vault-redirector Documentation

Release 0.2.0

Manheim

June 15, 2016

Contents

Python/Twisted application to redirect Hashicorp Vault client requests to the active node in a HA cluster. API Documentation: http://vault-redirector-twisted.readthedocs.org/en/latest/

CHAPTER 1

Status

This application is currently very young. Please ensure it meets your needs before using it in production.

NOTE: My initial plan was to implement this in Go. My Go knowledge is severely lacking, and the performance of Python/Twisted at 1,000 requests per second is within 25% of the Go variant. Please consider this package to be temporary, until work on the Go version (https://github.com/manheim/vault-redirector) continues.

CHAPTER 2

Purpose

There's a bit of a gap in usability of Vault in a High Availability mode, at least in AWS:

- Vault's HA architecture is based on an active/standby model; only one server can be active at a time, and any others are standby. Standby servers respond to all API requests with a 307 Temporary Redirect to the Active server, but can only do this if they're unsealed (in the end of the HA docs: "It is important to note that only unsealed servers act as a standby. If a server is still in the sealed state, then it cannot act as a standby as it would be unable to serve any requests should the active server fail.").
- HashiCorp recommends managing infrastructure individually, i.e. not in an auto-scaling group. In EC2, if you want to run Consul on the same nodes, this is an absolute requirement as Consul requires static IP addresses in order for disaster recovery to work without downtime and manual changes.

As a result, we're left with a conundrum:

1. We can't put Vault behind an Elastic Load Balancer, because that would cause all API requests to appear to have the ELB's source IP address. Not only does this render any of the IP- or subnet-based authorization methods unusable, but it also means we lose client IPs in the audit logs (which is likely a deal-breaker for anyone concerned with security).
2. The only remaining option for HA, at least in AWS, is to use Route53 round-robin DNS records that have the IPs of all of the cluster members. This poses a problem because if one node in an N-node cluster is either offline or sealed, approximately 1/N of all client requests will be directed to that node and fail.

While it would be good for all clients to automatically retry these requests, it appears that most client libraries (and even the `vault` command line client) do not currently support this. While retry logic would certainly be good to implement in any case, it adds latency to retrieving secrets (in the common case where the cluster is reachable, but some nodes are down) and also does not account for possible DNS caching issues. Furthermore, we're providing Vault as a service to our organization; relying on retries would mean either adding retry logic to every Vault client library and getting those changes merged, or deviating from our plan of "here's your credentials and endpoint, see the upstream docs for your language's client library."

The best solution to this problem would be for Vault issue #799, a request to add PROXY Protocol support to Vault, to be completed. Both AWS ELBs and HAProxy support this, and it would alleviate issue #1 above, allowing us to run Vault behind a load balancer but still have access to the original client IP address.

This small service is intended to provide an interim workaround until that solution is implemented.

CHAPTER 3

Functionality

We take advantage of Vault's 307 redirects (and the assumption that any protocol-compliant client library will honor them). Instead of connecting directly to the Vault service, clients connect to a load-balanced daemon running on the Vault nodes. This daemon asynchronously polls Consul for the health status of the Vault instances, and therefore knows the currently-active Vault instance at all times. All incoming HTTP(S) requests are simply 307 redirected to the active instance. As this service can safely be load balanced, it will tolerate failed nodes better than round-robin DNS. Since it redirects the client to the active node, the client's IP address will be properly seen by Vault.

CHAPTER 4

Requirements

1. Python 2.7, 3.3, 3.4 or 3.5 and `pip`; virtualenv is recommended.
2. `gcc` or another suitable C compiler and the python headers/development package for your OS; the `twisted` package has dependencies with native extensions which must be compiled at install time.
3. The twisted package (will be installed automatically via `pip`).
4. The requests package (will be installed automatically via `pip`).
5. If you wish to use TLS for incoming connections (highly recommended), the pyOpenSSL and pem packages. These can be automatically installed along with vault-redirector by using `pip install vault-redirector[tls]`.
6. Consul running and configured with service checks for Vault (see below)
7. Vault 0.6.0+

4.1 Consul Service Checks

In order to determine the active Vault instance, `vault-redirector` requires that Consul be running and monitoring the health of all Vault instances. Redirection can be to either the IP address or Consul node name running the active service.

The current implementation in this package requires Vault 0.6.0+, which automatically registers its own service and health checks with Consul.

CHAPTER 5

Installation

We recommend installing inside an isolated virtualenv. If you don't want to do that and would rather install system-wide, simply skip to the last two steps:

1. Ensure that `gcc` or another suitable C compiler is installed.
2. `virtualenv vault`
3. `source vault/bin/activate`
4. `pip install vault-redirector`; if you wish to use TLS for incoming connections (highly recommended) you'll also need the `pyOpenSSL` and `pem` packages, which will be installed automatically if you instead run `pip install vault-redirector[tls]`

CHAPTER 6

Usage

6.1 Command Line Usage

All options and configuration are passed in via command-line options.

```
jantman@exodus$ vault-redirector -h
usage: vault-redirector [-h] [-v] [-l] [-V] [-S] [-I] [-p POLL_INTERVAL]
                        [-P BIND_PORT] [-C CHECKID] [-c CERT_PATH]
                        [-k KEY_PATH]
                        CONSUL_HOST_PORT

Python/Twisted application to redirect Hashicorp Vault client requests to the
active node in a HA cluster

positional arguments:
  CONSUL_HOST_PORT      Consul address in host:port form

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         verbose output. specify twice for debug-level output.
                        See also -l|--log-enable
  -l, --log-disable     If specified, disable ALL logging after initial setup.
                        This can be changed at runtime via signals
  -V, --version         show program's version number and exit
  -S, --https           Redirect to HTTPS scheme instead of plain HTTP.
  -I, --ip              redirect to active node IP instead of name
  -p POLL_INTERVAL, --poll-interval POLL_INTERVAL
                        Consul service health poll interval in seconds
                        (default 5.0)
  -P BIND_PORT, --port BIND_PORT
                        Port number to listen on (default 8080)
  -C CHECKID, --checkid CHECKID
                        Consul service CheckID for Vault (default:
                        "service:vault"
  -c CERT_PATH, --cert-path CERT_PATH
                        Path to PEM-encoded TLS certificate. If you need a
                        certificate chain to verify trust, this file should be
                        composed of the server certificate followed by one or
                        more chain certificates. If specified, you must also
                        specify -k|--key-path
  -k KEY_PATH, --key-path KEY_PATH
                        Path to PEM-encoded TLS private key. If specified, you
                        must also specify -c|--cert-path
```

By default, `vault-redirector` will redirect clients to the hostname (Consul health check **node name**) of the active Vault node, over plain HTTP. This can be changed via the `-I | --ip` and `-S | --https` options.

6.2 Usage with TLS

vault-redirector is capable of listening with TLS/HTTPS, which is strongly recommended as clients will send their authentication tokens as HTTP headers. To do so, specify the `-k|--key-path` and `-c|--cert-path` options with the paths to the key and certificate files, respectively. Each should be a PEM-encoded file; if your certificate requires a trust/CA chain to be presented to clients, the file at `cert-path` should be a combined certificate and chain file, composed of the PEM-encoded certificate concatenated with one or more PEM-encoded chain certificates.

6.3 Running as a Daemon / Service

For anything other than testing, `vault-redirector` should be run as a system service. There is no built-in daemonizing support; this is left up to your operating system.

Here is an example systemd service unit file for `vault-redirector`, assuming you wish to run it as a `vaultredirector` user and group, and it is installed into a virtualenv at `/usr/local/vault-redirector`, and Consul is running on localhost (127.0.0.1) on port 8500. This will start the service with logging disabled (`-l`) but set to INFO level (`-v`); logging can be turned on with SIGUSR1 as described below.

```
[Unit]
Description=Vault Redirector
Requires=basic.target network.target
After=basic.target network.target

[Service]
Type=simple
User=vaultredirector
Group=vaultredirector
PrivateDevices=yes
PrivateTmp=yes
ProtectSystem=full
ProtectHome=read-only
CapabilityBoundingSet=
NoNewPrivileges=yes
ExecStart=/usr/local/vault-redirector/bin/vault-redirector -v -l 127.0.0.1:8500
RestartSec=5s
TimeoutStopSec=30s
Restart=always
# disable all rate limiting; let it restart forever
StartLimitInterval=0

[Install]
WantedBy=multi-user.target
```

If you wish to both use TLS for incoming connections and redirect to a HTTPS URL for Vault, the `ExecStart` line would be:

```
ExecStart=/usr/local/vault-redirector/bin/vault-redirector -v -l -S --cert-path=/path/to/server.crt -
```

6.4 Health Check

Vault-redirector will respond to a request path of /vault-redirector-health with a JSON body something like the following; this can be used for load balancer health checks. If the active vault instance is known, the HTTP status code will be 200. Otherwise (i.e. if there is no active vault node or if Consul is unreachable) it will be a 503.

```
{
  "healthy": true,
  "application": "vault-redirector",
  "version": "0.1.0",
  "consul_host_port": "127.0.0.1:8500",
  "source": "https://github.com/manheim/vault-redirector-twisted",
  "active_vault": "vault_hostname_or_ip:port",
  "last_consul_poll": "YYYY-MM-DDTHH:MM:SS"
}
```

CHAPTER 7

Logging and Debugging

Python's logging framework can impose a slight performance penalty even for messages which are below the level set to be displayed (simple testing reports 10x execution time for logging to a level below what's set, vs guarding the log statements with a conditional). As a result, in addition to Python's normal logging verbosity levels, all logging statements after initial setup are guarded by a global "logging enabled" boolean; if logging is not enabled, the calls to Python's logging framework will never be made. This behavior can be enabled by running the process with the `-l` or `--log-disable` options (which is the recommended production configuration).

Note that this functionality is completely separate from the logging module's levels, which are controlled by the `-v` / `-vv` options (and are not currently changeable at runtime).

At any time, logging can be enabled by sending SIGUSR1 to the process, or disabled by sending SIGUSR2 to the process.

CHAPTER 8

Support

Please open any issues or feature requests in the manheim/vault-redirector-twisted GitHub issue tracker They will be dealt with as time allows. Please include as much detail as possible, including your version of `vault-redirector` and the Python version and OS/distribution it's running on, as well as the command line arguments used when running it. Debug-level logs will likely be very helpful.

CHAPTER 9

Development

Pull requests are welcome. Please cut them against the `master` branch of the manheim/vault-redirector-twisted repository.

It is expected that test coverage increase or stay the same, that all tests pass, that any new code have complete test coverage, and that code conforms to pep8 and passes pyflakes.

After making any changes to the code, before submitting a pull request, run `tox -e docs` to regenerate the API documentation. Commit any changes to the auto-generated files under `docs/source`.

9.1 Installing for Development

1. Fork the manheim/vault-redirector-twisted repository on GitHub.
2. Clone your fork somewhere on your local machine and `cd` to the clone:

```
$ git clone git@github.com:YOUR-GITHUB-USER/vault-redirector-twisted.git
$ cd vault-redirector-twisted
```

3. Add the manheim upstream repository as a git upstream, so you can pull in upstream changes, and fetch it:

```
$ git remote add upstream https://github.com/manheim/vault-redirector-twisted.git
$ git fetch upstream
```

4. Create a virtualenv for testing and running vault-redirector, install your local source into it, and install `tox` for testing:

```
$ virtualenv .
$ source bin/activate
$ pip install -e .
$ pip install tox pyOpenSSL pem
```

5. Check out a new git branch. If you're working on a GitHub issue you opened, your branch should be called "issues/N" where N is the issue number.

9.2 Testing

Testing is accomplished via pytest and tox. By default tests will be run for Python 2.7, 3.3, 3.4. 3.5 and the documentation. Each supported Python interpreter has two test suites, `unit` and `acceptance`. The `acceptance` suite will actually run vault redirector bound to an available port (but with the Consul active node query code mocked out) and make example HTTP requests against it.

To run the tests locally, with your virtualenv activated, run `tox -e py<version>-(unit|acceptance)` where `<version>` is one of the Python versions in `tox.ini` (i.e. "27", "33", "34" or "35"). You will need to already have the appropriate Python interpreter version installed on your system. When the tests are run locally, coverage reports will be generated in the `htmlcov/` directory.

To generate documentation locally, run `tox -e docs`; the HTML output will be in `docs/build/html`. This must be done after making any code changes, and any changes to the auto-generated files under `docs/source/` must be committed.

Automated testing is accomplished via TravisCI (it's free for any open source project). If you have a TravisCI account linked to your GitHub, you should be able to add your fork for automated testing without any changes to the repository.

CHAPTER 10

Maintenance

Instructions for repository maintainers follow:

10.1 Fixing Issues / Making Changes

Note that all commit messages should be of the form `issue #<ISSUE_NUM>: <descriptive message>`. When you've verified that the issue is fixed and update `CHANGES.rst`, your final commit message should be of the form `fixes #<ISSUE_NUM>: <descriptive message>`.

1. Follow the instructions above for installing for development.
2. Cut a new branch named after the GitHub issue ("issues/ISSUE_NUMBER").
3. Make your code changes as needed, and write or update tests. It's preferred that you commit early and often, to make it easier to isolate work that needs improvements.
4. Run tests locally at least for py27 and py35: `tox -e py27-unit,py27-acceptance,py35-unit,py35-acceptanc`
5. Examine the test results and the coverage reports in `htmlcov/` (the reports will be written for the last-run unit test suite). Iterate until you have full coverage and passing tests.
6. Run `tox -e docs` to generate documentation locally. Examine it for correctness, and commit any changes to the auto-generated files under `docs/source/`.
7. Update `CHANGES.rst` with a description of your change and a link to the GitHub issue. Commit that.
8. Push your branch to origin. If you believe it's ready, open a pull request for it.

10.2 Handling PRs

1. Ensure that all Travis tests are passing for the PR, and that code coverage is still 100% (for all Python versions).
2. Check out the pull request locally. To do this simply, you can edit `.git/config` in your clone of the repository, and under the `[remote "origin"]` section add the following lines. Then `git fetch origin` and you can check out PRs locally like `git checkout refs/pull/origin/PR_NUM`. Note that this will be read-only.

```
fetch = +refs/pull/*/head:refs/pull/origin/*
```

3. Run `tox -e docs` and ensure there are no changes to the auto-generated docs and that they look correct.
4. Ensure there is an appropriate `CHANGES.rst` entry for the changes.

5. Ensure that `README.rst`, if it has been changed, renders correctly on GitHub.
6. If there are any changes to the local repository, cut a new branch locally, commit them, and push it to your fork. You can either ask the original PR author to pull in your changes, or you can close their PR and open a new one for your branch (be sure to reference the closed PR in a comment).
7. Merge the PR to master.

10.3 Release Process

1. Open an issue for the release; cut a branch off `master` for that issue.
2. Build docs (`tox -e docs`) and ensure they look correct. Commit any changes to the auto-generated files.
3. Ensure that Travis tests are passing in all environments.
4. Ensure that test coverage is no less than the last release (ideally, 100%).
5. Ensure there are entries in `CHANGES.rst` for all changes since the last release, and that they link to the GitHub issues.
6. Increment the version number in `vault_redirector/version.py` and add version and release date to CHANGES.rst. Mention the issue in the commit for this, and push to GitHub.
7. Confirm that README.rst renders correctly on GitHub.
8. Upload package to testpypi and confirm that README.rst renders correctly.
    - Make sure your ~/.pypirc file is correct (a repo called `test` for https://testpypi.python.org/pypi) and that you have `twine` installed in your virtualenv. Then:
    - `rm -Rf dist`
    - `python setup.py register -r https://testpypi.python.org/pypi`
    - `python setup.py sdist bdist_wheel`
    - `twine upload -r test dist/*`
    - Check that the README renders at https://testpypi.python.org/pypi/vault-redirector
9. Create a pull request for the release to be merge into master. Upon successful Travis build, merge it.
10. Tag the release in Git, push tag to GitHub:

- tag the release. for now the message is quite simple: `git tag -a X.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
- push the tag to GitHub: `git push origin X.Y.Z`

11. Upload package to live pypi:
    - `twine upload dist/*`
12. make sure any GH issues fixed in the release were closed.

CHAPTER 11

Contents

11.1 vault_redirector

11.1.1 vault_redirector package

Submodules

vault_redirector.redirector module

vault_redirector.runner module

vault_redirector.version module

11.2 Changelog

11.2.1 0.2.0 (2016-06-15)

- Breaking change to how the active node is determined from Consul. Prior to this version, we looked for a 'vault' service with a health check named 'service:vault' (configurable via the VaultRedirector class constructor, or the `-c | --checkid` command line argument) that was passing. With Vault 0.6.0's automatic registration of service and health checks in Consul, this needs to change. The logic to find the active node now looks for a node in Consul that has the 'vault' service and a tag of 'active'.

11.2.2 0.1.1 (2016-04-21)

- issue #2 * add timestamp of last active Vault update to health status output as `last_consul_poll` key * fix critical issue where active node did not update if logging was disabled

11.2.3 0.1.0 (2016-04-15)

- initial release

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

12.1 License

vault-redirector is licensed under the MIT license; see `LICENSE` for the text of the license.

V

V