PATHspider Documentation

Release 0.9.0

the pathspider authors

July 06, 2016

Contents

Contents:

CHAPTER 1

Introduction

pathspider spiders paths

CHAPTER 2

Using PATHspider

2.1 Quickstart

2.1.1 Dependencies

PATHspider is a command line tool. If you have installed PATHspider from a package manager (e.g. apt or pip), you will already have all the dependencies you need installed.

If you are working from the source distribution (e.g. cloned git repository) then you will need to install some dependencies. On Debian GNU/Linux:

```
# sudo apt install python3-libtrace python3-twisted python3-zope.interface
```

In order to build the documentation from source, you will also need the following dependencies:

```
# sudo apt install python3-sphinx python3-repoze.sphinx.autointerface
```

2.1.2 Usage

You can run PATHspider from the command line. In order for the Observer to work, you will need permissions to capture raw packets from the network interface. If you've installed from apt then either the executable in /usr/bin will have been setuid or will have filesystem permissions set.

Note: If you're running from the source distribution, you will need to execute pathspider as:

```
# sudo /usr/bin/env PYTHONPATH=. python3 pathspider/run.py [...]
```

```
# pathspider -h
usage: run.py [-h] [-s] [-l] [-p PLUGIN] [-i INTERFACE] [-w WORKER_COUNT]
              INPUTFILE OUTPUTFILE

Pathspider will spider the paths.

positional arguments:
  INPUTFILE             a file containing a list of remote hosts to test, with
                        any accompanying metadata expected by the pathspider
                        test. this file should be formatted as a comma-
                        seperated values file.
  OUTPUTFILE            the file to output results data to
```

```
optional arguments:
  -h, --help            show this help message and exit
  -s, --standalone      run in standalone mode. this is the default mode (and
                        currently the only supported mode). in the future,
                        mplane will be supported as a mode of operation.
  -l, --list-plugins    print the list of installed plugins
  -p PLUGIN, --plugin PLUGIN
                        use named plugin
  -i INTERFACE, --interface INTERFACE
                        the interface to use for the observer
  -w WORKER_COUNT, --worker-count WORKER_COUNT
                        number of workers to use
```

2.1.3 Example

You can run a small study using ECNSpider and the included *webinput.csv* file to measure path transparency to ECN for a small selection of web servers:

```
# pathspider -i eth0 -w 10 examples/webinput.csv /tmp/results.txt
```

Note: The location of the example input file may be different if you've installed pathspider from a package manager. On Debian systems it is installed as */usr/share/doc/pathspider/examples/webinput.csv*.

CHAPTER 3

Architecture

The PATHspider architecture has four components, illustrated in the diagram below: the *configurator*, the workers, the *observer* and the merger. Each component is implemented as one or more threads, launched when PATHspider starts.

For each target hostname and/or address, with port numbers where appropriate, PATHspider enqueues a job, to be distributed amongst the worker threads when available. Each worker performs one connection with the "A" configuration and one connection with the "B" configuration. The "A" configuration will always be connected first and serves as the base line measurement, followed by the "B" configuration. This allows detection of hosts that do not respond rather than failing as a result of using a particular transport protocol or extension. These sockets remain open for a post-connection operation.

Some transport options require a system-wide parameter change, for example enabling ECN in the Linux kernel. This requires locking and synchronisation. Using semaphores, the configurator waits for each worker to complete an operation and then changes the state to perform the next batch of operations. This process cycles continually until no more jobs remain. In a typical experiment, multiple workers (on the order of hundreds) are active, since much of the time in a connection test is spent waiting for an answer for the target or a timeout to fire.

In addition, packets are separately captured for analysis by the observer using Python bindings for libtrace. First, the observer assigns each incoming packet to a flow based on the source and destination addresses, as well as the TCP, UDP or SCTP ports when available. The packet and its associated flow are then passed to a function chain. The functions in this chain may be simple functions, such as counting the number of packets or octets seen for a flow, or more complex functions, such as recording the state of flags within packets and analysis based on previously observed packets in the flow. For example, a function may record both an ECN negotiation attempt and whether the host successfully negotiated use of ECN.

A function may alert the observer that a flow should have completed and that the flow information can be matched with the corresponding job record and passed to the merger. The merger extracts the fields needed for a particular measurement campaign from the records produced by the worker and the observer.

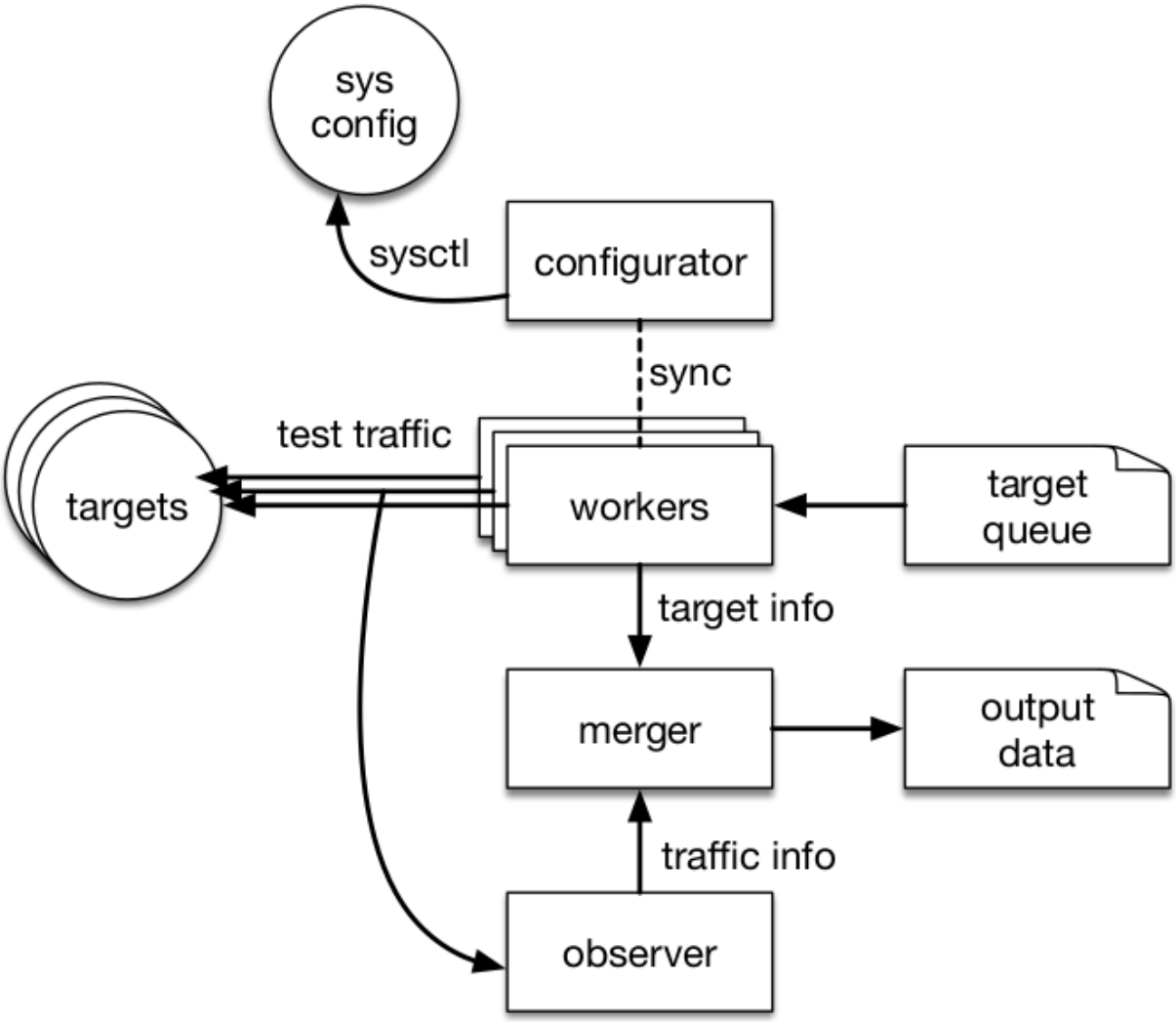

Fig. 3.1: An overview of the PATHspider architecture

CHAPTER 4

Writing a plugin

PATHspider is written to be extensible and the plugins that included in the PATHspider distribution are only examples of the measurements that PATHspider can perform.

The exact specification of plugins is defined in *pathspider.base.ISpider*, though much of the functionality required is implemented by the abstract *pathspider.base.Spider* class which plugins should inherit.

4.1 Required Functions

In order to write a plugin you will need to produce implementations for the following: *config_zero*, *config_one*, *connect* and *merge*.

Optionally, you can provide *pre_connect* and *post_connect*.

4.1.1 Configurator

These functions perform global changes that may be required between performing the baseline (A) and the experimental (B) configurations. The changes may be a call to sysctl, changes via netfilter or a call to a robot arm to reposition the satellite array. In the event that global state changes are not required, these can be implemented as no-ops.

An example implementation of these methods can be found in *ecnspider3*:

ECNSpider.**config_zero**()
: Disables ECN negotiation via sysctl.

ECNSpider.**config_one**()
: Enables ECN negotiation via sysctl.

4.1.2 (Pre-,Post-)Connection

The pre-connection function will run only once, and the result of the pre-connection operation will be available to both runs of the connection and post-connection functions.

If you require to pass different values depending on the configuration, you can perform two operations in the pre-connect function, returning a tuple, and selecting the value to use based on the configuration in the later functions.

An example implementation of these methods can be found in *ecnspider3*:

ECNSpider.**connect**(*job*, *pcs*, *config*)
: Performs a TCP connection.

ECNSpider.**post_connect**(*job*, *conn*, *pcs*, *config*)
Close the socket gracefully.

4.1.3 Merging

The merge function will be called for every job and given the job record and the observer record. The merge function is then to return the final record to be recorded in the dataset for the measurement run.

Warning: It is possible for the Observer to return a NO_FLOW object in some circumstances, where the flow has not been observed. Any implementation must handle this gracefully.

An example implementation of this method can be found in *ecnspider3*:

ECNSpider.**merge**(*flow*, *res*)
Merge flow records.

Includes the configuration and connection success or failure of the socket connection with the flow record.

4.2 Plugin Template

A template plugin is available in the plugins that ship with the PATHspider distribution:

class templatespider.**TemplateSpider**
A template PATHspider plugin.

4.3 ISpider Interface

PATHspider will expect the following functions and attributes to be available in any plugin:

interface pathspider.base.**ISpider**
The ISpider interface defines the expected interface for PATHspider plugins.

create_observer(*self*)

terminate(*self*)

shutdown(*self*)

configurator(*self*)

worker(*self*, *worker_number*)

activate(*self*, *worker_count*, *libtrace_uri*)

pre_connect(*self*, *job*)

merge(*self*, *flow*, *res*)

config_zero(*self*)

config_one(*self*)

merger(*self*)

connect(*self*, *job*, *pcs*, *config*)

exception_wrapper(*self*, *target*, **args*, ***kwargs*)

start(*self*)

post_connect(*self*, *job*, *conn*, *pcs*, *config*)

add_job(*self*, *job*)

CHAPTER 5

Abstract Spider

The core functionality of PATHspider is implemented in *pathspider.base.Spider*. The documentation for this class is below:

class pathspider.base.**Spider**

A spider consists of a configurator (which alternates between two system configurations), a large number of workers (for performing some network action for each configuration), an Observer which derives information from passively observed traffic, and a thread that merges results from the workers with flow records from the collector.

activate(*worker_count*, *libtrace_uri*)

The activate function performs initialisation of a pathspider plugin.

Parameters

- **worker_count** (*int*) – The number of workers to use.
- **libtrace_uri** (*str*) – The URI to pass to the Observer to describe the interface on which packets should be captured.

See also *pathspider.base.ISpider.activate()*

It is expected that this function will be overloaded by plugins, though the plugin should always make a call to the activate() function of the abstract Spider class as this initialises all of the base functionality:

```
super().activate(worker_count=worker_count,
                 libtrace_uri=libtrace_uri,
                 check_interrupt=check_interrupt)
```

This can be used to initialise any variables which may be required in the object. Do not initialise any variables in the __init__ method, or perform any other operations there as all plugins must be instantiated in order to be loaded and this will cause unnecessary delays in the starting of pathspider.

add_job(*job*)

Adds a job to the job queue.

If PATHspider is currently stopping, the job will not be added to the queue.

config_one()

Changes the global state or system configuration for the experimental measurements.

config_zero()

Changes the global state or system configuration for the baseline measurements.

configurator()

Thread which synchronizes on a set of semaphores and alternates between two system states.

connect (*job*, *pcs*, *config*)
Performs the connection.

Parameters

- **job** (*dict*) – The job record.
- **pcs** (*dict*) – The result of the pre-connection operations(s).
- **config** (*int*) – The current state of the configurator (0 or 1).

Returns object – Any result of the connect operation to be passed to *pathspider.base.Spider.post_connect()*.

The connect function is used to perform the connection operation and is run for both the A and B test. This method is not implemented in the abstract *pathspider.base.Spider* class and must be implemented by any plugin.

Sockets created during this operation can be returned by the function for use in the post-connection phase, to minimise the time that the configurator is blocked from moving to the next configuration.

create_observer()
Create a flow observer.

This function is called by the base Spider logic to get an instance of *pathspider.observer.Observer* configured with the function chains that are requried by the plugin.

This method is not implemented in the abstract *pathspider.base.Spider* class and must be implemented by any plugin.

For more information on how to use the flow observer, see *Observer*.

merge (*flow*, *res*)
Merge a job record with a flow record.

Parameters

- **flow** (*dict*) – The flow record.
- **res** (*dict*) – The job record.

Returns tuple – Final record for job.

In order to create a final record for reporting on a job, the final job record must be merged with the flow record. This function should be implemented by any plugin to provide the logic for this merge as the keys used in these records cannot be known by PATHspider in advance.

This method is not implemented in the abstract *pathspider.base.Spider* class and must be implemented by any plugin.

merger()
Thread to merge results from the workers and the observer.

post_connect (*job*, *conn*, *pcs*, *config*)
Performs post-connection operations.

Parameters

- **job** (*dict*) – The job record.
- **conn** (*object*) – The result of the connection operation(s).
- **pcs** (*dict*) – The result of the pre-connection operations(s).

- **config** (*int*) – The state of the configurator during *pathspider.base.Spider.connect()*.

Returns dict – Result of the pre-connection operation(s).

The post_connect function can be used to perform any operations that must be performed after each connection. It will be run for both the A and the B configuration, and is not synchronised with the configurator.

Plugins to PATHspider can optionally implement this function. If this function is not overloaded, it will be a noop.

Any sockets or other file handles that were opened during *pathspider.base.Spider.connect()* should be closed in this function if they have not been already.

pre_connect (*job*)
Performs pre-connection operations.

Parameters **job** (*dict*) – The job record.

Returns dict – Result of the pre-connection operation(s).

The pre_connect function can be used to perform any operations that must be performed before each connection. It will be run only once per job, with the same result passed to both the A and B connect calls. This function is not synchronised with the configurator.

Plugins to PATHspider can optionally implement this function. If this function is not overloaded, it will be a noop.

shutdown ()
Shut down PathSpider in an orderly fashion, ensuring that all queued jobs complete, and all available results are merged.

start ()
This function starts a PATHspider plugin.

In order to run, the plugin must have first been activated by calling its `activate()` method. This function causes the following to happen:

•Set the running flag

•Create an *pathspider.observer.Observer* and start its process

•Start the merger thread

•Start the configurator thread

•Start the worker threads

The number of worker threads to start was given when activating the plugin.

terminate ()
Shut down PathSpider as quickly as possible, without any regard to completeness of results.

worker (*worker_number*)
This function provides the logic for the worker threads.

Parameters **worker_number** (*int*) – The unique number of the worker.

The workers operate as continuous loops:

•Fetch next job from the job queue

•Perform pre-connection operations

•Acquire a lock for “config_zero”

•Perform the “config_zero” connection

- Release “config_zero”
- Acquire a lock for “config_one”
- Perform the “config_one” connection
- Release “config_one”
- Perform post-connection operations for config_zero and pass the result to the merger
- Perform post-connection operations for config_one and pass the result to the merger
- Do it all again

If the job fetched is the SHUTDOWN_SENTINEL, then the worker will terminate as this indicates that all the jobs have now been processed.

CHAPTER 6

Observer

6.1 Observer Function Chains

PATHspider's observer will accept functions and pass python-libtrace dissected packets along with the associated flow record to them for every packet recieved.

Function Chain	Description
new_flow_chain	Functions to initialise fields in the flow record for new flows.
ip4_chain	Functions to record details from IPv4 headers.
ip6_chain	Functions to record details from IPv6 headers.
tcp_chain	Functions to record details from TCP headers.
udp_chain	Functions to record details from UDP headers.
l4_chain	Functions to record details from other layer 4 headers.

If a function returns False, the Observer will consider the flow to be finished and will pass it to be merged with the job record after a short delay.

6.2 Observer Implementation

class pathspider.observer.**Observer**(*lturi*, *new_flow_chain=[]*, *ip4_chain=[]*, *ip6_chain=[]*, *tcp_chain=[]*, *udp_chain=[]*, *l4_chain=[]*)

Wraps a packet source identified by a libtrace URI, parses packets to divide them into flows, passing these packets and flows onto a function chain to allow data to be associated with each flow.

__init__(*lturi*, *new_flow_chain=[]*, *ip4_chain=[]*, *ip6_chain=[]*, *tcp_chain=[]*, *udp_chain=[]*, *l4_chain=[]*)

Create an Observer.

Parameters

- **new_flow_chain** (*array(function)*) – Array of functions to initialise new flows.
- **ip4_chain** (*array(function)*) – Array of functions to pass IPv4 headers to.
- **ip6_chain** (*array(function)*) – Array of functions to pass IPv6 headers to.
- **tcp_chain** (*array(function)*) – Array of functions to pass TCP headers to.
- **udp_chain** (*array(function)*) – Array of functions to pass UDP headers to.
- **l4_chain** (*array(function)*) – Array of functions to pass other layer 4 headers to.

See also *Observer Documentation*

flush()

purge_idle(*timeout=30*)

run_flow_enqueuer(*flowqueue*, *irqueue=None*)

Symbols

A

C

E

F

I

M

O

P

R

S

T

W