-rules-repo, specify glob-style patterns for files from which to load the regex rules. Can be specified multiple times.

--config <config>

Read configuration from specified file. [default: tartufo.toml]

-q, --quiet, --no-quiet

Quiet mode. No outputs are reported if the scan is successful and doesn't find any issues

-v, --verbose

Display more verbose output. Specifying this option multiple times will incrementally increase the amount of output.

--log-timestamps, --no-log-timestamps

Enable or disable timestamps in logging messages.

Default
`True`

--entropy-sensitivity <entropy_sensitivity>

Modify entropy detection sensitivity. This is expressed as on a scale of 0 to 100, where 0 means "totally nonrandom" and 100 means "totally random". Decreasing the scanner's sensitivity increases the likelihood that a given string will be identified as suspicious.

Default
`75`

-V, --version

Show the version and exit.

pre-commit

Scan staged changes in a pre-commit hook.

```
tartufo pre-commit [OPTIONS]
```

Options

--include-submodules, --exclude-submodules

Controls whether the contents of git submodules are scanned

Default
`False`

scan-folder

Scan a folder.

```
tartufo scan-folder [OPTIONS] TARGET
```

Options

--recurse, --no-recurse

Recurse and scan the entire folder

Default
`True`

--git-check, --no-git-check

Skip check if the folder is a git repo

Default
`True`

Arguments

TARGET

Required argument

scan-local-repo

Scan a repository already cloned to your local system.

```
tartufo scan-local-repo [OPTIONS] REPO_PATH
```

Options

--branch <branch>

Specify a branch name to scan only that branch.

--include-submodules, --exclude-submodules

Controls whether the contents of git submodules are scanned

Default
`False`

-p, --progress

Controls whether to display a progress bar

Default
`False`

Arguments

REPO_PATH

Required argument

scan-remote-repo

Automatically clone and scan a remote git repository.

```
tartufo scan-remote-repo [OPTIONS] GIT_URL
```

Options

--branch <branch>

Specify a branch name to scan only that branch.

-wd, --work-dir <work_dir>

Specify a working directory; this is where the repository will be cloned to before scanning.

--include-submodules, --exclude-submodules

Controls whether the contents of git submodules are scanned

Default
False

-p, --progress

Controls whether to display a progress bar

Default
False

Arguments

GIT_URL

Required argument

update-signatures

Update deprecated signatures for a local repository.

```
tartufo update-signatures [OPTIONS] REPO_PATH
```

Options

--branch <branch>
Specify a branch name to scan only that branch.

--include-submodules, --exclude-submodules
Controls whether the contents of git submodules are scanned

Default
`False`

--update-configuration, --no-update-configuration
Whether or not to overwrite the tartufo config file.

Default
`True`

--remove-duplicates, --no-remove-duplicates
Whether or not to remove duplicated signatures.

Default
`True`

Arguments

REPO_PATH
Required argument

3.3 Features

3.3.1 Modes of Operation

While `tartufo` started its life with one primary mode of operation, scanning the history of a git repository, it has grown other time to have a number of additional uses and modes of operation. These are all invoked via different sub-commands of `tartufo`.

Git Repository History Scan

This is the "classic" use case for `tartufo`: Scanning the history of a git repository. There are two ways to invoke this functionality, depending if you are scanning a repository which you already have cloned locally, or one on a remote system.

Scanning a Local Repository

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

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

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

Note: If you are using `podman` in place of `docker`, you will need to add the `--privileged` flag to the `run` command, in order to avoid a permission denied error.

Scanning a Remote Repository

```
$ tartufo scan-remote-repo https://github.com/godaddy/tartufo.git
```

To use `docker`:

```
$ docker run --rm godaddy/tartufo scan-remote-repo 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.

Displaying Scan Progress

When running any Git history scan, you can show scan progress by using the `--progress` or `-p` flag.

```
$ tartufo scan-local-repo /path/to/my/repo --progress
```

```
 Scanning master (1 of 59)[17942]  [#----------------------------------]    4%  00:01:26

Legend:
  master   = current branch being scanned
  1 of 59  = number of branches completed (plus current branch) and total number of␣
↪branches
  17942    = number of commits in current branch to process
  4%       = percentage of commits on current branch completed
  00:01:26 = estimated time to complete current branch
```

Accessing Repositories via SSH from Docker

When accessing repositories via SSH, the `docker` runtime needs to have access to your SSH keys for authorization. To allow this, make sure `ssh-agent` is running on your host machine and has the key added. You can verify this by running `ssh-add -L` on your host machine. You then need to point Docker at that running SSH agent.

Using Docker for Linux, that will look something like this:

```
$ docker run --rm -v "/path/to/my/repo:/git" \
  -v $SSH_AUTH_SOCK:/agent -e SSH_AUTH_SOCK=/agent \
  godaddy/tartufo scan-local-repo /git
```

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

Scanning a Folder

Operating in this mode, tartufo scans the files in a local folder, rather than operating on git commit history. This is ideal for locating secrets in the latest version of source files, or files not in source control.

```
$ tartufo scan-folder .
```

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

Note: If you are using `podman` in place of `docker`, you will need to add the `--privileged` flag to the `run` command, in order to avoid a permission denied error.

This will scan all files and folders in the specified directory including .git and any other files that may not be in source control. Perform a git clean or use a fresh clone of the repository before running scanning a folder and add `.git` to the `exclude-paths`.

Pre-commit Hook

This mode of operation instructs tartufo to scan staged, uncommitted changes in a local repository. This is the flip-side of the primary mode of operation. Instead of checking for secrets you have already checked in, this helps prevent you from committing new secrets!

When running this sub-command, the caller's current working directory is assumed to be somewhere within the local clone's tree and the repository root is determined automatically.

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

Manual Setup

To set up a pre-commit hook for `tartufo` by hand, you can place the following in a `.git/hooks/pre-commit` file inside your local repository clone:

Executing tartufo Directly

```
#!/bin/sh

# Redirect output to stderr.
exec 1>&2

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

Or, Using Docker

```
#!/bin/sh

# Redirect output to stderr.
exec 1>&2

# Check for suspicious content.
docker run -t --rm -v "$PWD:/git" godaddy/tartufo pre-commit
```

Git will execute `tartufo` before actually committing any of your changes. If any problems 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.

Using the "pre-commit" tool

New in version 2.0.0.

If you want a slightly more automated approach which can be more easily shared to ensure a unified setup across all developer's systems, you can use the wonderful pre-commit tool.

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

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

Warning: You probably don't actually want to use the *main* 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 4.0.0.

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

3.3.2 Scan Types

`tartufo` offers multiple types of scans, each of which can be optionally enabled or disabled, while looking through its target for secrets.

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 as described in the *Rule Patterns* section of the *Configuration* document.

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 can be found in the file `data/default_regexes.json`.

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.

3.3.3 Scan Limiting (Exclusions)

By its very nature, especially when it comes to high entropy scans, `tartufo` can encounter a number of false positives. Whether those are things like links to git commit hashes, tokens/passwords used for tests, or any other variety of thing, there needs to be a way to tell `tartufo` to ignore those things, and not report them out as issues. For this reason, we provide multiple methods for excluding these items.

Excluding Submodule Paths

New in version 2.7.0.

By default, any path in the repository specified as a submodule will be excluded from scans. Since these are upstream repositories over which you may not have direct control, `tartufo` will not hold you accountable for the secrets in those. If you want to include these in your scans, you can specify the `--include-submodules` option.

```
> tartufo ... --include-submodules
```

Entropy Limiting

New in version 2.5.0.

If you find that you are getting a high number of false positives from entropy scanning, you can configure highly granular exclusions to these findings as described in the *Entropy Exclusion Patterns* section of the *Configuration* document.

Limiting by Signature

New in version 2.0.0.

Every time an issue is found during a scan, `tartufo` will generate a "signature" for that issue. This is a stable hash generated from the filename and the actual string that was identified as being an issue. You can configure highly granular exclusions to these signatures as described in the *Excluding Signatures* section of the *Configuration* document.

Limiting Scans by Path

New in version 2.5.0.

By default `tartufo` will scan all objects tracked by Git. You can limit scanning by either including fewer paths or excluding some of them. You can configure these paths as described in the *Limiting Scans by Path* section of the *Configuration* document.

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.

Would you like to know more?

3.4 Configuration

`tartufo` has a wide variety of options to customize its operation available *on the command line*. Some of these options, however, can be a bit unwieldy and lead to an overly cumbersome command. It also becomes difficult to reliably reproduce the same command in all environments when done this way.

To help with these problems, `tartufo` can also be configured by way of a configuration file! You can tell tartufo what config file to use, or, it will automatically discover one for you. Starting in the current working directory, and traversing backward up the directory tree, it will search for both a `tartufo.toml` and a `pyproject.toml`. The latter is searched for as a matter of convenience for Python projects, such as `tartufo` itself. For an example of the tree traversal, let's say you running `tartufo` from the directory `/home/my_user/projects/my_project`. `tartufo` will look for the configuration files first in this directory, then in `/home/my_user/projects/`, then in `/home/my_user`, etc.

Within these files, `tartufo` will look for a section labeled `[tool.tartufo]` to find its configuration, and will load all items from there just as though they had been specified on the command line. This file must be written in the TOML format, which should look mostly familiar if you have dealt with any other configuration file format before.

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 = "."
regex = true
entropy = true
exclude-path-patterns = [
 {path-pattern = 'poetry\.lock'},
 {path-pattern = 'pyproject\.toml'},
 # To not have to escape `\` in regexes, use single quoted
 # TOML 'literal strings'
 {path-pattern = 'docs/source/(.*)\.rst'},
]
exclude-signatures = [
```

(continues on next page)

(continued from previous page)

```
    {signature = "62f22e4500140a6ed959a6143c52b0e81c74e7491081292fce733de4ec574542"},
    {signature = "ecbbe1edd6373c7e2b88b65d24d7fe84610faafd1bc2cf6ae35b43a77183e80b"},
]
```

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.

For a full list of available command line options, check out the *Usage* document.

3.4.1 Excluding Signatures

You might see the following header in the output for an issue:

```
Reason: Regular Expression Match
Detail: Generic Secret
Filepath: tests/test_scanner.py
Signature: 2a3cb329b81351e357b09f1b97323ff726e72bd5ff8427c9295e6ef68226e1d1
Branch: origin/master
Date: 2020-05-26 18:41:44
Hash: 70e59852f95f63dfa1e77822469d5414e5925716
Commit: Fix tests and shift more code away from the cli module
```

Looking at this information, it's clear that this issue was found in a test file, and it's probably okay. Of course, you will want to look at the actual body of what was found and determine that for yourself. But let's say that this really is okay, and we want tell `tartufo` to ignore this issue in future scans. To do this, you can add it to your config file.

```
[tool.tartufo]
exclude-signatures = [
  "2a3cb329b81351e357b09f1b97323ff726e72bd5ff8427c9295e6ef68226e1d1",
]
```

Done! This particular issue will no longer show up in your scan results.

As of version 3.0, a new format for specifying exclusion signatures has been added.

```
[tool.tartufo]
exclude-signatures = [
    {signature = "2a3cb329b81351e357b09f1b97323ff726e72bd5ff8427c9295e6ef68226e1d1",␣
↪reason = "reason for exclusion"},
]
```

Note: Currently both formats of signature exclusions are supported. However, only TOML array of tables format will be supported in future versions.

3.4.2 Limiting Scans by Path

You can include or exclude paths for scanning using Python Regular Expressions (regex) and the *–include-path-patterns* and *–exclude-path-patterns* options.

> **Warning:** Using include patterns is more dangerous, since it's easy to miss the creation of new secrets if future files don't match an existing include rule. We recommend only using fine-grained exclude patterns instead.

```
[tool.tartufo]
include-path-patterns = [
   'src/',
   'gradle/',
   # regexes must match the entire path, but can use python's regex syntax
   # for case-insensitive matching and other advanced options
   '(.*/)?id_[rd]sa$',
   # Single quoted strings in TOML don't require escapes for `\` in regexes
   '(?i).*\.(properties|conf|ini|txt|y(a)?ml)$',
]
exclude-path-patterns = [
   '(.*/)?\.classpath$',
   '.*\.jmx$',
   '(.*/)?test/(.*/)?resources/',
]
```

The filter expressions can also be specified as command line arguments. Patterns specified like this are merged with any patterns specified in the config file:

```
> tartufo \
  --include-path-patterns 'src/' -ip 'gradle/' \
  --exclude-path-patterns '(.*/)?\.classpath$' -xp '.*\.jmx$' \
  scan-local-repo file://path/to/my/repo.git
```

As of version 3.0, a new format for specifying paths has been added.

```
[tool.tartufo]
include-path-patterns = [
   {path-pattern = 'src/', reason='reason for inclusion'},
]
exclude-path-patterns = [
   {path-pattern = 'poetry\.lock', reason='reason for exclusion'},
]
```

> **Note:** Currently all 3 formats are supported. However, only TOML array of tables format will be supported in future versions.

3.4.3 Configuration File Exclusive Options

New in version 3.0.

As of version 3.0, we have added several configuration options which are available only in the configuration file. This is due to the nature of their construction, and the fact that they would be exceedingly difficult to represent on the command line.

Rule Patterns

New in version 3.0.

`tartufo` comes bundled with a number of regular expression rules that it will check your code for by default. If you would like to scan for additional regular expressions, you may add them to your configuration with the `rule-patterns` directive. This directive utilizes a TOML array of tables, and thus can take one of two forms:

Option 1: Keeping it contained in your `[tool.tartufo]` table.

```
[tool.tartufo]
rule-patterns = [
    {reason = "RSA private key 2", pattern = "-----BEGIN EC PRIVATE KEY-----"},
    {reason = "Null characters in GitHub Workflows", pattern = '\0', path-pattern = '\.
↪github/workflows/(.*)\.yml'}
]
```

Option 2: Separating each rule out into its own table.

```
[[tool.tartufo.rule-patterns]]
reason = "RSA private key 2"
pattern = "-----BEGIN EC PRIVATE KEY-----"

[[tool.tartufo.rule-patterns]]
reason = "Null characters in GitHub Workflows"
pattern = '\0'
path-pattern = '\.github/workflows/(.*)\.yml'
```

Note: There are 3 different keys used here: `reason`, `pattern`, and `path-pattern`. Only `reason` and `pattern` are required. If no `path-pattern` is specified, then the pattern will be used to scan against all files.

Entropy Exclusion Patterns

Entropy scans can produce a high number of false positive matches such as git SHAs or MD5 digests. To avoid these false positives, you can use the `exclude-entropy-patterns` configuration option. These patterns will be applied to and matched against any strings flagged by entropy checks. As above, this directive utilizes an array of tables, enabling two forms:

Option 1:

```
[tool.tartufo]
exclude-entropy-patterns = [
    {path-pattern = 'docs/.*\.md$', pattern = '^[a-zA-Z0-9]$', reason = 'exclude all git␣
↪SHAs in the docs'},
```

(continues on next page)

(continued from previous page)

```
    {path-pattern = '\.github/workflows/.*\.yml', pattern = 'uses: .*@[a-zA-Z0-9]{40}',␣
↪reason = 'GitHub Actions'}
]
```

Option 2:

```
[[tool.tartufo.exclude-entropy-patterns]]
path-pattern = 'docs/.*\.md$'
pattern = '^[a-zA-Z0-9]$'
reason = 'exclude all git SHAs in the docs'

[[tool.tartufo.exclude-entropy-patterns]]
path-pattern = '\.github/workflows/.*\.yml'
pattern = 'uses: .*@[a-zA-Z0-9]{40}'
reason = 'GitHub Actions'
```

There are 5 relevant keys for this directive, as described below.

Key	Required	Value	Description
pattern	Yes	Regular expression	The pattern used to check against the match
path-pattern	No	Regular expression	A pattern to specify to what files the exclusion will apply
reason	No	String	A plaintext reason the exclusion has been added
match-type	No	String ("match" or "scope")	Whether to perform a search or match regex operation
scope	No	String ("word" or "line")	Whether to match against the current word or full line of text

3.5 Upgrading

Upgrading `tartufo` from release 2 to release 3 introduces some behavioral and interface changes. Current users of release 2 should review this summary to understand how to transition to release 3 as painlessly as possible.

3.5.1 General Behavioral Changes

`tartufo` release 3 is generally more accurate than previous releases. It may detect problems that were not recognized by release 2 scans (especially earlier 2.x releases). A scan of your code base prior to upgrading will simplify the process of identifying new findings that are attributable to these behavior changes so they can be remediated or suppressed.

Remote Repository Scanning

`tartufo` releases between 2.2.0 and 2.9.0 (inclusive) mishandled remote repositories. Only the repository's default branch was scanned; secrets present only on other branches would not be discovered.

Additionally, the `--branch branch-name` option did not operate correctly. Some versions scanned nothing and reported no errors, and other versions aborted immediately after reporting the branch did not exist (even when it did).

`tartufo` release 3 scans all remote repository branches by default, and correctly scans only a single branch if one is specified using `--branch`. As a consequence, it may discover secrets that were not reported by earlier versions.

These fixes were backported to `tartufo` release 2.10.0.

Live Output

`tartufo` release 3 reports findings incrementally as a scan progresses; previous releases did not perform any reporting until the entire scan was completed.

Entropy Scanning

Beginning with release 3, `tartufo` recognizes base64url-encoded strings in addition to base64-encoded strings.

If your code contains base64url encodings (or strings that look like base64url encodings), these strings now will be checked for high entropy and may produce new findings.

Additionally, strings that contain combinations of base64 and base64url character sets (whether they are actual encodings or not) will be scanned differently by release 3. Previously, base64 substrings would be extracted and scanned independently, but now the larger string will be scanned (once) in its entirety. This can result in signature changes (because the new suspect string is larger than the string recognized by release 2.x) and possibly fewer findings (because one longer string will be flagged instead of multiple substrings). Real-life files do not typically contain sequences that will exhibit this behavior.

Shallow Repositories

When `tartufo` release 2 scanned a shallow repository (a repository with no refs or branches found locally), it did not actually scan anything.

In the same situation, `tartufo` release 3 scans the repository HEAD as a single commit, effectively scanning the entire existing codebase (but none of its history) at once.

This scenario is commonly encountered in GitHub actions, which perform shallow checkouts.

Nonfunctional Options

`tartufo` release 3 uses pygit2 instead of GitPython to access git repositories. While this provides vastly improved performance with generally equivalent functionality, some less-frequently used options require reimplementation and currently are nonfunctional. We plan to provide either replacements or reimplementations in the future.

The `--since-commit` option is intended to restrict scans to a subset of repository history; the `--max-depth` option provides roughly the same functionality specified differently. Both options are ignored by `tartufo` release 3. Refer to #267 for more information about this topic.

3.5.2 Changes to Default Behavior

Some defaults have changed for the new release. If you wish to retain the previous behavior, adjust your configuration options to request it explicitly.

Regex Scanning

Previously, `tartufo` did not perform regex scanning for sensitive strings by default. Release 3 *does* perform regex scanning by default.

Explicitly disable regex scanning to preserve the old behavior:

```
[tool.tartufo]
regex = false
```

Alternatively, add `--no-regex` to your `tartufo` command line.

3.5.3 Retired Options

Some options that were deprecated in later 2.x releases no longer are supported by version 3. You will need to alter your command line and/or configuration options to specify the required information in a release 3-compatible manner.

Fetch Before Local Scans

`tartufo` release 2 supported command option `--fetch` for local repository scans, in order to force an update of the repository before scanning it. `tartufo` release 3 no longer recognizes this option.

Instead of using `--fetch`, perform an explicit `git fetch` command prior to executing `tartufo`.

If you were using `--no-fetch`, simply remove the option. `tartufo` release 3 never performs fetches prior to scanning local repositories.

Output Formatting

`tartufo` release 2 supported command options `--json` and `--compact` to control output formatting. `tartufo` release 3 no longer recognizes these options.

Replace `--json` with `--output-format json`, and replace `--compact` with `--output-format compact`.

Path Scoping

`tartufo` release 2 supported command options `--include-paths` and `--exclude-paths` in order to control which files were (or were not) scanned. In either case, the option accepted a filename which was expected to contain path patterns to include or exclude, respectively. `tartufo` release 3 no longer recognizes these options.

It is recommended that these path expressions be migrated from the external file to your `pyproject.toml` file and converted to TOML array of tables format. The supported formats are described in *Limiting Scans by Path*.

3.5.4 Deprecated Options

`tartufo` release 3 deprecates some release 2 options. Although no action is required at this time, replacing these options with their newer equivalents will reduce future disruptions when they are retired.

3.5.5 Updating Signatures

`tartufo` release 3.2.0 deprecated a number of signatures that were generated with the leading +/- from the git diff erroneously. These signatures will no longer work in release 4. An additional command `tartufo update-signatures` was added which scans a local repository, automatically updates the deprecated exclude-signatures in your tartufo config file, and removes any resulting duplicates.

Use `--no-update-configuration` to prevent `tartufo` from overwriting your config. Use `--no-remove-duplicates` to prevent `tartufo` from removing duplicate signatures.

When removing duplicate signatures `tartufo` will keep the first signature it finds and discard the rest.

External Rules Files

The `--rules` command option accepts a filename that is expected to contain one or more rule patterns. `tartufo` release 3 deprecates this option.

It is recommended that these patterns be migrated from the external file to your `pyproject.toml` file and converted to TOML array of tables format. The supported formats are described in *Rule Patterns*.

Entropy Scan Sensitivity

The new `--entropy-sensitivity` option is intended to replace both `--b64-entropy-score` and `--hex-entropy-score`. The new option adjusts sensitivity for both encodings consistently, using a scale of 0-100. To convert:

- Users of `--b64-entropy-score` should divide the provided value by 0.06 to obtain the equivalent `--entropy-sensitivity` setting
- Users of `--hex-entropy-score` should divide the provided value by 0.04 to obtain the equivalent `--entropy-sensitivity` setting

Users who require different base64 and hexadecimal sensitivities should open an issue that explains their use case.

3.6 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, or the Tartufo Mailing list.

3.6.1 Table of Contents

3.6.2 Answering Questions

One of the most important and immediate ways you can support this project is to answer questions on Slack , Github, or the Tartufo Mailing list. 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.

3.6.3 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.

3.6.4 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 'main' 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.

3.6.5 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.

3.6.6 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.

3.6.7 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! Additional info if you run into trouble: Poetry Environments

Step 1: Install PoetryStep 2: Run `poetry install`Step 3: Optionally Run `poetry shell`

Done!

Code Style

From PEP 8 – Style Guide for Python Code

> A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

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!

There can be more to code style than, "spaces vs tabs." Styling conventions, best practices, and language developments can all lead to changes to what is the best code style be followed. When existing code needs changing, or new code is submitted, questions can then arise as to what style to follow or what best practice takes precedence.

This isn't something that has a hard and fast rule. As a rule of thumb, we ask that contributors take each pull request as an opportunity to uplift the code they are touching to be in alignment with current recommendations. In an ideal world, the newest code in the codebase will reflect the best patterns to use, but if there is existing code being changed it is a balance between keeping style versus adoption of new ones.

There may be occasions when the maintainers of the project may ask a contributor to adopt a newer style or pattern to aid in uplifting the project as a whole and to help our community become better software developers.

We understand that time or other constraints may mean such requests are not able to be part of the pull request. In such cases please engage in communication with the maintainers. We would much rather have a pull request of a feature that aligns with the current codebase styles and patterns; and add an issue to the backlog to refactor with new patterns when bandwidth permits; than to have you not contribute a pull request.

3.6.8 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
```

3.6.9 Contributing as a Maintainer

On top of all our lovely contributors, we have a core group of people who act as maintainers of the project. They are the ones who are the gatekeepers, and make sure that issues are addressed, PRs are merged, and new releases issued, all while ensuring a high bar of quality for the code and the project.

Issuing a New Release

This process is thankfully mostly automated. There are, however, a handful of manual steps that must be taken to kick off that automation. It is all built this way to help ensure that issuing a release is a very conscious decision, requiring peer review, and cannot easily happen accidentally. The steps involved currently are:

- Create a new branch locally for the release, for example:

```
> git checkout -b releases/v2.1.0
```

- Tell Poetry to bump the version:

```
> poetry version minor
Bumping version from 2.0.1 to 2.1.0
```

  - Note: All this is doing, is updating the version number in the `pyproject.toml`. You can totally do this manually. This command just might be a bit quicker. And it's nice to have a command to do it for you. Yay automation!

- Update the CHANGELOG with the appropriate new version number and release date.
- Create a pull request for these changes, and get it approved!
- Once your PR has been merged, the final piece is to actually create the new release.
  1. Go to the `tartufo` releases page and click on `Draft a new release`.
  2. Enter an appropriate tag version (in this example, `v2.1.0`).
  3. Title the release. Generally these would just be in the form `Version 2.1.0`. (Not very creative, I know. But predictable!)
  4. Copy-paste the CHANGELOG entries for this new version into the description.
  5. Click `Publish release`!

Congratulations, you've just issued a new release for `tartufo`. The automation will take care of the rest!

3.6.10 Additional Resources

- General GitHub Documentation
- GitHub Pull Request documentation

3.7 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.

3.7.1 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.

3.8 Project History

3.8.1 v4.0.0 - Jan 17 2023

Features: * [#433](https://github.com/godaddy/tartufo/pull/433) - Dropped support for deprecated flags rules, b64, hex

and corresponding code around deprecated options. Removed support for old signatures which generated with +/- chars in git diff.

- [#411](https://github.com/godaddy/tartufo/pull/411) - Drop support for python 3.6. This version reached end of life several years ago, and end of security support at the end of 2021. Users with a requirement to run tartufo on this python version should remain at v3.3.x.
- [#403](https://github.com/godaddy/tartufo/pull/403) - Add support for python 3.11. * Update various support libraries to current versions * Rebase container to python 3.11 * Add CI step to verify container is operational
- [#348](https://github.com/godaddy/tartufo/pull/348) - Add –no-git-check option to skip confirmation dialog for scan-folder

3.8.2 v3.3.1 - 23 Nov 2022

Bug fixes: * [#408](https://github.com/godaddy/tartufo/issues/408) - 3.3.0 container broken

- Rebuild container using python 3.10 base instead of python 3.11
- Eliminates reference to missing library present in 3.3.0 container
- Eliminates requirement for build-it-yerself libraries in container

3.8.3 v3.3.0 - 22 Nov 2022

Features: * [#401](https://github.com/godaddy/tartufo/pull/401) - Add report output format

Bug fixes: * [#375](https://github.com/godaddy/tartufo/pull/376) - Update the "Password in URL" default_regexes.json to identify the following:

- usernames of lengths between 3-40
- passwords of length between 3-40
- URL domain name, port, path, query parameters, and fragments of any length
- [#372](https://github.com/godaddy/tartufo/pull/372) Handle the case where exclude-signatures is a list of strings

3.8.4 v3.2.1 - 20 July 2022

Features: * [#368](https://github.com/godaddy/tartufo/pull/368) - Add update-signatures command to migrate deprecated signatures

3.8.5 v3.2.0 - 6 July 2022

Bug fixes: * [#360](https://github.com/godaddy/tartufo/issues/360) - Fix ANSI escape sequences being written to files on redirection * [#363](https://github.com/godaddy/tartufo/pull/363) - Fix leading +/- in Tartufo matched_strings

3.8.6 v3.1.4 - 31 May 2022

Bug fixes:

- [#352](https://github.com/godaddy/tartufo/pull/352) - Fix tartufo ignoring new files added to a Git repo
- [#351](https://github.com/godaddy/tartufo/pull/351) - Make pre-commit check staged changes instead of entire working directory

Misc: * [356](https://github.com/godaddy/tartufo/pull/356) - Update documentation * [354](https://github.com/godaddy/tartufo/pull/354) - Add a tartufo scan step in Tartufo's CI

3.8.7 v3.1.3 - 4 April 2022

Bug fixes:

- [#329](https://github.com/godaddy/tartufo/issues/329) - Entropy exclusions(exclude-entropy-patterns) ignored when using

scan-local-repo * [#343](https://github.com/godaddy/tartufo/issues/343) - Entropy exclusions(exclude-entropy-patterns) ignored when using scan-remote-repo

3.8.8 v3.1.2 - 28 March 2022

Bug fixes:

- [#339](https://github.com/godaddy/tartufo/issues/339) - Fix *click* compatibility issues. Specifically: * Pin to < 8.1.0 for Python 3.6, as support for that version was dropped * Pin to >= 8.1.0 for Python 3.7+, and change *resultcallback* usage to *result_callback* * Upgraded to the latest version of *black*

3.8.9 v3.1.1 - 25 March 2022

Bug fixes:

- [#336](https://github.com/godaddy/tartufo/issues/336) - *_issue_file* was not defined by default, causing all scans to fail

3.8.10 v3.1.0 - 24 March 2022

Features:

- [#328](https://github.com/godaddy/tartufo/pull/328) - Buffer issues beyond –buffer-size to a temporary file

Bug fixes:

- [#330](https://github.com/godaddy/tartufo/pull/330) - Allow newer versions of pygit2 for newer versions of Python

3.8.11 v3.0.0 - 5 January 2022

Version 3.0.0. Stable Release.

3.8.12 v3.0.0-rc.3 - 13 December 2021

Bug fixes:

- [#301](https://github.com/godaddy/tartufo/issues/301) - Parse new-style option values correctly, avoid duplicate processing of global options, and don't generate spurious deprecation warnings for these options.
- [#303](https://github.com/godaddy/tartufo/pull/303) - Include or exclude git submodules only if we're not working with a mirror clone.

3.8.13 v3.0.0-rc.2 - 09 December 2021

Bug fixes:

- [#296](https://github.com/godaddy/tartufo/pull/296), [#297](https://github.com/godaddy/tartufo/pull/297) - Fix our Docker image so that it actually builds, and the tartufo command works
- [#298](https://github.com/godaddy/tartufo/pull/298) - Fix how we determine whether we are scanning a shallow clone, so that it is more bulletproof.

3.8.14 v3.0.0-rc.1 - 09 December 2021

Bug fixes:

- [#284](https://github.com/godaddy/tartufo/pull/284) - Fix handling of first commit during local scans; an exception was raised instead of processing the commit.

Misc:

- [#282](https://github.com/godaddy/tartufo/pull/282) - Remove old style config for *exclude-entropy-patterns*
- [#292](https://github.com/godaddy/tartufo/pull/292) - Use the latest *click* to provide better output on boolean flag defaults

Features:

- [#270](https://github.com/godaddy/tartufo/issues/270) - When no refs/branches are found locally, tartufo will now scan the repo HEAD as a single commit, effectively scanning the entire codebase at once.
- [#265](https://github.com/godaddy/tartufo/issues/265) - Adds new *–entropy-sensitivity* option which provides a friendlier way to adjust entropy detection sensitivity. This replaces *–b64-entropy-score* and *–hex-entropy-score*, which now are marked as deprecated.
- [#273](https://github.com/godaddy/tartufo/issues/273) - Entropy checking support routines have been rewritten to utilize library abstractions and operate more efficiently while returning identical results.
- [#177](https://github.com/godaddy/tartufo/issues/177) - [base64url](https://datatracker.ietf.org/doc/html/rfc4648#section-5) encodings are now recognized and scanned for entropy.
- [#268](https://github.com/godaddy/tartufo/issues/268) - Adds a new *–recurse / –no-recurse* flag which allows users to recursively scan the entire directory or just the root directory
- [#256](https://github.com/godaddy/tartufo/issues/256) - Deprecated *–rules* in favor of a new *rule-patterns* config option. This is the final piece of config that was still stored in an external file.
- [#202](https://github.com/godaddy/tartufo/issues/202) - Supports new format of exclusions in config file with the ability to specify the reason along with exclusion
- [#257](https://github.com/godaddy/tartufo/issues/257) - Supports new format of include-path-patterns and exclude-path-patterns in config file with the ability to specify the reason along with the path-patterns.

3.8.15 v3.0.0-alpha.1 - 11 November 2021

Bug fixes:

- [#247](https://github.com/godaddy/tartufo/issues/247) - The *–branch* qualifier now works again when using *scan-remote-repo*.

Features:

- [#227](https://github.com/godaddy/tartufo/pull/227) - Report findings incrementally as scan progresses instead of holding all of them until it has completed. This is a re-implementation of [#108](https://github.com/godaddy/tartufo/pull/108); thanks to @dclayton-godaddy for showing the way.
- [#244](https://github.com/godaddy/tartufo/pull/244) - Drops support for *–fetch/–no-fetch* option for local scans
- [#253](https://github.com/godaddy/tartufo/issues/253) - Drops support for *–json* and *–compact* and consolidates the two options into one *—output-format json/compact/text*
- [#259](https://github.com/godaddy/tartufo/pull/259) - Adds a new *–scan-filenames/–no-scan-filenames* flag which allows users to enable or disable file name scanning.
- [#254](https://github.com/godaddy/tartufo/pull/260) - Changes the default value of *–regex/–no-regex* to True.

Misc:

- [#255](https://github.com/godaddy/tartufo/issues/255) - Removed deprecated flags –include-paths and –exclude-paths

3.8.16 v2.10.1 - 27 December 2021

Bug fixes:

- [#309](https://github.com/godaddy/tartufo/pull/309) Fixes an issue where verbose output display

would error out if the new-style entropy exclusion pattern was used

3.8.17 v2.10.0 - 3 November 2021

Bug fixes:

- [#247](https://github.com/godaddy/tartufo/issues/247) All versions of tartufo from v2.2.0 through v2.9.0 inclusive mishandle *scan-remote-repo*. Only the repository's default branch was scanned, and secrets present in other branches would not be discovered. Additionally, the *–branch branch-name* option did not operate correctly for remote repositories. Some versions would scan nothing and report no errors, and other versions aborted immediately, claiming the branch did not exist (even if it did). v2.10.0 corrects these problems and may detect secrets that were not reported by previous versions.

Features:

- [#231](https://github.com/godaddy/tartufo/issues/231) Change toml parsing library to use tomlkit

Other changes:

- [#251](https://github.com/godaddy/tartufo/issues/251) Document update to use –no-fetch flag to all scan-local-repo

3.8.18 v2.9.0 - 19 October 2021

Bug fixes:

- Reverted [#222](https://github.com/godaddy/tartufo/pull/222) – users had been relying on the previously implemented behavior, causing this change to break their pipelines.

Features:

- Behavior introduced in [#222](https://github.com/godaddy/tartufo/pull/222) is now opt-in via an updated config specification for *exclude-entropy-patterns*. This is now done via a TOML table, rather than a specifically patterned string. Users who have the old style configuration will now receive a *DeprecationWarning* stating that the old behavior will go away with v3.0.
- Fixed up warning handling so that we can display *DeprecationWarnings* to users more easily.
- [#223](https://github.com/godaddy/tartufo/pull/223) New flags (*-b64/–b64-entropy-score* and *-hex/–hex-entropy-score*) allow for user tuning of the entropy reporting sensitivity. They default to 4.5 and 3.0, respectively.

3.8.19 v2.8.1 - 11 October 2021

Bug fixes:

- [#222](https://github.com/godaddy/tartufo/pull/222) - Allow exclude-entropy-patterns to match lines containing partial matches – thanks to @kbartholomew-godaddy for the work on this one!

3.8.20 v2.8.0 - 14 September 2021

Features:

- [#83](https://github.com/godaddy/tartufo/issues/83) - New *scan-folder* command to scan files without viewing as a git repository.

Bug fixes:

- [#220](https://github.com/godaddy/tartufo/pull/220) - Display an explicit error message when a requested branch is not found, as opposed to failing silently.

Misc:

- [#219](https://github.com/godaddy/tartufo/pull/219) - Incremental optimizations; using *__slots__* for the *Issue* class to improve memory consumption, and a small logic speed-up in when we generate the diff between commits. Both of these should help at least some when it comes to scanning very large repositories.

3.8.21 v2.7.1 - 23 August 2021

Bug fixes:

- [#211](https://github.com/godaddy/tartufo/issues/211) - Attempt to fix a case where output encoding could be set to cp1252 on Windows, which would cause a crash if unicode characters were printed. Now issues are output as utf-8 encoded bytestreams instead.

3.8.22 v2.7.0 - 10 August 2021

Features:

- [#96](https://github.com/godaddy/tartufo/issues/96) - Explicitly handle submodules. Basically, always ignore them by default. There is also a new option to toggle this functionality: *–include-submodules*
- Add *exclude_entropy_patterns* to output

3.8.23 v2.6.0 - 30 June 2021

Features:

- [#194](https://github.com/godaddy/tartufo/issues/194) - Half bugfix, half feature. Now when an excluded signature in your config file is found as an entropy match, tartufo will realize that and no longer report it as an issue.
- [#5](https://github.com/godaddy/tartufo/issues/5) - Remove the dependency on *truffleHogRegexes*. This enables us to take full control of the default set of regex checks.

Bug fixes:

- [#179](https://github.com/godaddy/tartufo/issues/179) - Iterate over commits in topological order, instead of date order.

3.8.24 v2.5.0 - 15 June 2021

Features:

- [#145](https://github.com/godaddy/tartufo/issues/145) - Adds *–exclude-path-patterns* and *–include-path-patterns* to simplify config in a single .toml file
- [#87](https://github.com/godaddy/tartufo/issues/87) - Adds *–exclude-entropy-patterns* to allow for regex-based exclusions

Bug fixes:

- Write debug log entries when binary files are encountered
- Pinned all linting tools to specific versions and set all tox envs to use poetry
- Disabled codecov due to security breach

3.8.25 v2.4.0 - 05 March 2021

Features:

- #76 - Added logging! You can now use the *-v/–verbose* option to increase the amount of output from tartufo. Specifying multiple times will incrementally increase what is output.
- Added a *–log-timestamps/–no-log-timestamps* option (default: True) so that timestamps can be hidden in log messages. This could be helpful when, for example, comparing the output from multiple runs.
- #107 - Added a *–compact/–no-compact* option for abbreviated output on found issues, to avoid unintentionally spamming yourself. (Thanks to @dclayton-godaddy for his work on this one)

Bug fixes:

- #158 - The *–branch* option was broken and would not actually scan anything

3.8.26 v2.3.1 - 16 February 2021

Bug fixes:

- Added rust toolchain to allow for building of latest cryptography

Other changes:

- Added no-fetch to code snippets and note about what it does

3.8.27 v2.3.0 - 04 February 2021

Features:

- #42 - Report output on clean or successful scan. Add new *-q/–quiet* option to suppress output
- #43 - Report out of the list of exclusions. Add new *-v/–verbose* option to print exclusions
- #159 - Switched our primary development branch from *master* -> *main*
- Updated BFG refs from 1.13.0 to 1.13.2

3.8.28 v2.2.1 - 02 December 2020

Bugfixes:

- Rev build and release versions to match

3.8.29 v2.2.0 - 02 December 2020

Features:

- #119 - Added a new *–fetch*/*–no-fetch* option for local scans, controlling whether the local clone is refreshed before scan. (Thanks @jgowdy!)
- #125 - Implement CODEOWNERS and auto-assignment to maintainers on PRs

Bugfixes:

- #115 - Strange behavior can manifest with invalid sub-commands
- #117 - Ignore whitespace-only lines in exclusion files
- #118 - Local scans fetch remote origin
- #121 - Match rules specified with –git-rules-repo were not included in scans
- #140 - Ensure a valid output folder name in Windows

Other changes:

- #95 - Run CI across Linux, Windows, and MacOS
- #130 - Added references to Tartufo GoogleGroups mailing list to docs
- Fixed testing in Pypy3 and explicitly added Python 3.9 support
- #134 - Documented the release process
- #143 - Updated GitHub Action hashes to newest rev to address <https://github.blog/changelog/2020-10-01-github-actions-deprecating-set-env-and-add-path-commands/> where possible

3.8.30 v2.0.1 - 09 October 2020

- Fix the Docker build & deploy

3.8.31 v2.0.0 - 09 October 2020

- #74, #75 - Rewrote and refreshed the documentation for the new 2.0 usage (via #111)

3.8.32 v2.0.0a2 - 05 October 2020

This bugfix release is to take care of a handful of issues discovered during the initial alpha release for 2.0.

- #68 - Added consistent documentation through the codebase for classes, methods, and all other API elements (via #92)
- #90 - Presenting a friendlier error message when there is an error interacting with git (via #93)
- #94 - Fix tests that were failing on MacOS (via #97)
- #86 - Treat *tartufo.toml* preferentially over *pyproject.toml* when loading config (via #101)

- #91 - Load config from scanned repositories. This functionality previously existed in 1.x, but was missed during the rebuild for v2.0. This also resulted in a bit of an overall rewrite of config file discovery to eliminate some duplicated logic. (via #103)

3.8.33 v2.0.0a1 - 18 November 2020

This is a whole brand new tartufo! It's been entirely restructured, rewritten, retested, rebuilt, and remade! It's now more extensible, readable, testable, and usable.

New features include:

- #2 - Verified/approved exclusions are now handled by way of hash signatures. * These hashes are created on a combination of the matched string and filename where the match was found. They are generated using the *BLAKE2* hashing algorithm. (via #61)
- #7 - A working directory can now be specified to clone to when scanning a remote repository. (via #81)
- #11 - Removed the *–cleanup* option and added a *–output-dir* in its place. Issues are now written to disk only when specifically requested by providing an output directory. (via #82)
- #39 - The functionality is now split into sub-commands (via #78) Available sub-commands are, for now: * pre-commit * scan-local-repo * scan-remote-repo
- The entire library has been refactored and nearly all logic has been put into its most appropriate place. It should now be possible to use this whole tool as a library, and not just a CLI application. (via #29, #65, #67, #70)

Bug fixes include:

- #55 - The tests no longer iterate over this repository's history; everything has been sufficiently split out to make it more testable without needing to look at an actual git history. (via #70)
- #72 - Specifying a non-git path no longer causes an error (via #80)

Other changes:

- Issues found during the scan are now represented by a class, instead of some amorphous dictionary (via #29) * Further, since a single *Issue* is instantiated per match, the output key for the matches has changed from *strings_found* to *matched_string*.
- #25 - Set up full documentation on Read The Docs (via #38)
- #30 - Support for Python 2 has been dropped (via #31)
- #58 - CI is now handled by GitHub Actions (via #59)

3.8.34 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.

3.8.35 v1.1.1 - 13 December 2019

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

3.8.36 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

3.8.37 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.

3.8.38 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*

3.8.39 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.

3.8.40 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

3.9 Would you like to know more?

If the other documentation left you wondering what to do with the results of your scans, and unsure how to get rid of those pesky leaked secrets, then look no further!

3.9.1 End-to-End Example

An End-to-End example walkthrough of a `tartufo` scan and the process of purging the dirty evil passwords that somehow ended up in your code commits. We will use an additional tool: `BFG` (https://rtyley.github.io/bfg-repo-cleaner/. More on this later!)

Note: OPTIONAL Development only: Setup poetry if you want to use the most recent non-released build from github (may not be stable)

This project uses Poetry to manage its dependencies and do a lot of the heavy lifting. So you'll need to clone the tartufo repo and setup poetry!

```
git clone git@github.com:godaddy/tartufo.git
```

Development Use Only Poetry Setup: Setting up a development environment

---

1. Clone your repo!

   Select and clone the repo you want to run tartufo on

   ```
   # Clone your repo, variables used later:
   GITHUBPROJECT="yourproject"
   GITHUBREPO="myrepo.git"
   GITHUBADDRESS="github.com"
   git clone --mirror git@${GITHUBADDRESS}:${GITHUBPROJECT}/${GITHUBREPO}
   ```

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

   Scan your repo!

   ```
   # Run Tartufo on your repo and create a list of high entropy items to remove:
   tartufo --regex --output-format json scan-local-repo ${GITHUBREPO} | \
       jq -r '.found_issues[].matched_string' | \
       sort -u > remove.txt
   ```

   Now you have a "bad password" file! Take a look through it, see if anything is wrong. This file will be used by `BFG` to replace these flagged "bad password" entries with `***REMOVED***`.

   Note: It is important that you read through this file to make sure there are not exceptions that you want to remove and exclude with tartufo! Read more about configuring exclusions here: *Scan Limiting (Exclusions)*

   ---

3. Cleanup repo using `BFG` and the above remove.txt file

There's a very slick tool designed to clean up git commit history called BFG. By default, `BFG` doesn't modify the contents of your latest commit on your main (or 'HEAD') branch, even though it will clean all the commits before it. This of course means if you have active code with "bad passwords", `tartufo` will still fail. But let's take the bulk of the old entries out first.

```
# Cleanup with BFG
wget https://repo1.maven.org/maven2/com/madgag/bfg/1.13.2/bfg-1.13.2.jar
# Make a backup
cp -r ${GITHUBREPO} backup_${GITHUBREPO}
java -jar bfg-1.13.2.jar --replace-text remove.txt ${GITHUBREPO}
```

4. Uh Oh!

   Occasionally the results will be too big to process all at once. If that happens, simply split up the results and loop through them.

```
# occasionally the results will be to big to process all at once
split -l 200 remove.txt
for f in x*; do java -jar bfg-1.13.2.jar --replace-text $f ${GITHUBREPO}; done
```

5. Proceed with cleanup/audit

   Now you have removed the low hanging fruit, it's time to look at the tough stuff

```
# run tartufo again to check for any remaining potential secrets
leftovers=`tartufo --regex -od ~/temp scan-local-repo ${GITHUBREPO}`
tmppath=`echo -e "$leftovers" | tail -n1 | awk '{print $6}'`
# look through the remaining strings
# if there's anything that looks like it shouldn't be there, dig into it and clear␣
↪it out
cat ${tmppath}/* | jq '. | " \(.file_path) \(.matched_string) \(.signature)"' |␣
↪sort -u
```

6. Take a good look at the output of the above, make sure there are no secrets or other sensitive data remaining.

   Now you are going to exclude the signatures for the remaining items (which you have verified are non-risk)

```
# now you are ready to ignore those webhook urls:
cat ${tmppath}/* | jq -r '.signature' | sort -u > allsignatures.txt
sed -i -e 's/$/\",/g' -e 's/^/  \"/g' allsignatures.txt
linestr=`grep -n 'exclude-signatures = \[' tartufo.toml`
line=`echo $linestr | cut -d ":" -f 1`
line=$(($line+1))
{ head -n $(($line-1)) tartufo.toml; cat allsignatures.txt; tail -n +$line tartufo.
↪toml; } > tartufo.toml_new
mv tartufo.toml tartufo.toml_bak
mv tartufo.toml_new tartufo.toml
# one final run to make sure your signatures are all set
tartufo --regex scan-local-repo ${gitrepo}
```

7. Once you are happy with the data that is being stored, time to commit the changes back up!

Important: This does a force push, effectively rewriting the history of your git repository!

After doing this, you will want to be absolutely certain that all users who have previously cloned this repository

pull down a fresh clone in order to prevent re-introducing the former bad history.

```
cd ${GITHUBREPO}
git reflog expire --expire=now --all && git gc --prune=now --aggressive
git push
```

8. Danger Will Robinson, Danger!

   You MAY get an error (example error below). If so, keep reading!

```
(.venv) you@LTDV-you:~/tartufo/yourrepo.git$ git push
Counting objects: 1014, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (359/359), done.
Writing objects: 100% (1014/1014), 130.35 KiB | 0 bytes/s, done.
Total 1014 (delta 662), reused 964 (delta 638)
remote: Resolving deltas: 100% (662/662), completed with 24 local objects.
To git@GITHUBADDRESS:yourproject/yourrepo.git
 + 56f7476...c76ed2b main -> main (forced update)
 ! [remote rejected] refs/pull/1/head -> refs/pull/1/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/2/head -> refs/pull/2/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/3/head -> refs/pull/3/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/4/head -> refs/pull/4/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/5/head -> refs/pull/5/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/6/head -> refs/pull/6/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/7/head -> refs/pull/7/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/8/head -> refs/pull/8/head (deny updating a hidden␣
↪ref)
 ! [remote rejected] refs/pull/9/head -> refs/pull/9/head (deny updating a hidden␣
↪ref)
error: failed to push some refs to 'git@GITHUBADDRESS:yourproject/yourrepo.git'
(.venv) you@LTDV-you:~/tartufo/yourrepo.git$
```

   If you get the above error, it might actually be okay; simply re-run `tartufo` from your main branch. Only continue with the below steps if there are results that are not clean. Please note, this solution will remove PR history (but not commit history):

```
# create a new blank repo, put the name below
NEWGITHUBREPO="my-repo-tartufoized.git"
cd ../
rm -rf ${GITHUBREPO}
# Create a bare clone of the repository.
git clone --bare git@${GITHUBADDRESS}:${GITHUBPROJECT}/${GITHUBREPO}
# Mirror-push to the new temporary repository
cd ${GITHUBREPO}
git push --mirror git@${GITHUBADDRESS}:${GITHUBPROJECT}/${NEWGITHUBREPO}
```

(continues on next page)

(continued from previous page)

```
cd ..
rm -rf ${GITHUBREPO}
# bare clones are missing data, it is easier to re-clone the repo now that it does␣
↪not have PRs
git clone git@${GITHUBADDRESS}:${GITHUBPROJECT}/${NEWGITHUBREPO}
# Now run bfg
java -jar bfg-1.13.2.jar --replace-text remove.txt ${NEWGITHUBREPO}
cd ${NEWGITHUBREPO}
git reflog expire --expire=now --all && git gc --prune=now --aggressive
git push
# re-run tartufo on new repo
tartufo --regex -od ~/temp scan-remote-repo git@${GITHUBADDRESS}:${GITHUBPROJECT}/$
↪{NEWGITHUBREPO}
# should have very little (if any) output. check the newly outputed results in the␣
↪given tmp folder
ls ~/temp/tartufo-scan-results-/ | wc -l
```

Done!

3.10 API

This part of the documentation lists the full API reference of all public classes and functions.

3.10.1 tartufo.config

tartufo.config.**compile_path_rules**(*patterns*)

Take a list of regex strings and compile them into patterns.

Any line starting with # will be ignored.

Parameters
patterns (Iterable[str]) – The list of patterns to be compiled

Return type
List[Pattern]

tartufo.config.**compile_rules**(*patterns*)

Take a list of regex string with paths and compile them into a List of Rule.

Parameters
patterns (Iterable[Dict[str, str]]) – The list of patterns to be compiled

Return type
List[*Rule*]

Returns
List of Rule objects

tartufo.config.**configure_regexes**(*include_default=True*, *rule_patterns=None*, *rules_repo=None*, *rules_repo_files=None*)

Build a set of regular expressions to be used during a regex scan.

Parameters

- **include_default** (bool) – Whether to include the built-in set of regexes
- **rules_files** – A list of files to load rules from
- **rule_patterns** (Optional[Iterable[Dict[str, str]]]) – A set of previously-collected rules
- **rules_repo** (Optional[str]) – A separate git repository to load rules from
- **rules_repo_files** (Optional[Iterable[str]]) – A set of patterns used to find files in the rules repo

Return type
Set[*Rule*]

Returns
Set of *Rule* objects to be used for regex scans

tartufo.config.**load_config_from_path**(*config_path*, *filename=None*, *traverse=True*)

Scan a path for a configuration file, and return its contents.

All key names are normalized to remove leading "-"/"–" and replace "-" with "_". For example, "–repo-path" becomes "repo_path".

In addition to checking the specified path, if `traverse` is `True`, this will traverse up through the directory structure, looking for a configuration file in parent directories. For example, given this directory structure:

```
working_dir/
|- tartufo.toml
|- group1/
|  |- project1/
|  |  |- tartufo.toml
|  |- project2/
|- group2/
   |- tartufo.toml
   |- project1/
   |- project2/
      |- tartufo.toml
```

The following `config_path` values will load the configuration files at the corresponding paths:

config_path	file
working_dir/group1/project1/	working_dir/group1/project1/tartufo.toml
working_dir/group1/project2/	working_dir/tartufo.toml
working_dir/group2/project1/	working_dir/group2/tartufo.toml
working_dir/group2/project2/	working_dir/group2/project2/tartufo.toml

Parameters

- **config_path** (Path) – The path to search for configuration files
- **filename** (Optional[str]) – A specific filename to look for. By default, this will look for both `tartufo.toml` and then `pyproject.toml`.
- **traverse** (*bool*) –

Raises

- **FileNotFoundError** – If no config file was found
- ***types.ConfigException*** – If a config file was found, but could not be read

Return type
Tuple[Path, MutableMapping[str, Any]]

Returns
A tuple consisting of the config file that was discovered, and the contents of that file loaded in as TOML data

tartufo.config.**load_rules_from_file**(*rules_file*)

Load a set of JSON rules from a file and return them as compiled patterns.

Parameters
rules_file (TextIO) – An open file handle containing a JSON dictionary of regexes

Raises
ValueError – If the rules contain invalid JSON

Return type
Set[*Rule*]

tartufo.config.**read_pyproject_toml**(*ctx*, *_param*, *value*)

Read config values from a file and load them as defaults.

Parameters

- **ctx** (Context) – A context from a currently executing Click command
- **_param** (Parameter) – The command parameter that triggered this callback
- **value** (str) – The value passed to the command parameter

Raises
click.FileError – If there was a problem loading the configuration

Return type
Optional[str]

3.10.2 tartufo.scanner

class tartufo.scanner.**FolderScanner**(*global_options*, *target*, *recurse*)

Bases: *ScannerBase*

Used to scan a folder.

Used for scanning a folder.

Parameters

- **global_options** (*GlobalOptions*) – The options provided to the top-level tartufo command
- **target** (str) – The local filesystem path to scan
- **recurse** (bool) – Whether to recurse into sub-folders of the target

b64_entropy_limit

Returns low entropy limit for suspicious base64 encodings

static calculate_entropy(*data*)

Calculate the Shannon entropy for a piece of data.

This essentially calculates the overall probability for each character in *data* to be to be present. By doing this, we can tell how random a string appears to be.

Adapted from http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html

Parameters
data (str) – The data to be scanned for its entropy

Return type
float

Returns
The amount of entropy detected in the data

property chunks: Generator[*Chunk*, None, None]

Yield the individual files in the target directory.

property completed: bool

Return True if scan has completed

Returns
True if scan has completed; False if scan is in progress

compute_scaled_entropy_limit(*maximum_bitrate*)

Determine low entropy cutoff for specified bitrate

Parameters
maximum_bitrate (float) – How many bits does each character represent?

Return type
float

Returns
Entropy detection threshold scaled to the input bitrate

property config_data: MutableMapping[str, Any]

Supplemental configuration to be merged into the *_options information.

entropy_string_is_excluded(*string*, *line*, *path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **string** (str) – String to check against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – Path to check against rule path pattern

Return type
bool

Returns
True if excluded, False otherwise

evaluate_entropy_string(*chunk*, *line*, *string*, *min_entropy_score*)

Check entropy string using entropy characters and score.

Parameters

- **chunk** (*Chunk*) – The chunk of data to check
- **line** (str) – Source line containing string of interest
- **string** (str) – String to check
- **min_entropy_score** (float) – Minimum entropy score to flag

Return type
Generator[*Issue*, None, None]

Returns
Generator of issues flagged

property excluded_entropy: List[*Rule*]
Get a list of regexes used as an exclusive list of paths to scan.

property excluded_paths: List[Pattern]
Get a list of regexes used to match paths to exclude from the scan

excluded_signatures
Get a list of the signatures of findings to be excluded from the scan results.

Returns
The signatures to be excluded from scan results

global_options: *GlobalOptions*

hex_entropy_limit
Returns low entropy limit for suspicious hexadecimal encodings

property included_paths: List[Pattern]
Get a list of regexes used as an exclusive list of paths to scan

property issue_count: int

property issue_file: IO

property issues: List[*Issue*]
Get a list of issues found during the scan.

If the scan is still in progress, force it to complete first.

Returns
Any issues found during the scan.

load_issues()

Return type
Generator[*Issue*, None, None]

logger: Logger

recurse: bool

static rule_matches(*rule*, *string*, *line*, *path*)
Match string and path against rule.

Parameters

- **rule** (*Rule*) – Rule to perform match
- **string** (str) – string to match against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – path to match against rule path_pattern

Return type
bool

Returns
True if string and path matched, False otherwise.

property rules_regexes: Set[*Rule*]

Get a set of regular expressions to scan the code for.

Raises
types.ConfigException – If there was a problem compiling the rules

scan()

Run the requested scans against the target data.

This will iterate through all chunks of data as provided by the scanner implementation, and run all requested scans against it, as specified in *self.global_options*.

The scan method is thread-safe; if multiple concurrent scans are requested, the first will run to completion while other callers are blocked (after which they will each execute in turn, yielding cached issues without repeating the underlying repository scan).

Raises
types.ConfigException – If there were problems with the scanner's configuration

Return type
Generator[*Issue*, None, None]

scan_entropy(*chunk*)

Scan a chunk of data for apparent high entropy.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

scan_regex(*chunk*)

Scan a chunk of data for matches against the configured regexes.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

should_scan(*file_path*)

Check if the a file path should be included in analysis.

If non-empty, *self.included_paths* has precedence over *self.excluded_paths*, such that a file path that is not matched by any of the defined *self.included_paths* will be excluded, even when it is not matched by any of the defined *self.excluded_paths*. If either *self.included_paths* or *self.excluded_paths* are undefined or empty, they will have no effect, respectively. All file paths are included by this function when no inclusions or exclusions exist.

Parameters
file_path (str) – The file path to check for inclusion

Return type
bool

Returns
False if the file path is _not_ matched by *self.included_paths* (when non-empty) or if it is matched by *self.excluded_paths* (when non-empty), otherwise returns True

signature_is_excluded(*blob*, *file_path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **blob** (str) – The piece of data which is being scanned
- **file_path** (str) – The path and file name for the data being scanned

Return type
bool

store_issue(*issue*)

Return type
None

Parameters
issue (Issue) –

target: str

class tartufo.scanner.**GitPreCommitScanner**(*global_options*, *repo_path*, *include_submodules*)

Bases: *GitScanner*

For use in a git pre-commit hook.

Parameters

- **global_options** (*GlobalOptions*) – The options provided to the top-level tartufo command
- **repo_path** (str) – The local filesystem path pointing to the repository
- **include_submodules** (bool) – Whether to scan git submodules in the repository

_iter_diff_index(*diff*)

Iterate over a "diff index", yielding the individual file changes.

A "diff index" is essentially analogous to a single commit in the git history. So what this does is iterate over a single commit, and yield the changes to each individual file in that commit, along with its file path. This will also check the file path and ensure that it has not been excluded from the scan by configuration.

Note that binary files are wholly skipped.

Parameters
diff (Diff) – The diff index / commit to be iterated over

Return type
Generator[Tuple[str, str], None, None]

b64_entropy_limit

Returns low entropy limit for suspicious base64 encodings

static calculate_entropy(*data*)

Calculate the Shannon entropy for a piece of data.

This essentially calculates the overall probability for each character in *data* to be to be present. By doing this, we can tell how random a string appears to be.

Adapted from http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html

Parameters
data (str) – The data to be scanned for its entropy

Return type
float

Returns
The amount of entropy detected in the data

property chunks
Yield the individual file changes currently staged for commit.

property completed: bool
Return True if scan has completed

Returns
True if scan has completed; False if scan is in progress

compute_scaled_entropy_limit(*maximum_bitrate*)
Determine low entropy cutoff for specified bitrate

Parameters
maximum_bitrate (float) – How many bits does each character represent?

Return type
float

Returns
Entropy detection threshold scaled to the input bitrate

property config_data: MutableMapping[str, Any]
Supplemental configuration to be merged into the *_options information.

entropy_string_is_excluded(*string*, *line*, *path*)
Find whether the signature of some data has been excluded in configuration.

Parameters
- **string** (str) – String to check against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – Path to check against rule path pattern

Return type
bool

Returns
True if excluded, False otherwise

evaluate_entropy_string(*chunk*, *line*, *string*, *min_entropy_score*)
Check entropy string using entropy characters and score.

Parameters
- **chunk** (*Chunk*) – The chunk of data to check
- **line** (str) – Source line containing string of interest
- **string** (str) – String to check
- **min_entropy_score** (float) – Minimum entropy score to flag

Return type
Generator[*Issue*, None, None]

Returns
Generator of issues flagged

property excluded_entropy: List[*Rule*]

Get a list of regexes used as an exclusive list of paths to scan.

property excluded_paths: List[Pattern]

Get a list of regexes used to match paths to exclude from the scan

excluded_signatures

Get a list of the signatures of findings to be excluded from the scan results.

Returns
The signatures to be excluded from scan results

filter_submodules(*repo*)

Exclude all git submodules and their contents from being scanned.

Parameters
repo (Repository) – The repository being scanned

Return type
None

global_options: *GlobalOptions*

static header_length(*diff*)

Compute the length of the git diff header text.

Parameters
diff (str) – The diff being inspected for a header

Return type
int

hex_entropy_limit

Returns low entropy limit for suspicious hexadecimal encodings

property included_paths: List[Pattern]

Get a list of regexes used as an exclusive list of paths to scan

property issue_count: int

property issue_file: IO

property issues: List[*Issue*]

Get a list of issues found during the scan.

If the scan is still in progress, force it to complete first.

Returns
Any issues found during the scan.

load_issues()

Return type
Generator[*Issue*, None, None]

load_repo(*repo_path*)

Load and return the repository to be scanned.

Parameters
repo_path (str) – The local filesystem path pointing to the repository

Raises
types.GitLocalException – If there was a problem loading the repository

Return type
Repository

logger: Logger

repo_path: str

static rule_matches(*rule*, *string*, *line*, *path*)

Match string and path against rule.

Parameters

- **rule** (*Rule*) – Rule to perform match
- **string** (str) – string to match against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – path to match against rule path_pattern

Return type
bool

Returns
True if string and path matched, False otherwise.

property rules_regexes: Set[*Rule*]

Get a set of regular expressions to scan the code for.

Raises
types.ConfigException – If there was a problem compiling the rules

scan()

Run the requested scans against the target data.

This will iterate through all chunks of data as provided by the scanner implementation, and run all requested scans against it, as specified in *self.global_options*.

The scan method is thread-safe; if multiple concurrent scans are requested, the first will run to completion while other callers are blocked (after which they will each execute in turn, yielding cached issues without repeating the underlying repository scan).

Raises
types.ConfigException – If there were problems with the scanner's configuration

Return type
Generator[*Issue*, None, None]

scan_entropy(*chunk*)

Scan a chunk of data for apparent high entropy.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

scan_regex(*chunk*)

Scan a chunk of data for matches against the configured regexes.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

should_scan(*file_path*)

Check if the a file path should be included in analysis.

If non-empty, *self.included_paths* has precedence over *self.excluded_paths*, such that a file path that is not matched by any of the defined *self.included_paths* will be excluded, even when it is not matched by any of the defined *self.excluded_paths*. If either *self.included_paths* or *self.excluded_paths* are undefined or empty, they will have no effect, respectively. All file paths are included by this function when no inclusions or exclusions exist.

Parameters
file_path (str) – The file path to check for inclusion

Return type
bool

Returns
False if the file path is _not_ matched by *self.included_paths* (when non-empty) or if it is matched by *self.excluded_paths* (when non-empty), otherwise returns True

signature_is_excluded(*blob*, *file_path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **blob** (str) – The piece of data which is being scanned
- **file_path** (str) – The path and file name for the data being scanned

Return type
bool

store_issue(*issue*)

Return type
None

Parameters
issue (Issue) –

class tartufo.scanner.**GitRepoScanner**(*global_options*, *git_options*, *repo_path*)

Bases: *GitScanner*

Used for scanning a full clone of a git repository.

Parameters

- **global_options** (*GlobalOptions*) – The options provided to the top-level tartufo command
- **git_options** (*GitOptions*) – The options specific to interacting with a git repository
- **repo_path** (str) – The local filesystem path pointing to the repository

_iter_diff_index(*diff*)

Iterate over a "diff index", yielding the individual file changes.

A "diff index" is essentially analogous to a single commit in the git history. So what this does is iterate over a single commit, and yield the changes to each individual file in that commit, along with its file path. This will also check the file path and ensure that it has not been excluded from the scan by configuration.

Note that binary files are wholly skipped.

Parameters
diff (Diff) – The diff index / commit to be iterated over

Return type
Generator[Tuple[str, str], None, None]

b64_entropy_limit

Returns low entropy limit for suspicious base64 encodings

static calculate_entropy(*data*)

Calculate the Shannon entropy for a piece of data.

This essentially calculates the overall probability for each character in *data* to be to be present. By doing this, we can tell how random a string appears to be.

Adapted from http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html

Parameters
data (str) – The data to be scanned for its entropy

Return type
float

Returns
The amount of entropy detected in the data

property chunks: Generator[*Chunk*, None, None]

Yield individual diffs from the repository's history.

Raises
types.GitRemoteException – If there was an error fetching branches

property completed: bool

Return True if scan has completed

Returns
True if scan has completed; False if scan is in progress

compute_scaled_entropy_limit(*maximum_bitrate*)

Determine low entropy cutoff for specified bitrate

Parameters
maximum_bitrate (float) – How many bits does each character represent?

Return type
float

Returns
Entropy detection threshold scaled to the input bitrate

property config_data: MutableMapping[str, Any]

Supplemental configuration to be merged into the *_options information.

entropy_string_is_excluded(*string*, *line*, *path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **string** (str) – String to check against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – Path to check against rule path pattern

Return type
bool

Returns
True if excluded, False otherwise

evaluate_entropy_string(*chunk*, *line*, *string*, *min_entropy_score*)

Check entropy string using entropy characters and score.

Parameters

- **chunk** (*Chunk*) – The chunk of data to check
- **line** (str) – Source line containing string of interest
- **string** (str) – String to check
- **min_entropy_score** (float) – Minimum entropy score to flag

Return type
Generator[*Issue*, None, None]

Returns
Generator of issues flagged

property excluded_entropy: List[*Rule*]

Get a list of regexes used as an exclusive list of paths to scan.

property excluded_paths: List[Pattern]

Get a list of regexes used to match paths to exclude from the scan

excluded_signatures

Get a list of the signatures of findings to be excluded from the scan results.

Returns
The signatures to be excluded from scan results

filter_submodules(*repo*)

Exclude all git submodules and their contents from being scanned.

Parameters
repo (Repository) – The repository being scanned

Return type
None

git_options: *GitOptions*

global_options: *GlobalOptions*

static header_length(*diff*)

Compute the length of the git diff header text.

Parameters
diff (str) – The diff being inspected for a header

Return type
int

hex_entropy_limit

Returns low entropy limit for suspicious hexadecimal encodings

property included_paths: List[Pattern]

Get a list of regexes used as an exclusive list of paths to scan

property issue_count: int

property issue_file: IO

property issues: List[*Issue*]

Get a list of issues found during the scan.

If the scan is still in progress, force it to complete first.

Returns
Any issues found during the scan.

load_issues()

Return type
Generator[*Issue*, None, None]

load_repo(*repo_path*)

Load and return the repository to be scanned.

Parameters
repo_path (str) – The local filesystem path pointing to the repository

Raises
types.GitLocalException – If there was a problem loading the repository

Return type
Repository

logger: Logger

repo_path: str

static rule_matches(*rule*, *string*, *line*, *path*)

Match string and path against rule.

Parameters

- **rule** (*Rule*) – Rule to perform match
- **string** (str) – string to match against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – path to match against rule path_pattern

Return type
bool

Returns
True if string and path matched, False otherwise.

property rules_regexes: Set[*Rule*]

Get a set of regular expressions to scan the code for.

Raises
types.ConfigException – If there was a problem compiling the rules

scan()

Run the requested scans against the target data.

This will iterate through all chunks of data as provided by the scanner implementation, and run all requested scans against it, as specified in *self.global_options*.

The scan method is thread-safe; if multiple concurrent scans are requested, the first will run to completion while other callers are blocked (after which they will each execute in turn, yielding cached issues without repeating the underlying repository scan).

Raises
types.ConfigException – If there were problems with the scanner's configuration

Return type
Generator[*Issue*, None, None]

scan_entropy(*chunk*)

Scan a chunk of data for apparent high entropy.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

scan_regex(*chunk*)

Scan a chunk of data for matches against the configured regexes.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

should_scan(*file_path*)

Check if the a file path should be included in analysis.

If non-empty, *self.included_paths* has precedence over *self.excluded_paths*, such that a file path that is not matched by any of the defined *self.included_paths* will be excluded, even when it is not matched by any of the defined *self.excluded_paths*. If either *self.included_paths* or *self.excluded_paths* are undefined or empty, they will have no effect, respectively. All file paths are included by this function when no inclusions or exclusions exist.

Parameters
file_path (str) – The file path to check for inclusion

Return type
bool

Returns
False if the file path is _not_ matched by *self.included_paths* (when non-empty) or if it is matched by *self.excluded_paths* (when non-empty), otherwise returns True

signature_is_excluded(*blob*, *file_path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **blob** (str) – The piece of data which is being scanned
- **file_path** (str) – The path and file name for the data being scanned

Return type
bool

store_issue(*issue*)

Return type
None

Parameters
issue (Issue) –

class tartufo.scanner.**GitScanner**(*global_options*, *repo_path*)

Bases: *ScannerBase*, ABC

A base class for scanners looking at git history.

This is a lightweight base class to provide some basic functionality needed across all scanner that are interacting with git history.

Parameters

- **global_options** (*GlobalOptions*) – The options provided to the top-level tartufo command
- **repo_path** (str) – The local filesystem path pointing to the repository

_iter_diff_index(*diff*)

Iterate over a "diff index", yielding the individual file changes.

A "diff index" is essentially analogous to a single commit in the git history. So what this does is iterate over a single commit, and yield the changes to each individual file in that commit, along with its file path. This will also check the file path and ensure that it has not been excluded from the scan by configuration.

Note that binary files are wholly skipped.

Parameters
diff (Diff) – The diff index / commit to be iterated over

Return type
Generator[Tuple[str, str], None, None]

b64_entropy_limit

Returns low entropy limit for suspicious base64 encodings

static calculate_entropy(*data*)

Calculate the Shannon entropy for a piece of data.

This essentially calculates the overall probability for each character in *data* to be to be present. By doing this, we can tell how random a string appears to be.

Adapted from http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html

Parameters
data (str) – The data to be scanned for its entropy

Return type
float

Returns
The amount of entropy detected in the data

abstract property chunks: Generator[*Chunk*, None, None]

Yield "chunks" of data to be scanned.

Examples of "chunks" would be individual git commit diffs, or the contents of individual files.

property completed: bool

Return True if scan has completed

Returns
True if scan has completed; False if scan is in progress

compute_scaled_entropy_limit(*maximum_bitrate*)

Determine low entropy cutoff for specified bitrate

Parameters
maximum_bitrate (float) – How many bits does each character represent?

Return type
float

Returns
Entropy detection threshold scaled to the input bitrate

property config_data: MutableMapping[str, Any]

Supplemental configuration to be merged into the *_options information.

entropy_string_is_excluded(*string*, *line*, *path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **string** (str) – String to check against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – Path to check against rule path pattern

Return type
bool

Returns
True if excluded, False otherwise

evaluate_entropy_string(*chunk*, *line*, *string*, *min_entropy_score*)

Check entropy string using entropy characters and score.

Parameters

- **chunk** (*Chunk*) – The chunk of data to check
- **line** (str) – Source line containing string of interest
- **string** (str) – String to check
- **min_entropy_score** (float) – Minimum entropy score to flag

Return type
Generator[*Issue*, None, None]

Returns
Generator of issues flagged

property excluded_entropy: List[*Rule*]

Get a list of regexes used as an exclusive list of paths to scan.

property excluded_paths: List[Pattern]

Get a list of regexes used to match paths to exclude from the scan

excluded_signatures

Get a list of the signatures of findings to be excluded from the scan results.

Returns
The signatures to be excluded from scan results

filter_submodules(*repo*)

Exclude all git submodules and their contents from being scanned.

Parameters
repo (Repository) – The repository being scanned

Return type
None

global_options: *GlobalOptions*

static header_length(*diff*)

Compute the length of the git diff header text.

Parameters
diff (str) – The diff being inspected for a header

Return type
int

hex_entropy_limit

Returns low entropy limit for suspicious hexadecimal encodings

property included_paths: List[Pattern]

Get a list of regexes used as an exclusive list of paths to scan

property issue_count: int

property issue_file: IO

property issues: List[*Issue*]

Get a list of issues found during the scan.

If the scan is still in progress, force it to complete first.

Returns
Any issues found during the scan.

load_issues()

Return type
Generator[*Issue*, None, None]

abstract load_repo(*repo_path*)

Load and return the repository to be scanned.

Parameters
repo_path (str) – The local filesystem path pointing to the repository

Raises
types.GitLocalException – If there was a problem loading the repository

Return type
Repository

logger: Logger

repo_path: str

static rule_matches(*rule*, *string*, *line*, *path*)

Match string and path against rule.

Parameters

- **rule** (*Rule*) – Rule to perform match
- **string** (str) – string to match against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – path to match against rule path_pattern

Return type
bool

Returns
True if string and path matched, False otherwise.

property rules_regexes: Set[*Rule*]

Get a set of regular expressions to scan the code for.

Raises
types.ConfigException – If there was a problem compiling the rules

scan()

Run the requested scans against the target data.

This will iterate through all chunks of data as provided by the scanner implementation, and run all requested scans against it, as specified in *self.global_options*.

The scan method is thread-safe; if multiple concurrent scans are requested, the first will run to completion while other callers are blocked (after which they will each execute in turn, yielding cached issues without repeating the underlying repository scan).

Raises
types.ConfigException – If there were problems with the scanner's configuration

Return type
Generator[*Issue*, None, None]

scan_entropy(*chunk*)

Scan a chunk of data for apparent high entropy.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

scan_regex(*chunk*)

Scan a chunk of data for matches against the configured regexes.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

should_scan(*file_path*)

Check if the a file path should be included in analysis.

If non-empty, *self.included_paths* has precedence over *self.excluded_paths*, such that a file path that is not matched by any of the defined *self.included_paths* will be excluded, even when it is not matched by any of the defined *self.excluded_paths*. If either *self.included_paths* or *self.excluded_paths* are undefined or empty, they will have no effect, respectively. All file paths are included by this function when no inclusions or exclusions exist.

Parameters
file_path (str) – The file path to check for inclusion

Return type
bool

Returns
False if the file path is _not_ matched by *self.included_paths* (when non-empty) or if it is matched by *self.excluded_paths* (when non-empty), otherwise returns True

signature_is_excluded(*blob*, *file_path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **blob** (str) – The piece of data which is being scanned
- **file_path** (str) – The path and file name for the data being scanned

Return type
bool

store_issue(*issue*)

Return type
None

Parameters
issue (Issue) –

class tartufo.scanner.**Issue**(*issue_type*, *matched_string*, *chunk*)

Bases: object

Represent an issue found while scanning a target.

Parameters

- **issue_type** (*IssueType*) – What type of scan identified this issue
- **matched_string** (str) – The string that was identified as a potential issue
- **chunk** (*Chunk*) – The chunk of data where the match was found

OUTPUT_SEPARATOR: str = '~~~~~~~~~~~~~~~~~~~~~'

as_dict(*compact=False*)

Return a dictionary representation of an issue.

This is primarily meant to aid in JSON serialization.

Parameters
compact – True to return a dictionary with fewer fields.

Return type
Dict[str, Optional[str]]

Returns
A JSON serializable dictionary representation of this issue

chunk: *Chunk*

issue_detail: Optional[str]

issue_type: *IssueType*

matched_string: str

property signature: str

Generate a stable hash-based signature uniquely identifying this issue.

class tartufo.scanner.**ScannerBase**(*options*)

Bases: ABC

Provide the base, generic functionality needed by all scanners.

In fact, this contains all of the actual scanning logic. This part of the application should never differ; the part that differs, and the part that is left abstract here, is what content is provided to the various scans. For this reason, the *chunks* property is left abstract. It is up to the various scanners to implement this property, in the form of a generator, to yield all the individual pieces of content to be scanned.

Parameters
options (*GlobalOptions*) – A set of options to control the behavior of the scanner

b64_entropy_limit

Returns low entropy limit for suspicious base64 encodings

static calculate_entropy(*data*)

Calculate the Shannon entropy for a piece of data.

This essentially calculates the overall probability for each character in *data* to be to be present. By doing this, we can tell how random a string appears to be.

Adapted from http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html

Parameters
data (str) – The data to be scanned for its entropy

Return type
float

Returns
The amount of entropy detected in the data

abstract property chunks: Generator[*Chunk*, None, None]

Yield "chunks" of data to be scanned.

Examples of "chunks" would be individual git commit diffs, or the contents of individual files.

property completed: bool

Return True if scan has completed

Returns
True if scan has completed; False if scan is in progress

compute_scaled_entropy_limit(*maximum_bitrate*)

Determine low entropy cutoff for specified bitrate

Parameters
maximum_bitrate (float) – How many bits does each character represent?

Return type
float

Returns
Entropy detection threshold scaled to the input bitrate

property config_data: MutableMapping[str, Any]

Supplemental configuration to be merged into the *_options information.

entropy_string_is_excluded(*string*, *line*, *path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **string** (str) – String to check against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – Path to check against rule path pattern

Return type
bool

Returns
True if excluded, False otherwise

evaluate_entropy_string(*chunk*, *line*, *string*, *min_entropy_score*)

Check entropy string using entropy characters and score.

Parameters

- **chunk** (*Chunk*) – The chunk of data to check
- **line** (str) – Source line containing string of interest
- **string** (str) – String to check
- **min_entropy_score** (float) – Minimum entropy score to flag

Return type
Generator[*Issue*, None, None]

Returns
Generator of issues flagged

property excluded_entropy: List[*Rule*]

Get a list of regexes used as an exclusive list of paths to scan.

property excluded_paths: List[Pattern]

Get a list of regexes used to match paths to exclude from the scan

excluded_signatures

Get a list of the signatures of findings to be excluded from the scan results.

Returns
The signatures to be excluded from scan results

global_options: *GlobalOptions*

hex_entropy_limit

Returns low entropy limit for suspicious hexadecimal encodings

property included_paths: List[Pattern]

Get a list of regexes used as an exclusive list of paths to scan

property issue_count: int

property issue_file: IO

property issues: List[*Issue*]

Get a list of issues found during the scan.

If the scan is still in progress, force it to complete first.

Returns
Any issues found during the scan.

load_issues()

Return type
Generator[*Issue*, None, None]

logger: Logger

static rule_matches(*rule*, *string*, *line*, *path*)

Match string and path against rule.

Parameters

- **rule** (*Rule*) – Rule to perform match
- **string** (str) – string to match against rule pattern
- **line** (str) – Source line containing string of interest
- **path** (str) – path to match against rule path_pattern

Return type
bool

Returns
True if string and path matched, False otherwise.

property rules_regexes: Set[*Rule*]

Get a set of regular expressions to scan the code for.

Raises
types.ConfigException – If there was a problem compiling the rules

scan()

Run the requested scans against the target data.

This will iterate through all chunks of data as provided by the scanner implementation, and run all requested scans against it, as specified in *self.global_options*.

The scan method is thread-safe; if multiple concurrent scans are requested, the first will run to completion while other callers are blocked (after which they will each execute in turn, yielding cached issues without repeating the underlying repository scan).

Raises
types.ConfigException – If there were problems with the scanner's configuration

Return type
Generator[*Issue*, None, None]

scan_entropy(*chunk*)

Scan a chunk of data for apparent high entropy.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

scan_regex(*chunk*)

Scan a chunk of data for matches against the configured regexes.

Parameters
chunk (*Chunk*) – The chunk of data to be scanned

Return type
Generator[*Issue*, None, None]

should_scan(*file_path*)

Check if the a file path should be included in analysis.

If non-empty, *self.included_paths* has precedence over *self.excluded_paths*, such that a file path that is not matched by any of the defined *self.included_paths* will be excluded, even when it is not matched by any of the defined *self.excluded_paths*. If either *self.included_paths* or *self.excluded_paths* are undefined or empty, they will have no effect, respectively. All file paths are included by this function when no inclusions or exclusions exist.

Parameters
file_path (str) – The file path to check for inclusion

Return type
bool

Returns
False if the file path is _not_ matched by *self.included_paths* (when non-empty) or if it is matched by *self.excluded_paths* (when non-empty), otherwise returns True

signature_is_excluded(*blob*, *file_path*)

Find whether the signature of some data has been excluded in configuration.

Parameters

- **blob** (str) – The piece of data which is being scanned
- **file_path** (str) – The path and file name for the data being scanned

Return type
bool

store_issue(*issue*)

Return type
None

Parameters
issue (Issue) –

3.10.3 tartufo.types

exception tartufo.types.**BranchNotFoundException**

Raised if a branch was not found

class tartufo.types.**Chunk**(*contents*, *file_path*, *metadata*, *is_diff*)

A single "chunk" of text to be inspected during a scan

Parameters

- **contents** (str) – The actual text contents of the chunk
- **file_path** (str) – The file path that is being inspected
- **metadata** (Dict[str, Any]) – Commit/file metadata for the chunk being inspected
- **is_diff** (bool) – True if contents is diff output (vs raw data)

exception tartufo.types.**ConfigException**

Raised if there is a problem with the configuration

exception tartufo.types.**GitException**

Raised if there is a problem interacting with git

exception tartufo.types.**GitLocalException**

Raised if there is an error interacting with a local git repository

class tartufo.types.**GitOptions**(*since_commit*, *max_depth*, *branch*, *include_submodules*, *progress*)

Configuration options specific to git-based scans

Parameters

- **since_commit** (Optional[str]) – A commit hash to treat as a starting point in history for the scan
- **max_depth** (int) – A maximum depth, or maximum number of commits back in history, to scan
- **branch** (Optional[str]) – A specific branch to scan
- **include_submodules** (bool) – Whether to also scan submodules of the repository
- **progress** (*bool*) –

exception tartufo.types.**GitRemoteException**

Raised if there is an error interacting with a remote git repository

class tartufo.types.**GlobalOptions**(*rule_patterns*, *default_regexes*, *entropy*, *regex*, *scan_filenames*, *include_path_patterns*, *exclude_path_patterns*, *exclude_entropy_patterns*, *exclude_signatures*, *output_dir*, *temp_dir*, *buffer_size*, *git_rules_repo*, *git_rules_files*, *config*, *verbose*, *quiet*, *log_timestamps*, *output_format*, *entropy_sensitivity*)

Configuration options for controlling scans and output

Parameters

- **rule_patterns** (Tuple[Dict[str, str], ...]) – Dictionaries containing regex patterns to match against
- **default_regexes** (bool) – Whether to include built-in regex patterns in the scan
- **entropy** (bool) – Whether to enable entropy scans

- **regex** (bool) – Whether to enable regular expression scans
- **scan_filenames** (bool) – Whether to check filenames for potential secrets
- **include_path_patterns** (Tuple[Dict[str, str], ...]) – An exclusive list of paths to be scanned
- **exclude_path_patterns** (Tuple[Dict[str, str], ...]) – A list of paths to be excluded from the scan
- **exclude_entropy_patterns** (Tuple[Dict[str, str], ...]) – Patterns to be excluded from entropy matches
- **exclude_signatures** (Tuple[Dict[str, str], ...]) – Signatures of previously found findings to be excluded from the list of current findings
- **exclude_findings** – Signatures of previously found findings to be excluded from the list of current findings
- **output_dir** (Optional[str]) – A directory where detailed findings results will be written
- **temp_dir** (Optional[str]) – A directory where temporary files will be written
- **buffer_size** (int) – Maximum number of issues that will be buffered on the heap
- **git_rules_repo** (Optional[str]) – A remote git repository where additional rules can be found
- **git_rules_files** (Tuple[str, ...]) – The files in the remote rules repository to load the rules from
- **config** (Optional[TextIO]) – A configuration file from which default values are pulled
- **verbose** (int) – How verbose the scanner should be with its logging
- **quiet** (bool) – Whether to suppress all output
- **log_timestamps** (bool) – Whether to include timestamps in log output
- **output_format** (Optional[*OutputFormat*]) – What format should be output from the scan
- **entropy_sensitivity** (int) – A number from 0 - 100 representing the sensitivity of entropy scans. A value of 0 will detect totally non-random values, while a value of 100 will detect only wholly random values.

class tartufo.types.**IssueType**(*value*)

The method by which an issue was detected

class tartufo.types.**LogLevel**(*value*)

The various Python logging levels

class tartufo.types.**MatchType**(*value*)

What regex method to use when looking for a match

class tartufo.types.**OutputFormat**(*value*)

The formats in which tartufo is able to output issue summaries

class tartufo.types.**Rule**(*name*, *pattern*, *path_pattern*, *re_match_type*, *re_match_scope*)

A regular expression rule to be used for inspecting text during a scan

Parameters

- **name** (Optional[str]) – A unique name for the rule

- **pattern** (Pattern) – The regex pattern to be used by the pattern
- **path_pattern** (Optional[Pattern]) – A regex pattern to match against the file path(s)
- **re_match_type** (*MatchType*) – What type of regex operation to perform
- **re_match_scope** (Optional[*Scope*]) – What scope to perform the match against

exception tartufo.types.**ScanException**

Raised if there is a problem encountered during a scan

class tartufo.types.**Scope**(*value*)

The scope to search for a regex match

exception tartufo.types.**TartufoException**

Base class for all package exceptions

3.10.4 tartufo.util

tartufo.util.**clone_git_repo**(*git_url*, *target_dir=None*)

Clone a remote git repository and return its filesystem path.

Parameters

- **git_url** (str) – The URL of the git repository to be cloned
- **target_dir** (Optional[Path]) – Where to clone the repository to

Return type
Tuple[Path, str]

Returns
Filesystem path of local clone and name of remote source

Raises
types.GitRemoteException – If there was an error cloning the repository

tartufo.util.**del_rw**(*_func*, *name*, *_exc*)

Attempt to grant permission to and force deletion of a file.

This is used as an error handler for *shutil.rmtree*.

Parameters

- **_func** (Callable) – The original calling function
- **name** (str) – The name of the file to try removing
- **_exc** (Exception) – The exception raised originally when the file was removed

Return type
None

tartufo.util.**echo_result**(*options*, *scanner*, *repo_path*, *output_dir*)

Print all found issues out to the console, optionally as JSON.

Parameters

- **options** (*GlobalOptions*) – Global options object
- **scanner** (*ScannerBase*) – ScannerBase containing issues and excluded paths from config tree
- **repo_path** (str) – The path to the repository the issues were found in

- **output_dir** (Optional[Path]) – The directory that issue details were written out to

Return type
None

tartufo.util.**extract_commit_metadata**(*commit*, *branch_name*)

Grab a consistent set of metadata from a git commit, for user output.

Parameters

- **commit** (Commit) – The commit to extract the data from
- **branch_name** (str) – What branch the commit was found on

Return type
Dict[str, Any]

tartufo.util.**fail**(*msg*, *ctx*, *code=1*)

Print out a styled error message and exit.

Parameters

- **msg** (str) – The message to print out to the user
- **ctx** (Context) – A context from a currently executing Click command
- **code** (int) – The exit code to use; must be >= 1

Return type
NoReturn

tartufo.util.**find_strings_by_regex**(*text*, *regex*, *threshold=20*)

Locate strings ("words") of interest in input text

Each returned string must have a length, at minimum, equal to *threshold*. This is meant to return longer strings which are likely to be things like auto-generated passwords, tokens, hashes, etc.

Parameters

- **text** (str) – The text string to be analyzed
- **regex** (Pattern) – A pattern which matches all character sequences of interest
- **threshold** (int) – The minimum acceptable length of a matching string

Return type
Generator[str, None, None]

tartufo.util.**generate_signature**(*snippet*, *filename*)

Generate a stable hash signature for an issue found in a commit.

These signatures are used for configuring excluded/approved issues, such as secrets intentionally embedded in tests.

Parameters

- **snippet** (str) – A string which was found as a potential issue during a scan
- **filename** (str) – The file where the issue was found

Return type
str

tartufo.util.**is_shallow_clone**(*repo*)

Determine whether a repository is a shallow clone

This is used to work around https://github.com/libgit2/libgit2/issues/3058 Basically, any time a git repository is a "shallow" clone (it was cloned with *–max-depth N*), git will create a file at *.git/shallow*. So we simply need to test whether that file exists to know whether we are interacting with a shallow repository.

Parameters
repo (Repository) – The repository to check for "shallowness"

Return type
bool

tartufo.util.**path_contains_git**(*path*)

Determine whether a filesystem path contains a git repository.

Parameters
path (str) – The fully qualified path to be checked

Return type
bool

tartufo.util.**process_issues**(*repo_path*, *scan*, *options*)

Handle post-scan processing/reporting of a batch of issues.

Parameters

- **repo_path** (str) – The repository that was scanned
- **scan** (*ScannerBase*) – The scanner that performed the scan
- **options** (*GlobalOptions*) – The options to use for determining output

Return type
None

tartufo.util.**write_outputs**(*issues*, *output_dir*)

Write details of the issues to individual files in the specified directory.

Parameters

- **found_issues** – A list of issues to be written out
- **output_dir** (Path) – The directory where the files should be written
- **issues** (*Generator[Issue, None, None]*) –

Return type
List[str]

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

W