Peafowl Documentation

Release 1.0.0

Daniele De Sensi

Jan 18, 2021

USER GUIDE

CHAPTER

ONE

INTRODUCTION

Peafowl is a flexible and extensible Deep Packet Inspection (DPI) framework which can be used to identify the application protocols carried by IP (IPv4 and IPv6) packets and to extract and process data and metadata at different layers. Peafowl is implemented in **C**. However, **C++** and **Python** APIs are also provided. Since **C++** and **Python** wraps the *C* interface, they could introduce some small overhead (e.g. due to some extra data copies, etc...). As a rule of thumb, you should use the *C* interface if performance is a major concern, and **C++** or **Python** interfaces if you are more concerned about ease of use.

By using Peafowl it is possible to implement different kinds of applications like:

- URL filtering (for parental control or access control)
- User-Agent or Content-Type filtering (e.g. block traffic for mobile users, block video traffic, etc...)
- Security controls (e.g. block the traffic containing some malicious signatures or patterns)
- Data leak prevention
- Quality of Service and Traffic shaping (e.g. to give higher priority to VoIP traffic)

Peafowl is not tied to any specific technology for packet capture. Accordingly, you can capture the packets using pcap, sockets, DPDK, PF_RING or whatever technology you prefer.

To correctly identify the protocol also when its data is split among multiple IP fragments and/or TCP segments and to avoid the possibility of evasion attacks, if required, the framework can perform IP defragmentation and TCP stream reassembly.

For a detailed description of the framework, of its usage, its API and on how to extend it, please refer to the documentation.

Note: If you use Peafowl for scientific purposes, please cite our paper:

```
@inproceedings{ff:DPI:14,
    address = {Munich, Germany},
    author = {Danelutto, Marco and Deri, Luca and De Sensi, Daniele and Torquati, Massimo},
    booktitle = {Proceedings of 15th International Parallel Computing Conference ({ParCo})},
    doi = {10.3233/978-1-61499-381-0-92},
    editor = {Michael Bader and Arndt Bode and Hans-Joachim Bungartz and Michael Gerndt and Gerhard R.
    Joubert and Frans Peters},
    keywords = {fastflow, dpi, network monitoring},
    pages = {92 – 99},
    pdf = {http://pages.di.unipi.it/desensi/assets/pdf/2013_ParCo.pdf},
    publisher = {IOS Press},
```

```
    series = {Advances in Parallel Computing},
    title = {Deep Packet Inspection on Commodity Hardware using FastFlow},
    url = {http://ebooks.iospress.nl/publication/35869},
    volume = {25},
    year = {2013}
}
```

CHAPTER

TWO

CONTRIBUTIONS

Peafowl has been mainly developed by Daniele De Sensi (d.desensi.software@gmail.com).

The following people contributed to Peafowl:

- Daniele De Sensi (d.desensi.software@gmail.com): Main developer
- Michele Campus (michelecampus5@gmail.com): DNS, RTP and RTCP dissectors, L2 parsing
- Lorenzo Mangani (lorenzo.mangani@gmail.com): SIP, RTP and Skype dissectors
- max197616 (https://github.com/max197616): SSL dissector
- InSdi (https://github.com/InSdi) (indu.mss@gmail.com): Viber, Kerberos and MySQL dissectors
- QXIP B.V. (http://qxip.net/) sponsored the development of some Peafowl features (e.g. SIP, RTP, RTCP dissectors and others)
- CounterFlowAI (https://www.counterflow.ai/) sponsored the development of some Peafowl features (e.g. TCP statistics)
- David Cluytens (https://github.com/cldavid): QUIC5 dissector

I would like to thank Prof. Marco Danelutto, Dr. Luca Deri and Dr. Massimo Torquati for their essential help and valuable advices.

CHAPTER

THREE

CONTRIBUTING

If you would like to contribute to Peafowl development, for example by adding new protocols, please refer to the documentation.

CHAPTER

FOUR

DISCLAIMER

The authors of Peafowl are strongly against any form of censorship. Please make sure that you respect the privacy of users and you have proper authorization to listen, capture and inspect network traffic.

4.1 Building and Installing

First of all, download Peafowl:

```
$ git clone git://github.com/DanieleDeSensi/peafowl.git
$ cd peafowl
```

To install Peafowl:

C and C++

```
$ mkdir build
$ cd build
$ cmake ../
$ make
```

Then, you can install it:

```
$ make install
```

To install it into a non-default directory *dir*, simply specify the *-DCMAKE_INSTALL_PREFIX=dir* when calling *cmake*.

Python

```
$ pip install --user .
```

This will install a *pypeafowl* module.

If you want to build the Peafowl module without installing it:

```
$ mkdir build
$ cd build
$ cmake ../ -DENABLE_PYTHON=ON
$ make
$ cd ..
```

Then, simply copy the *./build/src/pypeafowl.so* file to your working directory.

4.2 Basic Usage

After installing Peafowl, it can be used by your application by specifying the appropriate compilation flags for C and C++ or by loading the `pypeafowl` module for Python:

C

Include the `peafowl/peafowl.h` header and add `-lpeafowl` flag to the linker options.

C++

Include the `peafowl/peafowl.hpp` header and add `-lpeafowl` flag to the linker options.

Python

```
import pypeafowl as pfwl
```

The first thing to do in your program, is creating an handle to the Peafowl library:

C

```
pfwl_state_t* handle = pfwl_init();
```

C++

```
peafowl::Peafowl* handle = new peafowl::Peafowl();
```

Python

```
handle = pfwl.Peafowl()
```

This call initializes the framework and returns an handle, which will be used for most of the framework calls. After creating the handle, you can start analyzing the network packets. As anticipated, Peafowl does not rely on any specific packet capture library, and only requires you to provide a pointer to the packet, which you can read with whatever mechanism you prefer (e.g. libpcap, etc..). To dissect the packet:

C

```
pfwl_flow_info_t* info;
pfwl_status_t status = pfwl_dissect_from_L3(handle, pkt, length, ts, info);
```

The parameters are:

- The handle to the framework
- The packet, as a pointer to the beginning of Layer 3 (IP) header
- The packet length
- A timestamp. By default it must specified with seconds resolution. However, this may be changed with appropriate calls (see **`API Reference`_** for details).
- The last parameter will be filled by Peafowl with the information about protocols detected at different layers and about the data and metadata carried by the different layers.

The call returns a status which provides additional information on the processing (or an error).

For example, to print the application protocol:

```
if(status >= PFWL_STATUS_OK){
  printf("%s\n", pfwl_get_L7_protocol_name(info.l7.protocol));
}
```

C++

```
peafowl::DissectionInfo info = handle->dissectFromL3(pkt, ts);
```

The parameters are:

- The packet, as a pointer to the beginning of Layer 3 (IP) header
- A timestamp. By default it must specified with seconds resolution. However, this may be changed with appropriate calls (see **`API Reference`_** for details).

This call returns a struct containing the status of the processing and information about protocols detected at different layers and about the data and metadata carried by the different layers.

For example, to print the application protocol:

```
if(!info.getStatus().isError()){
  std::cout << info.getL7().getProtocol().getName() << std::endl;
}
```

Python

```
info = handle.dissectFromL3(pkt, ts)
```

The parameters are:

- The packet, as a pointer to the beginning of Layer 3 (IP) header
- A timestamp. By default it must specified with seconds resolution. However, this may be changed with appropriate calls (see **`API Reference`_** for details).

This call returns a struct containing the status of the processing and information about protocols detected at different layers and about the data and metadata carried by the different layers.

For example, to print the application protocol:

```
if not info.getStatus().isError():
  print(info.getL7().getProtocol().getName())
```

Similar calls are available for analyzing the packet starting from the beginning of Layer 2 or Layer 4 header. For more information please refer to the **`API Reference`_**.

Eventually, when Peafowl is no more needed, you should deallocate the resources used by Peafowl:

C

```
pfwl_terminate(handle);
```

C++

```
delete handle;
```

Python

```
del handle
```

For a more detailed description of the aforementioned calls and for other API calls, please refer to the **`API Reference`_** documentation.

Some full working examples can be found in the demo folder:

- C API

- Python API

4.3 Flows

To identify the application protocol, packets are classified in bidirectional sets of packets called *flows*. All the packets in a *flow* share the same:

- Source IP and Destination IP addressess
- Source and Destination Ports
- Layer 4 protocol (TCP, UDP, etc...)

Flows are stored by Peafowl to correctly identify the protocol by correlating information of subsequent packets. When a connection is terminated (i.e. FINs arrived for TCP flows), or when no packets are received for a given amount of time (30 seconds by default), the *flow* is considered as terminated and it is removed from the Peafowl internal storage.

There are cases where the user may be interested not only in the information about the packet, but also on information about the flow (e.g. how many packets/bytes have been sent on that *flow*). Such information can be accessed after the packet has been processed. For example, to know how many packets have been received on that flow up to that moment:

C

```
double* packetsStat = info->flow_info->statistics[PFWL_STAT_PACKETS];
long num_packets = packetsStat[PFWL_DIRECTION_OUTBOUND] +
                   packetsStat[PFWL_DIRECTION_INBOUND];
```

C++

```
long numPackets = info.getStatistic(PFWL_STAT_PACKETS, PFWL_DIRECTION_OUTBOUND) +
                  info.getStatistic(PFWL_STAT_PACKETS, PFWL_DIRECTION_INBOUND)
```

Python

```
numPackets = info.getStatistic(pfwl.Statistic.PACKETS, pfwl.Direction.OUTBOUND) +
             info.getStatistic(pfwl.Statistic.PACKETS, pfwl.Direction.INBOUND)
```

Since the *flows* are bidirectional, we consider as *outbound* the packets flowing from source to destination host (as specified in the *info* structure) and as *inbound* the packets flowing from destination to source host.

In some cases, instead of accessing such statistic packet-by-packet, it may be more helpful to access them only once, when the flow terminates and it is removed from the Peafowl storage. It is possible to do that by specifying a callback function, which will be called by Peafowl when a flow terminates. This can be done in the following way:

C

```
void cb(pfwl_flow_info_t* flow_info){
  // You can here access to the information about the
  // flow before it is removed from the storage.
}

...

int main(int argc, char** argv){
  ...
  // Create peafowl handler, etc...
  pfwl_set_flow_termination_callback(handle, &cb);
```

(continues on next page)

(continued from previous page)

```
  ...
  // Start dissecting the packets
  ...
}
```

C++

```
class FlowManager: public peafowl::FlowManager{
public:
  void onTermination(const peafowl::FlowInfo& info){
    // You can here access to the information about the
    // flow before it is removed from the storage.
  }
};

...

int main(int argc, char** argv){
  ...
  // Create peafowl handler, etc...
  FlowManager fm;
  handle->setFlowManager(&fm);
  ...
  // Start dissecting the packets
  ...
}
```

Python

```
class FlowManagerPy(pfwl.FlowManager):
    def onTermination(self, f):
      # You can here access to the information about the
      # flow before it is removed from the storage.

...

def main():
  ...
  # Create peafowl handler, etc...
  fm = FlowManagerPy()
  p.setFlowManager(fm)
  ...
  # Start dissecting the packets
  ...

if __name__ == "__main__":
    main()
```

For a more detailed description of the aforementioned calls and for other API calls, please refer to the **`API Reference`_** documentation.

Some full working examples can be found in the demo folder:

- C API
- C++ API
- Python API

4.4 Fields Extraction

Besides identifying the application protocol carried by the packets, Peafowl can also extract some data and metadata carried by the application protocol (e.g. HTTP URL, DNS Server Name, etc...). We call each of these piece of information *fields* (i.e., HTTP URL is a *field*).

To avoid performance overhead caused by extracting fields which would not be needed by the user, the user needs first to specify which fields must be extracted by Peafowl. Then, the user can check, packet-by-packet if the field was present. If the field is present, the user can then check its value.

> **Warning:** Please note that the fields will be overwritten when the next packet is read, i.e. if you need to preserve the value of a field for a longer time, you need to make a copy of the field.

For example, to extract the HTTP URL field:

C

```c
// Create peafowl handler, etc...
// Tell Peafowl to extract HTTP URL
pfwl_field_add_L7(handle, PFWL_FIELDS_L7_HTTP_URL);
...
// Start dissecting the packets
...
// Check if the field is present and print it.
pfwl_string_t field;
if(!pfwl_field_string_get(info.l7.protocol_fields, PFWL_FIELDS_L7_HTTP_URL, &field)){
   printf("HTTP URL found: %.*s\n", (int) field.length, field.value);
}
```

> **Warning:** Please note that, to avoid copying data from the packet into another buffer, string fields are not '0' terminated and you must explicitely consider their length.

C++

```cpp
// Create peafowl handler, etc...
// Tell Peafowl to extract HTTP URL
handle.fieldAddL7(PFWL_FIELDS_L7_HTTP_URL);
...
// Start dissecting the packets
...
// Check if the field is present and print it.
peafowl::Field field = info.getField(PFWL_FIELDS_L7_HTTP_URL);
if(field.isPresent()){
  std::cout << "HTTP URL found: " << field.getString() << std::endl;
}
```

Python

```python
# Create peafowl handler, etc...
# Tell Peafowl to extract HTTP URL
handle.fieldAddL7(pfwl.Field.HTTP_URL)
...
# Start dissecting the packets
```

(continues on next page)

(continued from previous page)

```
...
# Check if the field is present and print it.
field = info.getField(pfwl.Field.HTTP_URL)
if field.isPresent():
  print("HTTP URL found: " + field.getString())
```

For a more detailed description of the aforementioned calls and for other API calls, please refer to the **`API Reference`_** documentation.

Some full working examples can be found in the demo folder:

- C API:
  - DNS Extraction : Extracts name server, authority server and IP address of name server from DNS packets
  - HTTP JPEG Dump : Dumps on the disk all the jpeg images carried by HTTP packets captured from a .pcap file or from the network.

4.5 Supported Protocols

Peafowl can identify some of the most common protocols. To add more protocols refer to the corresponding section in this document. The supported protocols are:

Protocol	Quality	Protocol	Quality	Protocol	Quality	Protocol	Quality
HTTP	5/5	SSL	5/5	POP	3/5	IMAP	5/5
SMTP	3/5	BGP	5/5	DHCP	5/5	DHCPv6	5/5
DNS	5/5	MDNS	5/5	NTP	5/5	SIP	5/5
RTP	4/5	RTCP	4/5	Skype	3/5	Hangout	3/5
WhatsApp	4/5	Telegram	?/5	Dropbox	3/5	Spotify	5/5
SSH	5/5	Bitcoin	4/5	Ethereum	4/5	Zcash	4/5
Monero	4/5	Stratum	5/5	JSON-RPC	5/5	SSDP	5/5
STUN	5/5	QUIC	5/5	MQTT	5/5	Viber	3/5
Kerberos	3/5	MySQL	5/5	Git	5/5		

At the moment, data and metadata extraction is supported for the following protocols (for a full list of fields please refer to the [Peafowl](include/peafowl/peafowl.h) header:

Protocol	Extracted data
HTTP	Any kind of HTTP header, HTTP body, HTTP version, etc...
SSL	Version, Cipher Suite, Server Name, Extensions, Elliptic Curves, JA3, JA3S
SIP	Request URI, Contact URI, Call ID, Method, etc...
DNS	Server name, authority name
QUIC	Version, SNI

4.6 C API

4.6.1 Peafowl C API

Class Hierarchy

File Hierarchy

Full API

Classes and Structs

Struct pfwl_array_t

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_array_t
A peafowl array.

Struct pfwl_dissection_info

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_dissection_info
The result of the identification process.

Struct pfwl_dissection_info_l2

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_dissection_info_l2
The result of the L2 identification process.

Struct pfwl_dissection_info_l3

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_dissection_info_l3
The result of the L3 identification process.

Struct pfwl_dissection_info_l4

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_dissection_info_l4
The result of the L4 identification process.

Struct pfwl_dissection_info_l7

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_dissection_info_l7
The result of the L7 identification process.

Struct pfwl_field

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_field
A generic field extracted by peafowl.

Struct pfwl_flow_info

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_flow_info
Public information about the flow.

Struct pfwl_pair_t

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_pair_t
A peafowl pair.

Struct pfwl_string_t

- Defined in file_include_peafowl_peafowl.h

Struct Documentation

struct pfwl_string_t
A string as represented by peafowl.

Enums

Enum pfwl_datalink_type

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_datalink_type
L2 datalink protocols supported by peafowl. When adding a new protocol, please update the pfwl_l2_protocols_names array in parsing_l2.c

Values:

enumerator PFWL_PROTO_L2_EN10MB
IEEE 802.3 Ethernet (10Mb, 100Mb, 1000Mb, and up)

enumerator PFWL_PROTO_L2_LINUX_SLL
Linux "cooked" capture encapsulation.

enumerator PFWL_PROTO_L2_IEEE802_11_RADIO
Radiotap link-layer information followed by an 802.11 header

enumerator PFWL_PROTO_L2_IEEE802_11
IEEE 802.11.

enumerator PFWL_PROTO_L2_IEEE802
IEEE 802.5 Token Ring.

enumerator PFWL_PROTO_L2_SLIP
SLIP, encapsulated with a LINKTYPE_SLIP header.

enumerator PFWL_PROTO_L2_PPP
PPP, as per RFC 1661 and RFC 1662.

enumerator PFWL_PROTO_L2_FDDI
FDDI, as specified by ANSI INCITS 239-1994.

enumerator PFWL_PROTO_L2_RAW
Raw IP.

enumerator PFWL_PROTO_L2_LOOP
OpenBSD loopback encapsulation.

enumerator PFWL_PROTO_L2_NULL
BSD loopback encapsulation.

enumerator PFWL_PROTO_L2_NUM
Special value to indicate an unsupported datalink type. This must be the last value

Enum pfwl_direction_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_direction_t
Possible packet directions.

Values:

enumerator PFWL_DIRECTION_OUTBOUND
From source to destination.

enumerator PFWL_DIRECTION_INBOUND
From destination to source.

Enum pfwl_dissector_accuracy_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_dissector_accuracy_t
Some dissector can run at a different accuracy level. This represent the level of accuracy that may be required to a dissector.

Values:

enumerator PFWL_DISSECTOR_ACCURACY_LOW
Low accuracy.

enumerator PFWL_DISSECTOR_ACCURACY_MEDIUM
Medium accuracy.

enumerator PFWL_DISSECTOR_ACCURACY_HIGH
High accuracy.

Enum pfwl_field_id_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_field_id_t
Protocol fields which can be extracted by peafowl.

Values:

enumerator PFWL_FIELDS_L7_SIP_REQUEST_URI
[STRING]

enumerator PFWL_FIELDS_L7_SIP_METHOD
[STRING]

enumerator PFWL_FIELDS_L7_SIP_CALLID
[STRING]

enumerator PFWL_FIELDS_L7_SIP_REASON
[STRING]

enumerator PFWL_FIELDS_L7_SIP_RTCPXR_CALLID
[STRING]

enumerator PFWL_FIELDS_L7_SIP_CSEQ
[STRING]

enumerator PFWL_FIELDS_L7_SIP_CSEQ_METHOD_STRING
[STRING]

enumerator PFWL_FIELDS_L7_SIP_VIA
[STRING]

enumerator PFWL_FIELDS_L7_SIP_CONTACT_URI
[STRING]

enumerator PFWL_FIELDS_L7_SIP_RURI_USER
[STRING]

enumerator PFWL_FIELDS_L7_SIP_RURI_DOMAIN
[STRING]

enumerator PFWL_FIELDS_L7_SIP_FROM_USER
[STRING]

enumerator PFWL_FIELDS_L7_SIP_FROM_DOMAIN
[STRING]

enumerator PFWL_FIELDS_L7_SIP_TO_USER
[STRING]

enumerator PFWL_FIELDS_L7_SIP_TO_DOMAIN
[STRING]

enumerator PFWL_FIELDS_L7_SIP_PAI_USER
[STRING]

enumerator PFWL_FIELDS_L7_SIP_PAI_DOMAIN
[STRING]

enumerator PFWL_FIELDS_L7_SIP_PID_URI
[STRING]

enumerator PFWL_FIELDS_L7_SIP_FROM_URI
[STRING]

enumerator PFWL_FIELDS_L7_SIP_TO_URI
[STRING]

enumerator PFWL_FIELDS_L7_SIP_RURI_URI
[STRING]

enumerator PFWL_FIELDS_L7_SIP_TO_TAG
[STRING]

enumerator PFWL_FIELDS_L7_SIP_FROM_TAG
[STRING]

enumerator PFWL_FIELDS_L7_DNS_NAME_SRV
[STRING] Server name

enumerator PFWL_FIELDS_L7_DNS_NS_IP_1
[STRING] Server name IP address

enumerator PFWL_FIELDS_L7_DNS_NS_IP_2
[STRING] Server name IP address

enumerator PFWL_FIELDS_L7_DNS_AUTH_SRV
[STRING] Authority name

enumerator PFWL_FIELDS_L7_SSL_VERSION
[NUMBER] SSL Version

enumerator PFWL_FIELDS_L7_SSL_VERSION_HANDSHAKE
[NUMBER] SSL Handshake Version (for client and server hellos)

enumerator PFWL_FIELDS_L7_SSL_HANDSHAKE_TYPE
[NUMBER] SSL Handshake type

enumerator PFWL_FIELDS_L7_SSL_CIPHER_SUITES
[STRING] Cypher Suites

enumerator PFWL_FIELDS_L7_SSL_EXTENSIONS
[STRING] Extensions

enumerator PFWL_FIELDS_L7_SSL_ELLIPTIC_CURVES
[STRING] Supported elliptic curves

enumerator PFWL_FIELDS_L7_SSL_ELLIPTIC_CURVES_POINT_FMTS
[STRING] Supported elliptic curves point formats

enumerator PFWL_FIELDS_L7_SSL_SNI
[STRING] Server name extension found in client certificate

enumerator PFWL_FIELDS_L7_SSL_CERTIFICATE
[STRING] Server name found in server certificate

enumerator PFWL_FIELDS_L7_SSL_JA3
[STRING] SSL JA3 Fingerprint (https://github.com/salesforce/ja3). If HANDSHAKE_TYPE == 0x01

enumerator PFWL_FIELDS_L7_HTTP_VERSION_MAJOR
[NUMBER] HTTP Version - Major

enumerator PFWL_FIELDS_L7_HTTP_VERSION_MINOR
[NUMBER] HTTP Version - Minor

enumerator PFWL_FIELDS_L7_HTTP_METHOD
[NUMBER] HTTP Method. For the possible values

enumerator PFWL_FIELDS_L7_HTTP_STATUS_CODE
[NUMBER] HTTP Status code

enumerator PFWL_FIELDS_L7_HTTP_MSG_TYPE
[NUMBER] HTTP request or response. For the possible values

enumerator PFWL_FIELDS_L7_HTTP_BODY
[STRING] HTTP Body

enumerator PFWL_FIELDS_L7_HTTP_URL
[STRING] HTTP URL

enumerator PFWL_FIELDS_L7_HTTP_HEADERS
[MMAP] HTTP headers

enumerator PFWL_FIELDS_L7_RTP_PTYPE
[NUMBER] RTP Payload Type

enumerator PFWL_FIELDS_L7_RTP_SEQNUM
[NUMBER] RTP Sequence Number

enumerator PFWL_FIELDS_L7_RTP_TIMESTP
[NUMBER] RTP Timestamp

enumerator PFWL_FIELDS_L7_RTP_SSRC
[NUMBER] RTP Syncronization Source Identifier (Host byte order)

enumerator PFWL_FIELDS_L7_RTCP_SENDER_ALL
[NUMBER] To extract all the Sender fields

enumerator PFWL_FIELDS_L7_RTCP_SENDER_SSRC
[NUMBER] RTCP Sender SSRC

enumerator PFWL_FIELDS_L7_RTCP_SENDER_TIME_MSW
[NUMBER] RTCP Sender timestamp MSW

enumerator PFWL_FIELDS_L7_RTCP_SENDER_TIME_LSW
[NUMBER] RTCP Sender timestamp LSW

enumerator PFWL_FIELDS_L7_RTCP_SENDER_TIME_RTP
[NUMBER] RTCP Sender timestamp RTP

enumerator PFWL_FIELDS_L7_RTCP_SENDER_PKT_COUNT
[NUMBER] RTCP Sender packet count

enumerator PFWL_FIELDS_L7_RTCP_SENDER_OCT_COUNT
[NUMBER] RTCP Sender octet count

enumerator PFWL_FIELDS_L7_RTCP_SENDER_ID
[NUMBER] RTCP Sender Identifier

enumerator PFWL_FIELDS_L7_RTCP_SENDER_FLCNPL
[NUMBER] RTCP Sender Fraction lost + Cumulative pkt lost

enumerator PFWL_FIELDS_L7_RTCP_SENDER_EXT_SEQN_RCV
[NUMBER] RTCP Sender Extended highest sequence number received

enumerator PFWL_FIELDS_L7_RTCP_SENDER_INT_JITTER
[NUMBER] RTCP Sender Interarrival Jitter

enumerator PFWL_FIELDS_L7_RTCP_SENDER_LSR
[NUMBER] RTCP Sender Last SR timestamp

enumerator PFWL_FIELDS_L7_RTCP_SENDER_DELAY_LSR
[NUMBER] RTCP Sender Delay last SR timestamp

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_ALL
[NUMBER] To extract all the Receiver fields

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_SSRC
[NUMBER] RTCP Receiver SSRC

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_ID
[NUMBER] RTCP Receiver Identifier

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_FLCNPL
[NUMBER] RTCP Receiver Fraction lost + Cumulative pkt lost

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_EXT_SEQN_RCV
[NUMBER] RTCP Receiver Extended highest sequence number received

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_INT_JITTER
[NUMBER] RTCP Receiver Interarrival Jitter

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_LSR
[NUMBER] RTCP Receiver Last SR timestamp

enumerator PFWL_FIELDS_L7_RTCP_RECEIVER_DELAY_LSR
[NUMBER] RTCP Receiver Delay last SR timestamp

enumerator PFWL_FIELDS_L7_RTCP_SDES_CSRC
[NUMBER] RTCP Source description CSRC ID

enumerator PFWL_FIELDS_L7_RTCP_SDES_TEXT
[STRING] RTCP Source description Text

enumerator PFWL_FIELDS_L7_JSON_RPC_FIRST
[NUMBER] Dummy value to mark first JSON RPC field.

enumerator PFWL_FIELDS_L7_JSON_RPC_VERSION
[NUMBER] JSON-RPC version.

enumerator PFWL_FIELDS_L7_JSON_RPC_MSG_TYPE
[NUMBER] Msg type. 0 = Request

enumerator PFWL_FIELDS_L7_JSON_RPC_ID
[STRING] Id field.

enumerator PFWL_FIELDS_L7_JSON_RPC_METHOD
[STRING] Method field.

enumerator PFWL_FIELDS_L7_JSON_RPC_PARAMS
[STRING] Params field.

enumerator PFWL_FIELDS_L7_JSON_RPC_RESULT
[STRING] Result field.

enumerator PFWL_FIELDS_L7_JSON_RPC_ERROR
[STRING] Error field.

enumerator PFWL_FIELDS_L7_JSON_RPC_LAST
[NUMBER] Dummy value to mark last JSON RPC field.

enumerator PFWL_FIELDS_L7_QUIC_VERSION
[STRING] Version.

enumerator PFWL_FIELDS_L7_QUIC_SNI
[STRING] Server Name Indication.

enumerator PFWL_FIELDS_L7_QUIC_UAID
[STRING] User Agent Identifier.

enumerator PFWL_FIELDS_L7_QUIC_JA3
[STRING] Quic/TLS JA3 Fingerprint (https://github.com/salesforce/ja3)

enumerator PFWL_FIELDS_L7_STUN_MAPPED_ADDRESS
[STRING] Mapped address (or xor-mapped address) (format x.y.z.w for IPv4 and a:b:c:d:e:f:g:h for IPv6).

enumerator PFWL_FIELDS_L7_STUN_MAPPED_ADDRESS_PORT
[NUMBER] Mapped address port (or xor-mapped port) .

enumerator PFWL_FIELDS_L7_NUM
[STRING] Dummy value to indicate number of fields. Must be the last field specified.

Enum pfwl_field_matching_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_field_matching_t
Possible type of matchings when associating tags to packets.

Values:

enumerator PFWL_FIELD_MATCHING_PREFIX
Prefix matching.

enumerator PFWL_FIELD_MATCHING_EXACT
Exact matching.

enumerator PFWL_FIELD_MATCHING_SUFFIX
Suffix matching.

enumerator PFWL_FIELD_MATCHING_ERROR
Invalid tag matching.

Enum pfwl_field_type_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_field_type_t
Possible types for peafowl fields.

Values:

enumerator PFWL_FIELD_TYPE_STRING

enumerator PFWL_FIELD_TYPE_NUMBER

enumerator PFWL_FIELD_TYPE_ARRAY

enumerator PFWL_FIELD_TYPE_PAIR

enumerator PFWL_FIELD_TYPE_MMAP

Enum pfwl_flows_strategy_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_flows_strategy_t
Possible strategies to adopt when there are too many flows in the flows table.

Values:

enumerator PFWL_FLOWS_STRATEGY_NONE
Flows are always added to the table.

enumerator PFWL_FLOWS_STRATEGY_SKIP
If the maximum capacity of the table is reached, new flows are not stored.

enumerator PFWL_FLOWS_STRATEGY_EVICT
If the maximum capacity of the table is reached, when a new flow is added the oldest flow is evicted.

Enum pfwl_protocol_l3_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_protocol_l3_t
L3 (IP) protocol.

Values:

enumerator PFWL_PROTO_L3_IPV4
IPv4.

enumerator PFWL_PROTO_L3_IPV6
IPv6.

enumerator PFWL_PROTO_L3_NUM
Special value. This must be the last value.

Enum pfwl_protocol_l7_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_protocol_l7_t
L7 (application level) protocol.

Values:

enumerator PFWL_PROTO_L7_DNS
DNS.

enumerator PFWL_PROTO_L7_MDNS
MDNS.

enumerator PFWL_PROTO_L7_DHCP
DHCP.

enumerator PFWL_PROTO_L7_DHCPv6
DHCPv6.

enumerator PFWL_PROTO_L7_NTP
NTP.

enumerator PFWL_PROTO_L7_SIP
SIP.

enumerator PFWL_PROTO_L7_RTP
RTP.

enumerator PFWL_PROTO_L7_RTCP
RTCP.

enumerator PFWL_PROTO_L7_SSH
SSH.

enumerator PFWL_PROTO_L7_SKYPE
Skype.

enumerator PFWL_PROTO_L7_HTTP
HTTP.

enumerator PFWL_PROTO_L7_BGP
BGP.

enumerator PFWL_PROTO_L7_SMTP
SMTP.

enumerator PFWL_PROTO_L7_POP3
POP3.

enumerator PFWL_PROTO_L7_IMAP
IMAP.

enumerator PFWL_PROTO_L7_SSL
SSL.

enumerator PFWL_PROTO_L7_HANGOUT
Hangout.

enumerator PFWL_PROTO_L7_WHATSAPP
WhatsApp.

enumerator PFWL_PROTO_L7_TELEGRAM
Telegram.

enumerator PFWL_PROTO_L7_DROPBOX
Dropbox.

enumerator PFWL_PROTO_L7_SPOTIFY
Spotify.

enumerator PFWL_PROTO_L7_BITCOIN
Bitcoin.

enumerator PFWL_PROTO_L7_ETHEREUM
Ethereum.

enumerator PFWL_PROTO_L7_ZCASH
Zcash.

enumerator PFWL_PROTO_L7_MONERO
Monero.

enumerator PFWL_PROTO_L7_STRATUM
Stratum mining protocol (can be used by Bitcoin, Zcash and others)

enumerator PFWL_PROTO_L7_JSON_RPC
Json-RPC.

enumerator PFWL_PROTO_L7_SSDP
SSDP.

enumerator PFWL_PROTO_L7_STUN
STUN.

enumerator PFWL_PROTO_L7_QUIC
QUIC.

enumerator PFWL_PROTO_L7_QUIC5
QUIC5.

enumerator PFWL_PROTO_L7_MQTT
MQTT.

enumerator PFWL_PROTO_L7_MYSQL
MySQL.

enumerator PFWL_PROTO_L7_VIBER
Viber.

enumerator PFWL_PROTO_L7_KERBEROS
Kerberos.

enumerator PFWL_PROTO_L7_TOR
Tor.

enumerator PFWL_PROTO_L7_GIT
Git.

enumerator PFWL_PROTO_L7_NUM
Dummy value to indicate the number of protocols.

enumerator PFWL_PROTO_L7_NOT_DETERMINED
Dummy value to indicate that the protocol has not been identified yet

enumerator PFWL_PROTO_L7_UNKNOWN
been identified

Dummy value to indicate that the protocol has not

Enum pfwl_statistic_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_statistic_t
A generic statistic for the flow. While a field is something related to the packet, a statistic is something related to the flow (e.g. packets per second, etc...).

Values:

enumerator PFWL_STAT_PACKETS
Number of packets, one value for each direction. Multiple IP fragments count like a single packet.

enumerator PFWL_STAT_BYTES
Number of bytes (from L3 start to end of packet).

enumerator PFWL_STAT_TIMESTAMP_FIRST
Timestamp of the first packet received for this flow. Resolution depends on the values provided through the pfwl_dissect_from_L* calls.

enumerator PFWL_STAT_TIMESTAMP_LAST
Timestamp of the last packet received for this flow. Resolution depends on the values provided through the pfwl_dissect_from_L* calls.

enumerator PFWL_STAT_L4_TCP_RTT_SYN_ACK
Round-Trip-Time (RTT), measuring delay from the first SYN received to the corresponding ACK. Resolution depends on the values provided through the pfwl_dissect_from_L* calls.

enumerator PFWL_STAT_L4_TCP_COUNT_SYN
Number of segments with SYN bit set.

enumerator PFWL_STAT_L4_TCP_COUNT_FIN
Number of segments with FIN bit set.

enumerator PFWL_STAT_L4_TCP_COUNT_RST
Number of segments with RST bit set.

enumerator PFWL_STAT_L4_TCP_COUNT_RETRANSMISSIONS
Number of retransmitted packets.

enumerator PFWL_STAT_L4_TCP_COUNT_ZERO_WINDOW
Number of zero window segments.

enumerator PFWL_STAT_L4_TCP_WINDOW_SCALING
Window scaling value (shift count) or -1 if the TCP option was not present.

enumerator PFWL_STAT_L7_PACKETS
Number of packets with a non-zero L7 payload.

enumerator PFWL_STAT_L7_BYTES
Number of L7 bytes. One value for each direction.

enumerator PFWL_STAT_NUM
Dummy value to indicate number of statistics. Must be the last stat specified.

Enum pfwl_status

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_status
Status of the identification process

Values:

enumerator PFWL_ERROR_L2_PARSING
L2 data unsupported, truncated or corrupted.

Errors

enumerator PFWL_ERROR_L3_PARSING
L3 data unsupported, truncated or corrupted.

enumerator PFWL_ERROR_L4_PARSING
L4 data unsupported, truncated or corrupted.

enumerator PFWL_ERROR_MAX_FLOWS
Maximum number of flows reached.

enumerator PFWL_ERROR_IPV6_HDR_PARSING
Error while parsing IPv6 headers.

enumerator PFWL_ERROR_IPSEC_NOTSUPPORTED
IPsec packet, not supported currently.

enumerator PFWL_ERROR_WRONG_IPVERSION
L3 protocol was neither IPv4 nor IPv6.

enumerator PFWL_STATUS_OK
Normal processing scenario.

enumerator PFWL_STATUS_IP_FRAGMENT
Received a fragment of an IP packet. If IP reassambly is enabled, the fragment has been stored and the data will be recompacted and analyzed when all the fragments will be received.

enumerator PFWL_STATUS_IP_DATA_REBUILT
The received datagram allowed the library to reconstruct a fragmented datagram. This status may only be returned if pfwl_parse_L3 is explicitely called. In this case, l3.pkt_refragmented will contain a pointer to the recomposed datagram. This pointer will be different from the packet provided by the user. The user should free() this pointer when it is no more needed.

enumerator PFWL_STATUS_TCP_OUT_OF_ORDER
Received an out of order TCP segment. If TCP defragmentation is enabled, the segment has been stored, and will be recomposed and analyzed when the other segments will be received.

enumerator PFWL_STATUS_TCP_CONNECTION_TERMINATED
FINs has been sent by both peers of the connection. This status is not set for connections closed by RST.

Enum pfwl_timestamp_unit_t

- Defined in file_include_peafowl_peafowl.h

Enum Documentation

enum pfwl_timestamp_unit_t
Units of timestamps used by Peafowl.

Values:

enumerator PFWL_TIMESTAMP_UNIT_MICROSECONDS
Microseconds.

enumerator PFWL_TIMESTAMP_UNIT_MILLISECONDS
Milliseconds.

enumerator PFWL_TIMESTAMP_UNIT_SECONDS
Seconds.

Unions

Union pfwl_basic_type_t

- Defined in file_include_peafowl_peafowl.h

Union Documentation

union pfwl_basic_type_t
#include <peafowl.h> A peafowl basic type.

Public Members

pfwl_string_t **string**
A string.

int64_t **number**
A number, in host byte order.

Union pfwl_ip_addr

- Defined in file_include_peafowl_peafowl.h

Union Documentation

union pfwl_ip_addr
#include <peafowl.h> An IP address.

Public Members

uint32_t **ipv4**
The IPv4 address.

struct in6_addr **ipv6**
The IPv6 address.

Functions

Function pfwl_convert_pcap_dlt

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l2_t **pfwl_convert_pcap_dlt** (int *dlt*)
pfwl_convert_pcap_dlt Converts a pcap datalink type (which can be obtained with the pcap_datalink(...) call), to a pfwl_datalink_type_t.

Return The peafowl datalink type. PFWL_DLT_NOT_SUPPORTED is returned if the specified datalink type is not supported by peafowl.

Parameters

- `dlt`: The pcap datalink type.

Function pfwl_create_flow_info_private

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_flow_info_private_t *__pfwl_create_flow_info_private__(pfwl_state_t *state, const pfwl_dissection_info_t *dissection_info)

Creates a new Peafowl flow info (to be called only if pfwl_dissect_L7 is called directly by the user).

Return The new Peafowl flow, needs to be deleted with pfwl_destroy_flow.

Parameters

- `state`: A pointer to the state of the library.
- `dissection_info`: Info about the flow (user needs to fill info up to L4 included).

Function pfwl_defragmentation_disable_ipv4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t __pfwl_defragmentation_disable_ipv4__(pfwl_state_t *state)

Disables IPv4 defragmentation.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

Function pfwl_defragmentation_disable_ipv6

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t __pfwl_defragmentation_disable_ipv6__(pfwl_state_t *state)

Disables IPv6 defragmentation.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

Function pfwl_defragmentation_enable_ipv4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_enable_ipv4** (pfwl_state_t **state*, uint16_t *table_size*)
Enables IPv4 defragmentation. It is enabled by default.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the library state.
- `table_size`: The size of the table to be used to store IPv4 fragments informations.

Function pfwl_defragmentation_enable_ipv6

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_enable_ipv6** (pfwl_state_t **state*, uint16_t *table_size*)
Enables IPv6 defragmentation. It is enabled by default.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the library state.
- `table_size`: The size of the table to be used to store IPv6 fragments informations.

Function pfwl_defragmentation_set_per_host_memory_limit_ipv4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_set_per_host_memory_limit_ipv4** (pfwl_state_t **state*, uint32_t *per_host_memory_limit*)
Sets the amount of memory (in bytes) that a single host can use for IPv4 defragmentation.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the library state.
- `per_host_memory_limit`: The maximum amount of memory that any IPv4 host can use.

Function pfwl_defragmentation_set_per_host_memory_limit_ipv6

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_set_per_host_memory_limit_ipv6**(pfwl_state_t **state*, uint32_t *per_host_memory_limit*)

Sets the amount of memory (in bytes) that a single host can use for IPv6 defragmentation.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the library state.
- `per_host_memory_limit`: The maximum amount of memory that any IPv6 host can use.

Function pfwl_defragmentation_set_reassembly_timeout_ipv4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_set_reassembly_timeout_ipv4**(pfwl_state_t **state*, uint8_t *timeout_seconds*)

Sets the maximum time (in seconds) that can be spent to reassembly an IPv4 fragmented datagram. Is the maximum time gap between the first and last fragments of the datagram.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `timeout_seconds`: The reassembly timeout.

Function pfwl_defragmentation_set_reassembly_timeout_ipv6

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_set_reassembly_timeout_ipv6**(pfwl_state_t **state*, uint8_t *timeout_seconds*)

Sets the maximum time (in seconds) that can be spent to reassembly an IPv6 fragmented datagram. Is the maximum time gap between the first and last fragments of the datagram.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

- `timeout_seconds`: The reassembly timeout.

Function pfwl_defragmentation_set_total_memory_limit_ipv4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_set_total_memory_limit_ipv4**(pfwl_state_t **state*, uint32_t *total_memory_limit*)

Sets the total amount of memory (in bytes) that can be used for IPv4 defragmentation. If defragmentation is disabled and then enabled again, this function must be called again.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library
- `total_memory_limit`: The maximum amount of memory that can be used for IPv4 defragmentation.

Function pfwl_defragmentation_set_total_memory_limit_ipv6

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_defragmentation_set_total_memory_limit_ipv6**(pfwl_state_t **state*, uint32_t *total_memory_limit*)

Sets the total amount of memory (in bytes) that can be used for IPv6 defragmentation. If defragmentation is disabled and then enabled again, this function must be called again.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library
- `total_memory_limit`: The maximum amount of memory that can be used for IPv6 defragmentation.

Function pfwl_destroy_flow_info_private

- Defined in file_include_peafowl_peafowl.h

Function Documentation

void **pfwl_destroy_flow_info_private**(pfwl_flow_info_private_t *info)

Destroys a flow info created with pfwl_create_flow.

Parameters

- `info`: The flow to be destroyed.

Function pfwl_dissect_from_L2

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_from_L2**(pfwl_state_t *state, **const** unsigned char *pkt, size_t *length*, double *timestamp*, *pfwl_protocol_l2_t* *datalink_type*, *pfwl_dissection_info_t* *dissection_info*)

Dissects the packet starting from the beginning of the L2 (datalink) header.

Return The status of the identification process.

Parameters

- `state`: The state of the library.
- `pkt`: The pointer to the beginning of datalink header.
- `length`: Length of the packet.
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the pfwl_set_timestamp_unit call. By default it is assumed that the timestamps unit is 'seconds'.
- `datalink_type`: The datalink type. They match 1:1 the pcap datalink types. You can convert a PCAP datalink type to a Peafowl datalink type by calling the function 'pfwl_convert_pcap_dlt'.
- `dissection_info`: The result of the dissection. All its bytes must be set to 0 before calling this call. Dissection information from L2 to L7 will be filled in by this call.

Function pfwl_dissect_from_L3

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_from_L3**(pfwl_state_t *state, **const** unsigned char *pkt, size_t *length*, double *timestamp*, *pfwl_dissection_info_t* *dissection_info*)

Dissects the packet starting from the beginning of the L3 (IP) header.

Return The status of the identification process.

Parameters

- `state`: The state of the library.
- `pkt`: The pointer to the beginning of IP header.

- `length`: Length of the packet (from the beginning of the IP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the pfwl_set_timestamp_unit call. By default it is assumed that the timestamps unit is 'seconds'.
- `dissection_info`: The result of the dissection. Bytes of dissection_info.l3, dissection_info.l4, dissection_info.l7 must be set to 0 before calling this call. Dissection information from L3 to L7 will be filled in by this call.

Function pfwl_dissect_from_L4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_from_L4**(pfwl_state_t **state*, **const** unsigned char **pkt*, size_t *length*, double *timestamp*, *pfwl_dissection_info_t* **dissection_info*)

Dissects the packet starting from the beginning of the L4 (UDP or TCP) header.

Return The status of the identification process.

Parameters

- `state`: The state of the library.
- `pkt`: The pointer to the beginning of UDP or TCP header.
- `length`: Length of the packet (from the beginning of the UDP or TCP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the pfwl_set_timestamp_unit call. By default it is assumed that the timestamps unit is 'seconds'.
- `dissection_info`: The result of the dissection. Bytes of dissection_info.l4, dissection_info.l7 must be set to 0 before calling this call. Dissection information about L3 header must be filled in by the caller. Dissection information from L4 to L7 will be filled in by this call.

Function pfwl_dissect_L2

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_L2**(**const** unsigned char **packet*, *pfwl_protocol_l2_t* *datalink_type*, *pfwl_dissection_info_t* **dissection_info*)

Extracts from the packet the L2 information.

Return The status of the identification process.

Parameters

- `packet`: A pointer to the packet.
- `datalink_type`: The datalink type. You can convert a PCAP datalink type to a Peafowl datalink type by calling the function 'pfwl_convert_pcap_dlt'.

- `dissection_info`: The result of the dissection. Dissection information about L2 headers will be filled in by this call.

Function pfwl_dissect_L3

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_L3**(pfwl_state_t **state*, **const** unsigned char **pkt*, size_t *length*, double *timestamp*, *pfwl_dissection_info_t* **dissection_info*)

Extracts from the packet the L3 information.

Return The status of the identification process.

Parameters

- `state`: The state of the library.
- `pkt`: The pointer to the beginning of IP header.
- `length`: Length of the packet (from the beginning of the IP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the pfwl_set_timestamp_unit call. By default it is assumed that the timestamps unit is 'seconds'.
- `dissection_info`: The result of the dissection. Bytes of dissection_info.l3, dissection_info.l4, dissection_info.l7 must be set to 0 before calling this call. Dissection information about L3 headers will be filled in by this call.

Function pfwl_dissect_L4

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_L4**(pfwl_state_t **state*, **const** unsigned char **pkt*, size_t *length*, double *timestamp*, *pfwl_dissection_info_t* **dissection_info*, pfwl_flow_info_private_t ***flow_info_private*)

Extracts from the packet the L4 information.

Return The status of the identification process.

Parameters

- `state`: The state of the library.
- `pkt`: The pointer to the beginning of UDP or TCP header.
- `length`: Length of the packet (from the beginning of the UDP or TCP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the pfwl_set_timestamp_unit call. By default it is assumed that the timestamps unit is 'seconds'.

- `dissection_info`: The result of the dissection. Bytes of dissection_info.l4, dissection_info.l7 must be set to 0 before calling this call. Dissection information about L3 headers must be filled in by the caller. l4.protocol must be filled in by the caller as well. Dissection information about L4 headers will be filled in by this call.
- `flow_info_private`: Will be filled by this library. *flow_info_private will point to the private information about the flow.

Function pfwl_dissect_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_status_t **pfwl_dissect_L7**(pfwl_state_t **state*, **const** unsigned char **pkt*, size_t *length*, *pfwl_dissection_info_t* **dissection_info*, pfwl_flow_info_private_t **flow_info_private*)

Extracts from the packet the L7 information. Before calling it, a check on L4 protocol should be done and the function should be called only if the packet is TCP or UDP. It should be used if the application already called pfwl_dissect_L4 or if the application already has the concept of 'flow'. In this case the first time that the flow is passed to the call, flow_info_private must be initialized with pfwl_init_flow_info(...) and stored with the flow already present in the application. With this call, information in dissection_info->flow are only set for L7 packets and bytes.

Return The status of the identification process.

Parameters

- `state`: The pointer to the library state.
- `pkt`: The pointer to the beginning of application data.
- `length`: Length of the packet (from the beginning of the L7 header).
- `dissection_info`: The result of the dissection. Bytes of dissection_info.l7 must be set to 0 before calling this call. Dissection information about L3 and L4 headers must be filled in by the caller. Dissection information about L7 packet will be filled in by this call.
- `flow_info_private`: The private information about the flow. It must be stored by the user and itialized with the pfwl_init_flow_info(...) call.

Function pfwl_field_add_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_field_add_L7** (pfwl_state_t *state, *pfwl_field_id_t field*)

Enables the extraction of a specific L7 field for a given protocol. When a protocol is identified, the default behavior is to not inspect the packets belonging to that flow anymore and keep simply returning the same protocol identifier.

If at least one field extraction is enabled for a certain protocol, then we keep inspecting all the new packets of that flow to extract such field. Moreover, if the application protocol uses TCP, then we have the additional cost of TCP reordering for all the segments. Is highly recommended to enable TCP reordering if it is not already enabled (remember that is enabled by default). Otherwise the informations extracted could be erroneous/incomplete.

Please note that this is only a suggestion given by the user to peafowl, and that in some cases the dissector could still extract the field, even if this has not been requested by the user. Indeed, in some cases the extraction of some fields may be needed for the correct identification of the protocol.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `field`: The field to extract.

Function pfwl_field_array_get_pair

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_field_array_get_pair** (*pfwl_field_t *fields*, *pfwl_field_id_t id*, size_t *position*, *pfwl_pair_t *pair*)

pfwl_field_array_get_pair Extracts a pair in a specific position, from a specific array field.

Return 0 if the field was present, 1 otherwise. If 1 is returned, 'pair' is not set.

Parameters

- `fields`: The list of fields.
- `id`: The field identifier.
- `position`: The position in the array.
- `pair`: The returned pair.

Function pfwl_field_array_length

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_field_array_length**(*pfwl_field_t* *fields, *pfwl_field_id_t* id, size_t *length)
pfwl_field_array_length Returns the size of a field representing an array of strings.

Return 0 if the field was present, 1 otherwise. If 1 is returned, 'size' is not set.

Parameters

- `fields`: The list of fields.
- `id`: The field identifier.
- `length`: The returned length.

Function pfwl_field_mmap_tags_add_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

void **pfwl_field_mmap_tags_add_L7**(pfwl_state_t *state, *pfwl_field_id_t* field, **const** char *key, **const** char *value, *pfwl_field_matching_t* matchingType, **const** char *tag)
pfwl_field_map_tags_add Adds a tag matching rule for a specific field.

Adds a tag matching rule for a specific multimap field.

Parameters

- `state`: A pointer to the state of the library.
- `field`: The field identifier.
- `key`: The key of the multimap value. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well.
- `value`: The value of the multimap value. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well.
- `matchingType`: Can be 'PREFIX', 'EXACT' or 'SUFFIX'.
- `tag`: The tag to assign to the packet when the field matches with 'value'.

Function pfwl_field_number_get

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_field_number_get** (*pfwl_field_t *fields*, *pfwl_field_id_t id*, int64_t **number*)
pfwl_field_number_get Extracts a specific numeric field from a list of fields.

Return 0 if the field was present, 1 otherwise. If 1 is returned, 'number' is not set.

Parameters

- `fields`: The list of fields.
- `id`: The field identifier.
- `number`: The extracted field, in host byte order.

Function pfwl_field_remove_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_field_remove_L7** (pfwl_state_t *state, *pfwl_field_id_t field*)
Disables the extraction of a specific L7 protocol field.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `field`: The field identifier.

Function pfwl_field_string_get

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_field_string_get** (*pfwl_field_t *fields*, *pfwl_field_id_t id*, *pfwl_string_t *string*)
pfwl_field_string_get Extracts a specific string field from a list of fields.

Return 0 if the field was present, 1 otherwise. If 1 is returned, 'string' is not set.

Parameters

- `fields`: The list of fields.
- `id`: The field identifier.
- `string`: The extracted field.

Function pfwl_field_string_tags_add_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

void **pfwl_field_string_tags_add_L7**(pfwl_state_t *state, *pfwl_field_id_t* field, **const** char *value, *pfwl_field_matching_t* matchingType, **const** char *tag)

pfwl_field_string_tags_add Adds a tag matching rule for a specific field.

Adds a tag matching rule for a specific string field.

Parameters

- `state`: A pointer to the state of the library.
- `field`: The field identifier.
- `value`: Is the string to be matched against the field. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well.
- `matchingType`: Can be 'PREFIX', 'EXACT' or 'SUFFIX'.
- `tag`: The tag to assign to the packet when the field matches with 'value'.

Function pfwl_field_tags_load_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

int **pfwl_field_tags_load_L7**(pfwl_state_t *state, *pfwl_field_id_t* field, **const** char *tags_file)

pfwl_field_tags_load Loads the associations between fields values and user-defined tags.

Loads the associations between fields values and user-defined tags. { "rules": [{"value": "google.com", "matchingType": "SUFFIX", "tag": "GOOGLE"}, {"value": "amazon.com", "matchingType": "SUFFIX", "tag": "AMAZON"}, ...], }

Parameters

- `state`: A pointer to the state of the library.
- `field`: The field identifier.
- `tags_file`: The name of the JSON file containing associations between fields values and tags. The structure of the JSON file depends from the type of 'field'. *If 'field' is a string:*

value: Is the string to be matched against the field. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well. matchingType: Can be 'PREFIX', 'EXACT' or 'SUFFIX'. tag: The tag to assign to the packet when the field matches with stringToMatch. *If 'field' is a multi map:*

{ "rules": [{"key": "Host", "value": "google.com", "matchingType": "SUFFIX", "tag": "GOOGLE"}, {"key": "Content-Type", "value": "amazon.com", "matchingType": "SUFFIX", "tag": "AMAZON"}, ...], }

key: The key to match in the multi map. 'value', 'matchingType' and 'tag' are the same as in the string case.

The 'tags_file' argument can be NULL and the matching rules can be added later with the pfwl_*_tags_add calls.

Return 0 if the loading was successful, 1 otherwise (e.g. error while parsing the json file, non existing file, etc...)

Function pfwl_field_tags_unload_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

void **pfwl_field_tags_unload_L7**(pfwl_state_t *state, *pfwl_field_id_t* field)

pfwl_field_tags_unload Unloads the associations between fields values and user-defined tags.

Unloads the associations between fields values and user-defined tags.

Parameters

- `state`: A pointer to the state of the library.
- `field`: The field identifier.

Function pfwl_get_L2_protocol_id

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l2_t **pfwl_get_L2_protocol_id**(**const** char ***const** *name*)

Returns the L2 protocol id corresponding to an L2 protocol string.

Return The L2 protocol id corresponding to an L2 protocol string.

Parameters

- `name`: The protocol string.

Function pfwl_get_L2_protocol_name

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ***pfwl_get_L2_protocol_name**(*pfwl_protocol_l2_t* protocol)

Returns the string represetation of an L2 protocol.

Return The string representation of the L2 protocol with id 'protocol'.

Parameters

- `protocol`: The L2 protocol identifier.

Function pfwl_get_L2_protocols_names

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ****const** **pfwl_get_L2_protocols_names**()
Returns the string represetations of the L2 protocols.

Return An array A of string, such that A[i] is the string representation of the L2 protocol with id 'i'.

Function pfwl_get_L3_protocol_id

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l3_t **pfwl_get_L3_protocol_id**(**const** char ***const** *name*)
Returns the L3 protocol id corresponding to an L3 protocol string.

Return The L3 protocol id corresponding to an L3 protocol string.

Parameters

- `name`: The protocol string.

Function pfwl_get_L3_protocol_name

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ***pfwl_get_L3_protocol_name**(*pfwl_protocol_l3_t protocol*)
Returns the string represetation of an L3 protocol.

Return The string representation of the L3 protocol with id 'protocol'.

Parameters

- `protocol`: The L3 protocol identifier.

Function pfwl_get_L3_protocols_names

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ****const pfwl_get_L3_protocols_names**()
Returns the string represetations of the L3 protocols.

Return An array A of string, such that A[i] is the string representation of the L3 protocol with id 'i'.

Function pfwl_get_L4_protocol_id

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l4_t **pfwl_get_L4_protocol_id**(**const** char ***const** *name*)
Returns the L4 protocol id corresponding to an L4 protocol string.

Return The L4 protocol id corresponding to an L4 protocol string.

Parameters

- `name`: The protocol string.

Function pfwl_get_L4_protocol_name

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ***pfwl_get_L4_protocol_name**(*pfwl_protocol_l4_t protocol*)
Returns the string represetation of an L4 protocol.

Return The string representation of the L4 protocol with id 'protocol'.

Parameters

- `protocol`: The L4 protocol identifier.

Function pfwl_get_L4_protocols_names

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ****const pfwl_get_L4_protocols_names**()
Returns the string represetations of the L4 protocols.

Return An array A of string, such that A[i] is the string representation of the L4 protocol with id 'i'.

Function pfwl_get_L7_field_id

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_field_id_t **pfwl_get_L7_field_id** (*pfwl_protocol_l7_t protocol*, **const** char ***field_name*)

Returns the id associated to a protocol field name.

Return The id associated to the protocol field with name 'field_name'.

Parameters

- `protocol`: The protocol.
- `field_name`: The name of the field.

Function pfwl_get_L7_field_name

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ***pfwl_get_L7_field_name** (*pfwl_field_id_t field*)

Returns the string represetation of a protocol field.

Return The string representation of the protocol field with id 'field'.

Parameters

- `field`: The protocol field identifier.

Function pfwl_get_L7_field_protocol

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l7_t **pfwl_get_L7_field_protocol** (*pfwl_field_id_t field*)

Returns the protocol associated to a field identifier.

Return The protocol associated to a field identifier.

Parameters

- `field`: The field identifier.

Function pfwl_get_L7_field_type

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_field_type_t **pfwl_get_L7_field_type**(*pfwl_field_id_t field*)
pfwl_field_type_get Returns the type of a field.

Returns the type of a field.

Return The type of 'field'.

Parameters

- `field`: The field.

Function pfwl_get_L7_protocol_id

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l7_t **pfwl_get_L7_protocol_id**(**const** char ***const** *name*)
Returns the L7 protocol id corresponding to an L7 protocol string.

Return The L7 protocol id corresponding to an L7 protocol string.

Parameters

- `name`: The protocol string.

Function pfwl_get_L7_protocol_name

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ***pfwl_get_L7_protocol_name**(*pfwl_protocol_l7_t protocol*)
Returns the string represetation of an L7 protocol.

Return The string representation of the protocol with id 'protocol'.

Parameters

- `protocol`: The L7 protocol identifier.

Function pfwl_get_L7_protocols_names

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ****const pfwl_get_L7_protocols_names**()
Returns the string represetations of the L7 protocols.

Return An array A of string, such that A[i] is the string representation of the L7 protocol with id 'i'.

Function pfwl_get_status_msg

- Defined in file_include_peafowl_peafowl.h

Function Documentation

const char ***pfwl_get_status_msg**(*pfwl_status_t status_code*)
Returns the string representing the status message associated to the specified status_code.

Return The status message.

Parameters

- `status_code`: The status code.

Function pfwl_guess_protocol

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_protocol_l7_t **pfwl_guess_protocol**(*pfwl_dissection_info_t identification_info*)
Guesses the protocol looking only at source/destination ports. This could be erroneous because sometimes protocols run over ports which are not their well-known ports.

Return Returns the possible matching protocol.

Parameters

- `identification_info`: Info about the identification done up to now (up to L4 parsing).

Function pfwl_has_protocol_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_has_protocol_L7** (*pfwl_dissection_info_t* *dissection_info, *pfwl_protocol_l7_t* protocol)
pfwl_has_protocol_L7 Checks if a specific L7 protocol has been identified in a given dissection info.

Checks if a specific L7 protocol has been identified in a given dissection info. ATTENTION: Please note that protocols are associated to flows and not to packets. For example, if for a given flow, the first packet carries IMAP data and the second packet carries SSL encrypted data, we will have:

For the first packet:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): 1
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): 0

For the second packet:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): 1
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): 1

For all the subsequent packets:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): 1
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): 1

Return 1 if the L7 protocol is carried by the flow, 0 otherwise.

Parameters

- `dissection_info`: The dissection info.
- `protocol`: The L7 protocol.

Function pfwl_http_get_header

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_http_get_header** (*pfwl_dissection_info_t* *dissection_info, **const** char *header_name, *pfwl_string_t* *header_value)
pfwl_http_get_header Extracts a specific HTTP header from the dissection info.

Return 0 if the http header was present, 1 otherwise. If 1 is returned, 'header_value' is not set.

Parameters

- `dissection_info`: The dissection info.
- `header_name`: The name of the header ('\0' terminated).
- `header_value`: The returned header value.

Function pfwl_init

- Defined in file_include_peafowl_peafowl.h

Function Documentation

pfwl_state_t *__pfwl_init__(void)
Initializes Peafowl. Initializes the library.

Return A pointer to the state of the library.

Function pfwl_init_flow_info

- Defined in file_include_peafowl_peafowl.h

Function Documentation

void __pfwl_init_flow_info__(pfwl_state_t *_state_, pfwl_flow_info_private_t *_flow_info_private_)
DEPRECATED. Initialize the flow informations passed as argument.

Parameters

- `state`: A pointer to the state of the library.
- `flow_info_private`: The private flow information, will be initialized by the library.

Function pfwl_protocol_l7_disable

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t __pfwl_protocol_l7_disable__(pfwl_state_t *_state_, _pfwl_protocol_l7_t_ _protocol_)
Disables an L7 protocol dissector.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `protocol`: The protocol to disable.

Function pfwl_protocol_l7_disable_all

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_protocol_l7_disable_all** (pfwl_state_t *_state_)
Disable all the protocol dissector.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

Function pfwl_protocol_l7_enable

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_protocol_l7_enable** (pfwl_state_t *_state_, *pfwl_protocol_l7_t* _protocol_)
Enables an L7 protocol dissector.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `protocol`: The protocol to enable.

Function pfwl_protocol_l7_enable_all

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_protocol_l7_enable_all** (pfwl_state_t *_state_)
Enables all the L7 protocol dissector.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

Function pfwl_set_expected_flows

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_set_expected_flows**(pfwl_state_t *_state_, uint32_t _flows_, _pfwl_flows_strategy_t_ _strategy_)
Sets the number of simultaneously active flows to be expected.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `flows`: The number of simultaneously active flows.
- `strategy`: If PFWL_FLOWS_STRATEGY_NONE, there will not be any limit to the number of simultaneously active flows. However, this could lead to slowdown when retrieving flow information. If PFWL_FLOWS_STRATEGY_SKIP, when that number of active flows is reached, if a new flow is created an error will be returned (PFWL_ERROR_MAX_FLOWS) and new flows will not be created. If PFWL_FLOWS_STRATEGY_EVICT, when when that number of active flows is reached, if a new flow is created the oldest flow will be evicted.

Function pfwl_set_flow_cleaner_callback

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_set_flow_cleaner_callback**(pfwl_state_t *_state_, pfwl_flow_cleaner_callback_t *_cleaner_)
DEPRECATED: Please use pfwl_set_flow_termination_callback. Sets the callback that will be called when a flow expires.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `cleaner`: The callback used to clear the user data.

Function pfwl_set_flow_termination_callback

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_set_flow_termination_callback**(pfwl_state_t **state*, pfwl_flow_termination_callback_t **cleaner*)

Sets the callback that will be called when a flow expires.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `cleaner`: The callback used to access flow information when it expires.

Function pfwl_set_max_trials

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_set_max_trials**(pfwl_state_t **state*, uint16_t *max_trials*)

Sets the maximum number of packets to use to identify the protocol. During the flow protocol identification, after this number of trials, if the library cannot decide between two or more protocols, one of them will be chosen, otherwise PFWL_PROTOCOL_UNKNOWN will be returned.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `max_trials`: Maximum number of trials. Zero will be consider as infinity.

Function pfwl_set_protocol_accuracy_L7

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_set_protocol_accuracy_L7**(pfwl_state_t **state*, *pfwl_protocol_l7_t* *protocol*, *pfwl_dissector_accuracy_t* *accuracy*)

Some L7 protocols dissectors (e.g. SIP) can be applied with a different level of accuracy (and of performance). By using this call the user can decide if running the dissector in its most accurate version (at the cost of a higher processing latency).

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `protocol`: The L7 protocol for which we want to change the accuracy.
- `accuracy`: The accuracy level.

Function pfwl_set_timestamp_unit

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_set_timestamp_unit** (pfwl_state_t *_state_, _pfwl_timestamp_unit_t_ _unit_)
Sets the unit of the timestamps used in the pfwl_dissect_* calls.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: The state of the library.
- `unit`: The unit of the timestamps.

Function pfwl_statistic_add

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_statistic_add** (pfwl_state_t *_state_, _pfwl_statistic_t_ _stat_)
pfwl_statistic_add Enables the computation of a specific flow statistic.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `stat`: The statistic to be enabled.

Function pfwl_statistic_remove

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_statistic_remove** (pfwl_state_t *_state_, _pfwl_statistic_t_ _stat_)
pfwl_statistic_remove Disables the computation of a specific flow statistic.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.
- `stat`: The statistic to be enabled.

Function pfwl_tcp_reordering_disable

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_tcp_reordering_disable**(pfwl_state_t *_state_)

If called, the library will not reorder out of order TCP packets. Out-of-order segments will be delivered to the dissectors as they arrive. This means that the dissector may not be able to identify the application protocol. Moreover, if there are callbacks saved for TCP based protocols, if TCP reordering is disabled, the extracted informations could be erroneous or incomplete.

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

Function pfwl_tcp_reordering_enable

- Defined in file_include_peafowl_peafowl.h

Function Documentation

uint8_t **pfwl_tcp_reordering_enable**(pfwl_state_t *_state_)

If enabled, the library will reorder out of order TCP packets (enabled by default).

Return 0 if succeeded, 1 otherwise.

Parameters

- `state`: A pointer to the state of the library.

Function pfwl_terminate

- Defined in file_include_peafowl_peafowl.h

Function Documentation

void **pfwl_terminate**(pfwl_state_t *_state_)

Terminates the library.

Parameters

- `state`: A pointer to the state of the library.

Defines

Define PFWL_MAX_L7_SUBPROTO_DEPTH

- Defined in file_include_peafowl_peafowl.h

Define Documentation

PFWL_MAX_L7_SUBPROTO_DEPTH
Maximum number of nested L7 protocols.

Define PFWL_TAGS_MAX

- Defined in file_include_peafowl_peafowl.h

Define Documentation

PFWL_TAGS_MAX
Maximum number of tags that can be associated to a packet.

Typedefs

Typedef pfwl_dissection_info_l2_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_dissection_info_l2* **pfwl_dissection_info_l2_t**
The result of the L2 identification process.

Typedef pfwl_dissection_info_l3_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_dissection_info_l3* **pfwl_dissection_info_l3_t**
The result of the L3 identification process.

Typedef pfwl_dissection_info_l4_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_dissection_info_l4* **pfwl_dissection_info_l4_t**
The result of the L4 identification process.

Typedef pfwl_dissection_info_l7_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_dissection_info_l7* **pfwl_dissection_info_l7_t**
The result of the L7 identification process.

Typedef pfwl_dissection_info_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_dissection_info* **pfwl_dissection_info_t**
The result of the identification process.

Typedef pfwl_field_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_field* **pfwl_field_t**
A generic field extracted by peafowl.

Typedef pfwl_flow_cleaner_callback_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

void() pfwl_flow_cleaner_callback_t (void *flow_udata)
Callback for flow cleaning. DEPRECATED: You should now use pfwl_flow_termination_callback_t This callback is called when the flow is expired and deleted. It can be used by the user to clear any data he/she associated to the flow through flow_info.udata.

Parameters

- `flow_udata`: A pointer to the user data specific to this flow.

Typedef pfwl_flow_info_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef struct *pfwl_flow_info* **pfwl_flow_info_t**
Public information about the flow.

Typedef pfwl_flow_termination_callback_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

void() pfwl_flow_termination_callback_t (pfwl_flow_info_t *flow_info)
Callback which is called when a flow terminates. This callback is called when the flow is expired and deleted. It can be used by the user to access flow information and to clear any data he/she associated to the flow.

Parameters

- `flow_info`: A pointer to the flow information.

Typedef pfwl_ip_addr_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef union *pfwl_ip_addr* **pfwl_ip_addr_t**
IP address.

An IP address.

Typedef pfwl_mmap_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef *pfwl_array_t* **pfwl_mmap_t**
A peafowl map (just an array of pairs at the moment).

Typedef pfwl_protocol_l2_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef enum *pfwl_datalink_type* **pfwl_protocol_l2_t**
L2 datalink protocols supported by peafowl. When adding a new protocol, please update the pfwl_l2_protocols_names array in parsing_l2.c

Typedef pfwl_protocol_l4_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef uint8_t **pfwl_protocol_l4_t**
L4 protocol. Values defined in include/netinet/in.h (IPPROTO_TCP, IPPROTO_UDP, IPPROTO_ICMP, etc. . .)

Typedef pfwl_status_t

- Defined in file_include_peafowl_peafowl.h

Typedef Documentation

typedef enum *pfwl_status* **pfwl_status_t**
Status of the identification process

4.7 C++ API

4.7.1 Peafowl C++ API

Class Hierarchy

File Hierarchy

Full API

Namespaces

Namespace peafowl

Contents

Classes

Functions

- *Function peafowl::convertPcapDlt*
- *Function peafowl::fieldGet*
- *Function peafowl::getL2ProtocolsNames*
- *Function peafowl::getL3ProtocolsNames*
- *Function peafowl::getL4ProtocolsNames*
- *Function peafowl::getL7FieldId*
- *Function peafowl::getL7FieldName*
- *Function peafowl::getL7FieldProtocol*
- *Function peafowl::getL7FieldType*
- *Function peafowl::getL7ProtocolsNames*

Typedefs

- *Typedef peafowl::Direction*
- *Typedef peafowl::DissectorAccuracy*
- *Typedef peafowl::FieldId*
- *Typedef peafowl::FieldMatching*
- *Typedef peafowl::FieldType*
- *Typedef peafowl::FlowsStrategy*
- *Typedef peafowl::Statistic*
- *Typedef peafowl::TimestampUnit*

Classes and Structs

Class DefragmentationOptions

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**DefragmentationOptions**
The *DefragmentationOptions* class describes options to customize *Peafowl*'s defragmentation routines.

Public Functions

DefragmentationOptions()
Constructor.

void **enableIPv4**(uint16_t *tableSize*)
Enables IPv4 defragmentation. It is enabled by default.

Parameters

- `tableSize`: The size of the table to be used to store IPv4 fragments informations.

void **enableIPv6**(uint16_t *tableSize*)
Enables IPv6 defragmentation. It is enabled by default.

Parameters

- `tableSize`: The size of the table to be used to store IPv6 fragments informations.

void **setPerHostMemoryLimitIPv4**(uint32_t *perHostMemoryLimit*)
Sets the amount of memory (in bytes) that a single host can use for IPv4 defragmentation.

Parameters

- `perHostMemoryLimit`: The maximum amount of memory that any IPv4 host can use.

void **setPerHostMemoryLimitIPv6**(uint32_t *perHostMemoryLimit*)
Sets the amount of memory (in bytes) that a single host can use for IPv6 defragmentation.

Parameters

- `perHostMemoryLimit`: The maximum amount of memory that any IPv6 host can use.

void **setTotalMemoryLimitIPv4**(uint32_t *totalMemoryLimit*)
Sets the total amount of memory (in bytes) that can be used for IPv4 defragmentation. If defragmentation is disabled and then enabled again, this function must be called again.

Parameters

- `totalMemoryLimit`: The maximum amount of memory that can be used for IPv4 defragmentation.

void **setTotalMemoryLimitIPv6**(uint32_t *totalMemoryLimit*)
Sets the total amount of memory (in bytes) that can be used for IPv6 defragmentation. If defragmentation is disabled and then enabled again, this function must be called again.

Parameters

- `totalMemoryLimit`: The maximum amount of memory that can be used for IPv6 defragmentation.

void **setReassemblyTimeoutIPv4**(uint8_t *timeoutSeconds*)
Sets the maximum time (in seconds) that can be spent to reassembly an IPv4 fragmented datagram. Is the maximum time gap between the first and last fragments of the datagram.

Parameters

- `timeoutSeconds`: The reassembly timeout.

void **setReassemblyTimeoutIPv6**(uint8_t *timeoutSeconds*)
Sets the maximum time (in seconds) that can be spent to reassembly an IPv6 fragmented datagram. Is the maximum time gap between the first and last fragments of the datagram.

Parameters

- `timeoutSeconds`: The reassembly timeout.

void **disableIPv4**()
Disables IPv4 defragmentation.

void **disableIPv6**()
Disables IPv6 defragmentation.

Class DissectionInfo

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**DissectionInfo**
The result of the identification process.

Public Functions

DissectionInfo(pfwl_dissection_info_t *dissectionInfo*, *Status* *status*)
Constructor.

Parameters

- dissectionInfo: The C dissection info.
- status: The status of the processing.

DissectionInfo &**operator=**(**const** pfwl_dissection_info_t &*rhs*)
Assignment operator.

Return The CPP dissection info.

Parameters

- rhs: The C dissection info.

ProtocolL7 **guessProtocol**() **const**
Guesses the protocol looking only at source/destination ports. This could be erroneous because sometimes protocols run over ports which are not their well-known ports.

Return Returns the possible matching protocol.

bool **hasProtocolL7**(*ProtocolL7* *protocol*) **const**
hasProtocolL7 Checks if a specific L7 protocol has been identified in a given dissection info.

Checks if a specific L7 protocol has been identified in a given dissection info. ATTENTION: Please note that protocols are associated to flows and not to packets. For example, if for a given flow, the first packet carries IMAP data and the second packet carries SSL encrypted data, we will have:

For the first packet:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): true
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): false

For the second packet:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): true
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): true

For all the subsequent packets:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): true
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): true

Return True if the L7 protocol is carried by the flow, false otherwise.

Parameters

- `protocol`: The L7 protocol.

Status **getStatus**() **const**
getStatus Returns the status of the processing.

Return The status of the processing.

DissectionInfoL2 **getL2**() **const**
getL2 Returns the L2 dissection information.

Return The L2 dissection information.

DissectionInfoL3 **getL3**() **const**
getL3 Returns the L3 dissection information.

Return The L3 dissection information.

DissectionInfoL4 **getL4**() **const**
getL4 Returns the L4 dissection information.

Return The L4 dissection information.

DissectionInfoL7 **getL7**() **const**
getL7 Returns the L7 dissection information.

Return The L7 dissection information.

FlowInfo **getFlowInfo**() **const**
getFlowInfo Returns the flow information.

Return The flow information.

const pfwl_dissection_info_t &**getNativeInfo**() **const**
Returns the C dissection info.

Return The C dissection info.

Class DissectionInfoL2

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**DissectionInfoL2**
The result of the L2 identification process.

Public Functions

DissectionInfoL2()
Constructor.

DissectionInfoL2(pfwl_dissection_info_l2_t *dissectionInfo*)
Copy constructor.

Parameters

- dissectionInfo: The information to be copied.

size_t **getLength**() **const**
Returns the length of the L2 header.

Return The length of the L2 header.

ProtocolL2 **getProtocol**() **const**
Returns the L2 protocol.

Return The L2 protocol.

pfwl_dissection_info_l2_t **getNative**() **const**
Returns the C representation of the L2 protocol.

Return The C representation of the L2 protocol.

Class DissectionInfoL3

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**DissectionInfoL3**
The result of the L3 identification process.

Public Functions

DissectionInfoL3()
Constructor.

DissectionInfoL3(pfwl_dissection_info_l3_t *dissectionInfo*)
Copy constructor.

Parameters

- dissectionInfo: The information to be copied.

size_t **getLength**() **const**
Returns the length of the L3 header.

Return The length of the L3 header.

size_t **getPayloadLength**() **const**
Returns the length of the L3 payload.

Return The length of the L3 payload.

IpAddress **getAddressSrc**() **const**
Returns the source address, in network byte order.

Return The source address, in network byte order.

IpAddress **getAddressDst**() **const**
Returns the destination address, in network byte order.

Return The destination address, in network byte order.

std::pair<**const** unsigned char*, size_t> **getRefragmentedPacket**() **const**
Returns the refragmented IP packet (if it was fragmented, starting from the first byte of L3 packet) and its length.

Return The refragmented IP packet (if it was fragmented, starting from the first byte of L3 packet) and its length.

ProtocolL3 **getProtocol**() **const**
Returns the L3 protocol.

Return The L3 protocol.

pfwl_dissection_info_l3_t **getNative**() **const**
Returns the C representation of the L3 protocol.

Return The C representation of the L3 protocol.

Class DissectionInfoL4

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**DissectionInfoL4**
The result of the L4 identification process.

Public Functions

DissectionInfoL4()
Constructor.

DissectionInfoL4(pfwl_dissection_info_l4_t *dissectionInfo*)
Copy constructor.

Parameters

- `dissectionInfo`: The information to be copied.

size_t **getLength**() **const**
Returns the length of the L4 header.

Return The length of the L4 header.

size_t **getPayloadLength**() **const**
Returns the length of the L4 payload.

Return The length of the L4 payload.

uint16_t **getPortSrc**() **const**
Returns the source port, in network byte order.

Return The source port, in network byte order.

uint16_t **getPortDst**() **const**
Returns the destination port, in network byte order.

Return The destination port, in network byte order.

Direction **getDirection**() **const**
Returns the packet direction (with respect to the source and destination addresses specified in the flow).

Return The packet direction.

std::pair<**const** unsigned char*, size_t> **getResegmentedPacket**() **const**
Returns the resegmented TCP payload and its length.

Return The resegmented TCP payload and its length.

ProtocolL4 **getProtocol**() **const**
Returns the L4 protocol.

Return The L4 protocol.

pfwl_dissection_info_l4_t **getNative**() **const**
Returns the C representation of the L4 protocol.

Return The C representation of the L4 protocol.

Class DissectionInfoL7

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**DissectionInfoL7**
The result of the L7 identification process.

Public Functions

DissectionInfoL7()
Constructor.

DissectionInfoL7(pfwl_dissection_info_l7_t *dissectionInfo*)
Copy constructor.

Parameters

- dissectionInfo: The information to be copied.

std::vector<*ProtocolL7*> **getProtocols**() **const**
Some L7 protocols may be carried by other L7 protocols. For example, Ethereum may be carried by JSON-RPC, which in turn may be carried by HTTP. If such a flow is found, we will have:

protocols[0] = HTTP

protocols[1] = JSON-RPC

protocols[2] = Ethereum

i.e., protocols are shown by the outermost to the innermost. Similarly, if Ethereum is carried by plain JSON-RPC, we would have:

protocols[0] = JSON-RPC

protocols[1] = Ethereum

This encapsulation can also hold over different packets of a given flow. E.g.IMAP over SSL has a few packet exchanged with plain IMAP and then the subsequent packets encapsulated within SSL. In such a case, the first IMAP packets will only have protocols[0] = IMAP. However, when the first SSL packet for the flow is received, we will have protocols[0] = IMAP and protocols[1] = SSL for that packet and for all the subsequent packets. Indeed, it is important to remark that protocols are associated to flows and not to packets. This call returns the list of L7 protocols identified for this packet.

Return The list of L7 protocols identified for this packet.

ProtocolL7 **getProtocol**() **const**
Returns the first protocol of the list, i.e. this call is equivalent to *getProtocols()*[0].

Return The first protocol of the list.

Field **getField**(*FieldId id*) **const**
Returns a specific protocol field.

Return The protocol field.

Parameters

- `id`: The identifier of the field.

std::vector<*Field*> **getFields**() **const**
Returns all the protocol fields.

Return All the protocol fields.

std::vector<std::string> **getTags**() **const**
Returns the tags associated to this packet.

Return The tags associated to this packet.

Field **httpGetHeader**(**const** char **headerName*) **const**
httpGetHeader Extracts a specific HTTP header from the dissection info.

Return The header value.

Parameters

- `headerName`: The name of the header ('\0' terminated).

pfwl_dissection_info_l7_t **getNative**() **const**
Returns the C representation of the L7 protocol.

Return The C representation of the L7 protocol.

Class Field

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**Field**
A generic field extracted by peafowl.

Public Functions

Field()
Constructs an empty field.

Field(pfwl_field_t *field*)
Copy constructor.

Parameters

- field: *Field* to be copied.

bool **isPresent**() **const**
Checks if the field was present in the packet.

Return True if the field was present, false otherwise.

std::string **getString**() **const**
Gets the field (as a string).

Return The field (as a string).

int64_t **getNumber**() **const**
Gets the field (as a number).

Return The field (as a number).

pfwl_field_t **getNative**() **const**
Gets the C representation of the field.

Return The C representation of the field.

Class FlowInfo

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**FlowInfo**
Public information about the flow.

Public Functions

FlowInfo()
Constructor.

FlowInfo(pfwl_flow_info_t *info*)
Copy constructor.

Parameters

- `info`: The information to be copied.

uint64_t **getId**() **const**
Returns the unique identifier of the flow. If multithreaded version is used, id is per-thread unique, i.e. two different flows, managed by two different threads may have the same id. If multithreaded *Peafowl* is used, the unique identifier will be the pair <thread_id, id>

Return The unique identifier of the flow.

uint16_t **getThreadId**() **const**
Returns the identifier of the thread that managed this flow.

Return The identifier of the thread that managed this flow.

IpAddress **getAddressSrc**() **const**
Returns the source address, in network byte order.

Return The source address, in network byte order.

IpAddress **getAddressDst**() **const**
Returns the destination address, in network byte order.

Return The destination address, in network byte order.

uint16_t **getPortSrc**() **const**
Returns the source port, in network byte order.

Return The source port, in network byte order.

uint16_t **getPortDst**() **const**
Returns the destination port, in network byte order.

Return The destination port, in network byte order.

ProtocolL2 **getProtocolL2**() **const**
Returns the L2 protocol.

Return The L2 protocol.

ProtocolL3 **getProtocolL3**() **const**
Returns the L3 protocol.

Return The L3 protocol.

ProtocolL4 **getProtocolL4**() **const**
Returns the L4 protocol.

Return The L4 protocol.

std::vector<*ProtocolL7*> **getProtocolsL7**() **const**
Some L7 protocols may be carried by other L7 protocols. For example, Ethereum may be carried by JSON-RPC, which in turn may be carried by HTTP. If such a flow is found, we will have:

protocols[0] = HTTP

protocols[1] = JSON-RPC

protocols[2] = Ethereum

i.e., protocols are shown by the outermost to the innermost. Similarly, if Ethereum is carried by plain JSON-RPC, we would have:

protocols[0] = JSON-RPC

protocols[1] = Ethereum

This encapsulation can also hold over different packets of a given flow. E.g.IMAP over SSL has a few packet exchanged with plain IMAP and then the subsequent packets encapsulated within SSL. In such a case, the first IMAP packets will only have protocols[0] = IMAP. However, when the first SSL packet for the flow is received, we will have protocols[0] = IMAP and protocols[1] = SSL for that packet and for all the subsequent packets. Indeed, it is important to remark that protocols are associated to flows and not to packets. This call returns the list of L7 protocols identified for this flow.

Return The list of L7 protocols identified for this flow.

double **getStatistic**(*Statistic* *stat*, *Direction* *dir*) **const**
Returns a flow statistic for a specific flow direction.

Return A flow statistic for a specific flow direction.

Parameters

- `stat`: The statistic.
- `dir`: The direction.

void ****getUserData**() **const**
Returns the user data associated to this flow.

Return The user data associated to this flow.

pfwl_flow_info_t **getNative**() **const**
Returns the C flow representation.

Return The C flow representation.

void **setUserData**(void **udata*)
Sets some user-specific data for this flow.

Parameters

- `udata`: User-specific data for this flow.

Class FlowManager

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**FlowManager**
The *FlowManager* class is a functor class, which is used to notify the user about some events concerning the flow (e.g. flow termination).

Public Functions

~FlowManager()

void **onTermination**(**const** *FlowInfo* &*info*)
Function which is called when a flow terminates. This function is called when the flow is expired and deleted. It can be used by the user to access flow information and to clear any data he/she associated to the flow. This function may be called by multiple threads concurrently. Any access to member variables should be appropriately managed by the implementer.

Parameters

- `info`: The flow information.

Class IpAddress

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**IpAddress**
IP Address.

Public Functions

IpAddress(pfwl_ip_addr *addr*, bool *isIPv6* = false)
Builds the IP address.

Parameters

- `addr`: The IP address.
- `isIPv6`: True if the address is an IPv6 address, false otherwise.

bool **isIPv4**() **const**
Checks if the address is an IPv4 address.

Return True if the address is an IPv4 address, false otherwise.

bool **isIPv6**() **const**
Checks if the address is an IPv6 address.

Return True if the address is an IPv6 address, false otherwise.

uint32_t **getIPv4**() **const**
Gets the address as an IPv4 address.

Return The IPv4 address.

struct in6_addr **getIPv6**() **const**
Gets the address as an IPv6 address.

Return The IPv6 address.

std::string **toString**() **const**
Returns a string representation of the IP address.

Return A string representation of the IP address.

Template Class Pair

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

template<typename **T**>
class peafowl::**Pair**
A peafowl pair.

Public Functions

Pair()
Constructs an empty pair.

Pair(*T first*, *T second*)
Constructs a pair.

Parameters

- `first`: The first element of the pair.
- `second`: The second element of the pair.

Class Peafowl

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**Peafowl**
This class is the *Peafowl* handler.

Public Functions

Peafowl()
Initializes *Peafowl*. Initializes the library.

~Peafowl()
Terminates the library.

void **setFlowManager**(*FlowManager* **flowManager*)
setFlowManager Sets the functor object which is called when the flow terminates.

Parameters

- `flowManager`: The functor object.

void **setExpectedFlows**(uint32_t *flows*, *FlowsStrategy strategy*)
Sets the number of simultaneously active flows to be expected.

Parameters

- `flows`: The number of simultaneously active flows.
- `strategy`: If PFWL_FLOWS_STRATEGY_NONE, there will not be any limit to the number of simultaneously active flows. However, this could lead to slowdown when retrieving flow information. If PFWL_FLOWS_STRATEGY_SKIP, when that number of active flows is reached, if a new flow is created an error will be returned (PFWL_ERROR_MAX_FLOWS) and new flows will not be created. If PFWL_FLOWS_STRATEGY_EVICT, when when that number of active flows is reached, if a new flow is created the oldest flow will be evicted.

void **setMaxTrials**(uint16_t *maxTrials*)
Sets the maximum number of packets to use to identify the protocol. During the flow protocol identification, after this number of trials, if the library cannot decide between two or more protocols, one of them will be chosen, otherwise PFWL_PROTOCOL_UNKNOWN will be returned.

Parameters

- `maxTrials`: Maximum number of trials. Zero will be consider as infinity.

void **setDefragmentationOptions**(**const** *DefragmentationOptions* &*options*)
setDefragmentationOptions Sets the IPv4/IPv6 defragmentation options.

Parameters

- `options`: The IPv4/IPv6 defragmentation options.

void **tcpReorderingEnable**()
If enabled, the library will reorder out of order TCP packets (enabled by default).

void **tcpReorderingDisable**()
If called, the library will not reorder out of order TCP packets. Out-of-order segments will be delivered to the dissectors as they arrive. This means that the dissector may not be able to identify the application protocol. Moreover, if there are callbacks saved for TCP based protocols, if TCP reordering is disabled, the extracted informations could be erroneous or incomplete.

void **protocolL7Enable**(*ProtocolL7 protocol*)
Enables an L7 protocol dissector.

Parameters

- `protocol`: The protocol to enable.

void **protocolL7Disable**(*ProtocolL7 protocol*)
Disables an L7 protocol dissector.

Parameters

- `protocol`: The protocol to disable.

void **protocolL7EnableAll**()
Enables all the L7 protocol dissector.

void **protocolL7DisableAll**()
Disable all the protocol dissector.

void **setTimestampUnit**(*TimestampUnit unit*)
Sets the unit of the timestamps used in the dissect* calls.

Parameters

- `unit`: The unit of the timestamps.

DissectionInfo **dissectFromL2**(**const** std::string &*pkt*, double *timestamp*, *ProtocolL2 datalinkType*)
Dissects the packet starting from the beginning of the L2 (datalink) header.

Return The result of the dissection from L2 to L7.

Parameters

- `pkt`: A string containing the packet.
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.
- `datalinkType`: The datalink type. You can convert a PCAP datalink type to a *Peafowl* datalink type by calling the function 'convertPcapDlt'.

DissectionInfo **dissectFromL3**(**const** std::string &*pkt*, double *timestamp*)
Dissects the packet starting from the beginning of the L3 (IP) header.

Return The result of the dissection from L3 to L7.

Parameters

- `pkt`: A string containing the packet (starting from the IP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.

DissectionInfo **dissectFromL4**(**const** std::string &*pkt*, double *timestamp*, **const** *DissectionInfo* &*info*)
Dissects the packet starting from the beginning of the L4 (UDP or TCP) header.

Return The result of the dissection from L3 to L7.

Parameters

- `pkt`: A string containing the packet (from the start of TCP/UDP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.
- `info`: The dissection information about L3 header.

DissectionInfo **dissectL2**(**const** std::string &*pkt*, pfwl_protocol_l2_t *datalinkType*)
Extracts from the packet the L2 information.

Return The result of the L2 dissection.

Parameters

- `pkt`: A string containing the packet.
- `datalinkType`: The datalink type. They match 1:1 the pcap datalink types. You can convert a PCAP datalink type to a *Peafowl* datalink type by calling the function 'convertPcapDlt'.

DissectionInfo **dissectL3**(**const** std::string &*pkt*, double *timestamp*)
Extracts from the packet the L3 information.

Return The result of the L3 dissection.

Parameters

- `pkt`: A string containing the packet (from the start of the IP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.

DissectionInfo **dissectL4**(**const** std::string *&pkt*, double *timestamp*, **const** *DissectionInfo* *&info*, FlowInfoPrivate *&flowInfoPrivate*)

Extracts from the packet the L4 information.

Return The result of the L4 dissection.

Parameters

- `pkt`: A string containing the packet (from the start of the TCP/UDP header).
- `timestamp`: The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.
- `info`: The dissection information about L3 header. L4 protocol must be specified by the caller as well.
- `flowInfoPrivate`: Will be filled by this call.

DissectionInfo **dissectL7**(**const** std::string *&pkt*, **const** *DissectionInfo* *&info*, FlowInfoPrivate *&flowInfoPrivate*)

Extracts from the packet the L7 information. Before calling it, a check on L4 protocol should be done and the function should be called only if the packet is TCP or UDP. It should be used if the application already called dissectL7 or if the application already has the concept of 'flow'. In this case the first time that the flow is passed to the call, flow_info_private must be initialized with pfwl_init_flow_info(...) and stored with the flow already present in the application. With this call, information in dissection_info->flow are only set for L7 packets and bytes.

Return The result of the L7 dissection.

Parameters

- `pkt`: A string containing the packet (from the start of application data).
- `info`: The dissection information about L3 and L4 headers.
- `flowInfoPrivate`: The private information about the flow. It must be stored by the user.

void **fieldAddL7**(*FieldId field*)

Enables the extraction of a specific L7 field for a given protocol. When a protocol is identified, the default behavior is to not inspect the packets belonging to that flow anymore and keep simply returning the same protocol identifier.

If at least one field extraction is enabled for a certain protocol, then we keep inspecting all the new packets of that flow to extract such field. Moreover, if the application protocol uses TCP, then we have the additional cost of TCP reordering for all the segments. Is highly recommended to enable TCP reordering if it is not already enabled (remember that is enabled by default). Otherwise the informations extracted could be erroneous/incomplete.

Please note that this is only a suggestion given by the user to peafowl, and that in some cases the dissector could still extract the field, even if this has not been requested by the user. Indeed, in some cases the extraction of some fields may be needed for the correct identification of the protocol.

Parameters

- `field`: The field to extract.

void **fieldRemoveL7**(*FieldId field*)

Disables the extraction of a specific L7 protocol field.

Parameters

- `field`: The field identifier.

void **setProtocolAccuracyL7** (*ProtocolL7 protocol*, *DissectorAccuracy accuracy*)

Some L7 protocols dissectors (e.g. SIP) can be applied with a different level of accuracy (and of performance). By using this call the user can decide if running the dissector in its most accurate version (at the cost of a higher processing latency).

Parameters

- `protocol`: The L7 protocol for which we want to change the accuracy.
- `accuracy`: The accuracy level.

void **fieldTagsLoadL7** (*FieldId field*, **const** char **tagsFile*)

fieldTagsLoadL7 Loads the associations between fields values and user-defined tags.

Loads the associations between fields values and user-defined tags. { "rules": [{"value": "google.com", "matchingType": "SUFFIX", "tag": "GOOGLE"}, {"value": "amazon.com", "matchingType": "SUFFIX", "tag": "AMAZON"}, ...], }

Parameters

- `field`: The field identifier.
- `tagsFile`: The name of the JSON file containing associations between fields values and tags. The structure of the JSON file depends from the type of 'field'. *If 'field' is a string:*

value: Is the string to be matched against the field. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well. matchingType: Can be 'PREFIX', 'EXACT' or 'SUFFIX'. tag: The tag to assign to the packet when the field matches with stringToMatch. *If 'field' is a multi map:*

{ "rules": [{"key": "Host", "value": "google.com", "matchingType": "SUFFIX", "tag": "GOOGLE"}, {"key": "Content-Type", "value": "amazon.com", "matchingType": "SUFFIX", "tag": "AMAZON"}, ...], }

key: The key to match in the multi map. 'value', 'matchingType' and 'tag' are the same as in the string case.

The 'tagsFile' argument can be NULL and the matching rules can be added later with the *TagsAdd calls.

void **fieldStringTagsAddL7** (*FieldId field*, **const** std::string &*value*, *FieldMatching matchingType*, **const** std::string &*tag*)

pfwl_field_string_tags_add Adds a tag matching rule for a specific field.

Adds a tag matching rule for a specific string field.

Parameters

- `field`: The field identifier.
- `value`: Is the string to be matched against the field. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well.
- `matchingType`: Can be 'PREFIX', 'EXACT' or 'SUFFIX'.
- `tag`: The tag to assign to the packet when the field matches with 'value'.

void **fieldMmapTagsAddL7** (*FieldId field*, **const** std::string &*key*, **const** std::string &*value*, *FieldMatching matchingType*, **const** std::string &*tag*)

fieldMmapTagsAddL7 Adds a tag matching rule for a specific field.

Adds a tag matching rule for a specific multimap field.

Parameters

- `field`: The field identifier.

- `key`: The key of the multimap value. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well.
- `value`: The value of the multimap value. The comparison will always be case insensitive. I.e. if searching for 'BarFoo', 'barfoo' and 'BaRfOo' will match as well.
- `matchingType`: Can be 'PREFIX', 'EXACT' or 'SUFFIX'.
- `tag`: The tag to assign to the packet when the field matches with 'value'.

void **fieldTagsUnloadL7** (*FieldId field*)
fieldTagsUnloadL7 Unloads the associations between fields values and user-defined tags.

Unloads the associations between fields values and user-defined tags.

Parameters

- `field`: The field identifier.

void **statisticAdd** (*Statistic stat*)
Enables the computation of a specific flow statistic.

Return 0 if succeeded, 1 otherwise.

Parameters

- `stat`: The statistic to be enabled.

void **statisticRemove** (*Statistic stat*)
Disables the computation of a specific flow statistic.

Return 0 if succeeded, 1 otherwise.

Parameters

- `stat`: The statistic to be enabled.

Class ProtocolL2

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class `peafowl::`**ProtocolL2**
L2 datalink protocols supported by peafowl. When adding a new protocol, please update the pfwl_l2_protocols_names array in parsing_l2.c

Public Functions

ProtocolL2 (pfwl_protocol_l2_t *protocol*)
Copy constructor

Parameters

- `protocol`: The protocol to copy.

ProtocolL2 (**const** std::string &*protocol*)
Build the protocol starting from its name.

Parameters

- `protocol`: The protocol name.

const std::string &**getName**() **const**
Returns the name of the protocol.

Return The name of the protocol.

pfwl_protocol_l2_t **getId**() **const**
Returns the identifier of the protocol.

Return The identifier of the protocol.

operator pfwl_protocol_l2_t() **const**
Accesses the protocol.

Friends

friend bool **operator==**(**const** *ProtocolL2* &*p1*, **const** pfwl_protocol_l2_t &*p2*)
Checks if two protocols are equal.

Return True if the two protocols are equal, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

friend bool **operator!=**(**const** *ProtocolL2* &*p1*, **const** pfwl_protocol_l2_t &*p2*)
Checks if two protocols are different.

Return True if the two protocols are different, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

Class ProtocolL3

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class `peafowl::`**ProtocolL3**
L3 (IP) protocol.

Public Functions

ProtocolL3(pfwl_protocol_l3_t *protocol*)
Copy constructor

Parameters

- `protocol`: The protocol to copy.

ProtocolL3(**const** std::string &*protocol*)
Build the protocol starting from its name.

Parameters

- `protocol`: The protocol name.

const std::string &**getName**() **const**
Returns the name of the protocol.

Return The name of the protocol.

pfwl_protocol_l3_t **getId**() **const**
Returns the identifier of the protocol.

Return The identifier of the protocol.

operator pfwl_protocol_l3_t() **const**
Accesses the protocol.

Friends

friend bool **operator==**(**const** *ProtocolL3* &*p1*, **const** pfwl_protocol_l3_t &*p2*)
Checks if two protocols are equal.

Return True if the two protocols are equal, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

friend bool **operator!=**(**const** *ProtocolL3* &*p1*, **const** pfwl_protocol_l3_t &*p2*)
Checks if two protocols are different.

Return True if the two protocols are different, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

Class ProtocolL4

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**ProtocolL4**
L4 protocol. Values defined in include/netinet/in.h (IPPROTO_TCP, IPPROTO_UDP, IPPROTO_ICMP, etc. . .)

Public Functions

ProtocolL4(pfwl_protocol_l4_t *protocol*)
Copy constructor

Parameters

- protocol: The protocol to copy.

ProtocolL4(**const** std::string &*protocol*)
Build the protocol starting from its name.

Parameters

- protocol: The protocol name.

const std::string &**getName**() **const**
Returns the name of the protocol.

Return The name of the protocol.

pfwl_protocol_l4_t **getId**() **const**
Returns the identifier of the protocol.

Return The identifier of the protocol.

operator pfwl_protocol_l4_t() **const**
Accesses the protocol.

Friends

friend bool **operator==**(**const** *ProtocolL4* &*p1*, **const** pfwl_protocol_l4_t &*p2*)
Checks if two protocols are equal.

Return True if the two protocols are equal, false otherwise.

Parameters

- p1: The first protocol.
- p2: The second protocol.

friend bool **operator!=**(**const** *ProtocolL4* &*p1*, **const** pfwl_protocol_l4_t &*p2*)
Checks if two protocols are different.

Return True if the two protocols are different, false otherwise.

Parameters

- p1: The first protocol.
- p2: The second protocol.

friend bool **operator==**(**const** *ProtocolL4* &*p1*, **const** int &*p2*)
Checks if two protocols are equal.

Return True if the two protocols are equal, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

friend bool **operator!=**(**const** *ProtocolL4* &*p1*, **const** int &*p2*)
Checks if two protocols are different.

Return True if the two protocols are different, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

Class ProtocolL7

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class `peafowl::ProtocolL7`
L7 (application level) protocol.

Public Functions

ProtocolL7(pfwl_protocol_l7_t *protocol*)
Copy constructor

Parameters

- `protocol`: The protocol to copy.

ProtocolL7(**const** std::string &*protocol*)
Build the protocol starting from its name.

Parameters

- `protocol`: The protocol name.

const std::string &**getName**() **const**
Returns the name of the protocol.

Return The name of the protocol.

pfwl_protocol_l7_t **getId**() **const**
Returns the identifier of the protocol.

Return The identifier of the protocol.

operator pfwl_protocol_l7_t() **const**
Accesses the protocol.

Friends

friend bool **operator==**(**const** *ProtocolL7* &*p1*, **const** pfwl_protocol_l7_t &*p2*)
Checks if two protocols are equal.

Return True if the two protocols are equal, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

friend bool **operator!=**(**const** *ProtocolL7* &*p1*, **const** pfwl_protocol_l7_t &*p2*)
Checks if two protocols are different.

Return True if the two protocols are different, false otherwise.

Parameters

- `p1`: The first protocol.
- `p2`: The second protocol.

Class Status

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class `peafowl::`**Status**
Status of the identification process

Public Functions

Status(pfwl_status_t *status*)
Copy constructor.

Parameters

- `status`: The C status.

std::string **getMessage**() **const**
Returns a string message associated to this status.

Return A string message associated to this status.

bool **isError**() **const**
Checks if this status is an error status.

Return True if this status is an error status, false otherwise.

Class String

- Defined in file_include_peafowl_peafowl.hpp

Class Documentation

class peafowl::**String**
A string as represented by peafowl.

Public Functions

String()
Constructs an empty string.

String(pfwl_string_t *string*)
Copy constructor.

Parameters

- string: *Field* to be copied.

const unsigned char ***getValue**() **const**
Returns the buffer containing the protocol field.

Return The buffer containing the protocol field.

size_t **getLength**() **const**
Returns the length of the buffer.

Return The length of the buffer.

Functions

Function peafowl::convertPcapDlt

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

ProtocolL2 peafowl::**convertPcapDlt**(int *dlt*)
convertPcapDlt Converts a pcap datalink type (which can be obtained with the pcap_datalink(...) call), to a pfwl_datalink_type_t.

Return The peafowl datalink type. PFWL_DLT_NOT_SUPPORTED is returned if the specified datalink type is not supported by peafowl.

Parameters

- dlt: The pcap datalink type.

Function peafowl::fieldGet

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

Field peafowl::**fieldGet**(std::vector<*Field*> *fields*, *FieldId id*)
fieldGet Extracts a specific field from a list of fields.

Return The extracted field.

Parameters

- fields: The list of fields.
- id: The field identifier.

Function peafowl::getL2ProtocolsNames

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

std::vector<std::string> peafowl::**getL2ProtocolsNames**()
Returns the string represetations of the L2 protocols.

Return An array A of string, such that A[i] is the string representation of the L2 protocol with id 'i'.

Function peafowl::getL3ProtocolsNames

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

std::vector<std::string> peafowl::**getL3ProtocolsNames**()
Returns the string represetations of the L3 protocols.

Return An array A of string, such that A[i] is the string representation of the L3 protocol with id 'i'.

Function peafowl::getL4ProtocolsNames

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

std::vector<std::string> `peafowl::getL4ProtocolsNames()`
Returns the string represetations of the L4 protocols.

Return An array A of string, such that A[i] is the string representation of the L4 protocol with id 'i'.

Function peafowl::getL7FieldId

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

FieldId `peafowl::getL7FieldId(`*ProtocolL7 protocol*, std::string *fieldName*`)`
Returns the id associated to a protocol field name.

Return The id associated to the protocol field with name 'fieldName'.

Parameters

- `protocol`: protocol The protocol.
- `fieldName`: The name of the field.

Function peafowl::getL7FieldName

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

std::string `peafowl::getL7FieldName(`*FieldId field*`)`
Returns the string represetation of a protocol field.

Return The string representation of the protocol field with id 'field'.

Parameters

- `field`: The protocol field identifier.

Function peafowl::getL7FieldProtocol

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

ProtocolL7 `peafowl::`**`getL7FieldProtocol`**`(`*FieldId field*`)`
Returns the protocol associated to a field identifier.

Return The protocol associated to a field identifier.

Parameters

- `field`: The field identifier.

Function peafowl::getL7FieldType

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

FieldType `peafowl::`**`getL7FieldType`**`(`*FieldId field*`)`
getL7FieldType Returns the type of a field.

Returns the type of a field.

Return The type of 'field'.

Parameters

- `field`: The field.

Function peafowl::getL7ProtocolsNames

- Defined in file_include_peafowl_peafowl.hpp

Function Documentation

std::vector<std::string> `peafowl::`**`getL7ProtocolsNames`**`()`
Returns the string represetations of the L7 protocols.

Return An array A of string, such that A[i] is the string representation of the L7 protocol with id 'i'.

Typedefs

Typedef peafowl::Direction

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_direction_t peafowl::**Direction**
Possible packet directions.

Typedef peafowl::DissectorAccuracy

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_dissector_accuracy_t peafowl::**DissectorAccuracy**
The accuracy of the dissector.

Typedef peafowl::FieldId

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_field_id_t peafowl::**FieldId**
Protocol fields which can be extracted by peafowl.

Typedef peafowl::FieldMatching

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_field_matching_t peafowl::**FieldMatching**
The field matching rule to be used in tagging packets.

Typedef peafowl::FieldType

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_field_type_t peafowl::**FieldType**
Possible types for peafowl fields.

Typedef peafowl::FlowsStrategy

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_flows_strategy_t `peafowl::`**FlowsStrategy**
Possible strategies to adopt when there are too many flows in the flows table.

Typedef peafowl::Statistic

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_statistic_t `peafowl::`**Statistic**
A generic statistic for the flow. While a field is something related to the packet, a statistic is something related to the flow (e.g. packets per second, etc...).

Typedef peafowl::TimestampUnit

- Defined in file_include_peafowl_peafowl.hpp

Typedef Documentation

typedef pfwl_timestamp_unit_t `peafowl::`**TimestampUnit**
Units of timestamps used by *Peafowl*.

4.8 Python API

pypeafowl	Peafowl python API

4.8.1 pypeafowl

Description

Peafowl python API

Classes

Direction	Members:
DissectionInfo	Extracted information about the packet.
DissectionInfoL2	L2 information about the packet.

continues on next page

Table 2 – continued from previous page

DissectionInfoL3	L3 information about the packet.
DissectionInfoL4	L4 information about the packet.
DissectionInfoL7	L7 information about the packet.
Field	This class represents a protocol field extracted from the packet.
FieldId	Members:
FlowInfo	Information about the flow.
FlowManager	This class wraps the function which is called when the flow terminates.
FlowsStrategy	Members:
IpAddress	IP address.
Peafowl	Handle to the Peafowl library.
ProtocolL2	L2 protocol.
ProtocolL3	L3 protocol.
ProtocolL4	L4 protocol.
ProtocolL7	L7 protocol.
Statistic	Members:
Status	Status of the dissection.
String	This class represents a string extracted from the packet.

pypeafowl.Direction

class pypeafowl.**Direction**

Members:

OUTBOUND [] From source address to destination address.

INBOUND [] From destination address to source address.

Direction.INBOUND
Direction.OUTBOUND
Direction.name
Direction.value

pypeafowl.Direction.INBOUND

Direction.**INBOUND = <Direction.INBOUND: 1>**

pypeafowl.Direction.OUTBOUND

Direction.**OUTBOUND = <Direction.OUTBOUND: 0>**

pypeafowl.Direction.name

property Direction.**name**

pypeafowl.Direction.value

property Direction.**value**

---

pypeafowl.DissectionInfo

class pypeafowl.**DissectionInfo**
Extracted information about the packet.

DissectionInfo.getL2(self)	Returns the L2 dissection info.
DissectionInfo.getL3(self)	Returns the L3 dissection info.
DissectionInfo.getL4(self)	Returns the L4 dissection info.
DissectionInfo.getL7(self)	Returns the L7 dissection info.
DissectionInfo.getStatus(self)	Returns the status of the processing.
DissectionInfo.guessProtocol(self)	Guesses the protocol looking only at source/destination ports.
DissectionInfo.hasProtocolL7(self, protocol)	Checks if a specific L7 protocol has been identified in a given dissection info.

pypeafowl.DissectionInfo.getL2

DissectionInfo.**getL2**(*self:* pypeafowl.DissectionInfo) → *pypeafowl.DissectionInfoL2*
Returns the L2 dissection info.

Returns: The L2 dissection info.

pypeafowl.DissectionInfo.getL3

DissectionInfo.**getL3**(*self:* pypeafowl.DissectionInfo) → *pypeafowl.DissectionInfoL3*
Returns the L3 dissection info.

Returns: The L3 dissection info.

pypeafowl.DissectionInfo.getL4

DissectionInfo.**getL4**(*self:* pypeafowl.DissectionInfo) → *pypeafowl.DissectionInfoL4*
Returns the L4 dissection info.

Returns: The L4 dissection info.

pypeafowl.DissectionInfo.getL7

DissectionInfo.**getL7**(*self:* pypeafowl.DissectionInfo) → *pypeafowl.DissectionInfoL7*
Returns the L7 dissection info.

Returns: The L7 dissection info.

pypeafowl.DissectionInfo.getStatus

DissectionInfo.**getStatus**(*self:* pypeafowl.DissectionInfo) → peafowl::Status
Returns the status of the processing.

Returns: The status of the processing.

pypeafowl.DissectionInfo.guessProtocol

DissectionInfo.**guessProtocol**(*self:* pypeafowl.DissectionInfo) → peafowl::ProtocolL7
Guesses the protocol looking only at source/destination ports. This could be erroneous because sometimes protocols run over ports which are not their well-known ports.

Returns: The possible matching protocol.

pypeafowl.DissectionInfo.hasProtocolL7

DissectionInfo.**hasProtocolL7**(*self:* *pypeafowl.DissectionInfo*, *protocol:* *peafowl::ProtocolL7*) → bool
Checks if a specific L7 protocol has been identified in a given dissection info. ATTENTION: Please note that protocols are associated to flows and not to packets. For example, if for a given flow, the first packet carries IMAP data and the second packet carries SSL encrypted data, we will have:

For the first packet:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): True
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): False

For the second packet:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): True

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): True

For all the subsequent packets:

- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_IMAP): True
- pfwl_has_protocol_L7(info, PFWL_PROTO_L7_SSL): True

Args:

protocol The L7 protocol.

Returns: True if the protocol was present, false otherwise.

pypeafowl.DissectionInfoL2

class pypeafowl.**DissectionInfoL2**
L2 information about the packet.

DissectionInfoL2.getLength(self)	Returns the length of the L2 header.
DissectionInfoL2.getProtocol(self)	Returns the L2 protocol.

pypeafowl.DissectionInfoL2.getLength

DissectionInfoL2.**getLength**(*self:* pypeafowl.DissectionInfoL2) → int
Returns the length of the L2 header.

Returns: The length of the L2 header.

pypeafowl.DissectionInfoL2.getProtocol

DissectionInfoL2.**getProtocol**(*self:* pypeafowl.DissectionInfoL2) → peafowl::ProtocolL2
Returns the L2 protocol.

Returns: The L2 protocol.

pypeafowl.DissectionInfoL3

class pypeafowl.**DissectionInfoL3**
L3 information about the packet.

DissectionInfoL3.getAddressDst(self)	Returns the source address.
DissectionInfoL3.getAddressSrc(self)	Returns the source address.
DissectionInfoL3.getLength(self)	Returns the length of the L3 header.
DissectionInfoL3.getPayloadLength(self)	Returns the L3 payload length.
DissectionInfoL3.getProtocol(self)	Returns the L3 protocol.
DissectionInfoL3.getRefragmentedPacket(self)	Returns the IP refragmented packet and its length.

pypeafowl.DissectionInfoL3.getAddressDst

DissectionInfoL3.**getAddressDst**(*self:* pypeafowl.DissectionInfoL3) → *pypeafowl.IpAddress*

Returns the source address.

Returns: The source address.

pypeafowl.DissectionInfoL3.getAddressSrc

DissectionInfoL3.**getAddressSrc**(*self:* pypeafowl.DissectionInfoL3) → *pypeafowl.IpAddress*

Returns the source address.

Returns: The source address.

pypeafowl.DissectionInfoL3.getLength

DissectionInfoL3.**getLength**(*self:* pypeafowl.DissectionInfoL3) → int

Returns the length of the L3 header.

Returns: The length of the L3 header.

pypeafowl.DissectionInfoL3.getPayloadLength

DissectionInfoL3.**getPayloadLength**(*self:* pypeafowl.DissectionInfoL3) → int

Returns the L3 payload length.

Returns: The L3 payload length.

pypeafowl.DissectionInfoL3.getProtocol

DissectionInfoL3.**getProtocol**(*self:* pypeafowl.DissectionInfoL3) → peafowl::ProtocolL3

Returns the L3 protocol.

Returns: The L3 protocol.

pypeafowl.DissectionInfoL3.getRefragmentedPacket

DissectionInfoL3.**getRefragmentedPacket**(*self:* pypeafowl.DissectionInfoL3) → Tuple[int, int]

Returns the IP refragmented packet and its length.

Returns: The IP refragmented packet and its length.

pypeafowl.DissectionInfoL4

class pypeafowl.**DissectionInfoL4**
L4 information about the packet.

DissectionInfoL4.getDirection(self)	Returns the packet direction with respect to the flow.
DissectionInfoL4.getLength(self)	Returns the length of the L4 header.
DissectionInfoL4.getPayloadLength(self)	Returns the length of the L4 payload..
DissectionInfoL4.getPortDst(self)	Returns the destination port, in network byte order.
DissectionInfoL4.getPortSrc(self)	Returns the source port, in network byte order.
DissectionInfoL4.getProtocol(self)	Returns the L4 protocol.
DissectionInfoL4.getResegmentedPacket(self)	Returns the resegmented TCP packet and its length.

pypeafowl.DissectionInfoL4.getDirection

DissectionInfoL4.**getDirection**(*self:* pypeafowl.DissectionInfoL4) → *pypeafowl.Direction*
Returns the packet direction with respect to the flow.

Returns: The packet direction with respect to the flow.

pypeafowl.DissectionInfoL4.getLength

DissectionInfoL4.**getLength**(*self:* pypeafowl.DissectionInfoL4) → int
Returns the length of the L4 header.

Returns: The length of the L4 header.

pypeafowl.DissectionInfoL4.getPayloadLength

DissectionInfoL4.**getPayloadLength**(*self:* pypeafowl.DissectionInfoL4) → int
Returns the length of the L4 payload..

Returns: The length of the L4 payload.

pypeafowl.DissectionInfoL4.getPortDst

DissectionInfoL4.**getPortDst**(*self:* pypeafowl.DissectionInfoL4) → int
Returns the destination port, in network byte order.

Returns: The destination port, in network byte order.

pypeafowl.DissectionInfoL4.getPortSrc

DissectionInfoL4.**getPortSrc**(*self:* pypeafowl.DissectionInfoL4) → int
Returns the source port, in network byte order.

Returns: The source port, in network byte order.

pypeafowl.DissectionInfoL4.getProtocol

DissectionInfoL4.**getProtocol**(*self:* pypeafowl.DissectionInfoL4) → peafowl::ProtocolL4
Returns the L4 protocol.

Returns: The L4 protocol.

pypeafowl.DissectionInfoL4.getResegmentedPacket

DissectionInfoL4.**getResegmentedPacket**(*self:* pypeafowl.DissectionInfoL4) → Tuple[int, int]
Returns the resegmented TCP packet and its length.

Returns: The resegmented TCP packet and its length.

pypeafowl.DissectionInfoL7

class pypeafowl.**DissectionInfoL7**
L7 information about the packet.

DissectionInfoL7.getField(self, id)	Returns a field associated to this packet.
DissectionInfoL7.getFields(self)	Returns the fields associated to this packet.
DissectionInfoL7.getProtocol(self)	Returns the first protocol of the list, i.e. this call is equivalent to calling getProtocols[0].
DissectionInfoL7.getProtocols(self)	Some L7 protocols may be carried by other L7 protocols.
DissectionInfoL7.getTags(self)	Returns the tags associated to this packet.
DissectionInfoL7.httpGetHeader(self, headerName)	Extracts a specific HTTP header from the dissection info.

pypeafowl.DissectionInfoL7.getField

DissectionInfoL7.**getField**(*self:* pypeafowl.DissectionInfoL7, *id:* pypeafowl.FieldId) → *pypeafowl.Field*
Returns a field associated to this packet.

Args:

id The identifier of the field.

Returns: The field associated to this packet.

pypeafowl.DissectionInfoL7.getFields

DissectionInfoL7.**getFields**(*self:* pypeafowl.DissectionInfoL7) → List[*pypeafowl.Field*]
Returns the fields associated to this packet.

Returns: The fields associated to this packet.

pypeafowl.DissectionInfoL7.getProtocol

DissectionInfoL7.**getProtocol**(*self:* pypeafowl.DissectionInfoL7) → peafowl::ProtocolL7
Returns the first protocol of the list, i.e. this call is equivalent to calling getProtocols[0].

Returns: The first protocol of the list.

pypeafowl.DissectionInfoL7.getProtocols

DissectionInfoL7.**getProtocols**(*self:* pypeafowl.DissectionInfoL7) → List[peafowl::ProtocolL7]
Some L7 protocols may be carried by other L7 protocols. For example, Ethereum may be carried by JSON-RPC, which in turn may be carried by HTTP. If such a flow is found, we will have:

protocols[0] = HTTP

protocols[1] = JSON-RPC

protocols[2] = Ethereum

i.e., protocols are shown by the outermost to the innermost. Similarly, if Ethereum is carried by plain JSON-RPC, we would have:

protocols[0] = JSON-RPC

protocols[1] = Ethereum

This encapsulation can also hold over different packets of a given flow. E.g.IMAP over SSL has a few packet exchanged with plain IMAP and then the subsequent packets encapsulated within SSL. In such a case, the first IMAP packets will only have protocols[0] = IMAP. However, when the first SSL packet for the flow is received, we will have protocols[0] = IMAP and protocols[1] = SSL for that packet and for all the subsequent packets. Indeed, it is important to remark that protocols are associated to flows and not to packets. This call returns the list of L7 protocols identified for this packet.

Returns: The list of L7 protocols identified for this packet.

pypeafowl.DissectionInfoL7.getTags

DissectionInfoL7.**getTags**(*self:* pypeafowl.DissectionInfoL7) → List[str]
Returns the tags associated to this packet.

Returns: The tags associated to this packet.

pypeafowl.DissectionInfoL7.httpGetHeader

DissectionInfoL7.**httpGetHeader**(*self:* pypeafowl.DissectionInfoL7, *headerName:* *str*) → *pypeafowl.Field*

Extracts a specific HTTP header from the dissection info.

Args:

headerName The name of the header ('0' terminated).

Returns: The header value.

pypeafowl.Field

class pypeafowl.**Field**

This class represents a protocol field extracted from the packet.

Field.getNumber(self)	Returns this field as a number.
Field.getString(self)	Returns this field as a string.
Field.isPresent(self)	This function checks if this field is present.

pypeafowl.Field.getNumber

Field.**getNumber**(*self:* pypeafowl.Field) → int

Returns this field as a number.

Returns: The number.

pypeafowl.Field.getString

Field.**getString**(*self:* pypeafowl.Field) → str

Returns this field as a string.

Returns: The string.

pypeafowl.Field.isPresent

Field.**isPresent**(*self:* pypeafowl.Field) → bool
This function checks if this field is present.

Returns: True if the field is present, False otherwise.

pypeafowl.FieldId

class pypeafowl.**FieldId**
Members:

HTTP_URL

FieldId.HTTP_URL
FieldId.name
FieldId.value

pypeafowl.FieldId.HTTP_URL

FieldId.**HTTP_URL = <FieldId.HTTP_URL: 43>**

pypeafowl.FieldId.name

property FieldId.**name**

pypeafowl.FieldId.value

property FieldId.**value**

---

pypeafowl.FlowInfo

class pypeafowl.**FlowInfo**
Information about the flow.

FlowInfo.getAddressDst(self)	Returns the destination address.
FlowInfo.getAddressSrc(self)	Returns the source address.
FlowInfo.getId(self)	Returns a unique identifier of the flow.
FlowInfo.getPortDst(self)	Returns the destination port.
FlowInfo.getPortSrc(self)	Returns the source port.
FlowInfo.getProtocolL2(self)	Returns the L2 protocol of this flow.
FlowInfo.getProtocolL3(self)	Returns the L3 protocol of this flow.
FlowInfo.getProtocolL4(self)	Returns the L4 protocol of this flow.
FlowInfo.getProtocolsL7(self)	Some L7 protocols may be carried by other L7 protocols.
FlowInfo.getStatistic(self, stat, dir)	Returns a statistic of this flow for a specific direction.

continues on next page

Table 13 – continued from previous page

FlowInfo.getThreadId(self)	Returns the identifier of the thread that managed this flow.
FlowInfo.getUserData(self)	Returns the user data associated to this flow.
FlowInfo.setUserData(self, udata)	Associates to this flow some user data.

pypeafowl.FlowInfo.getAddressDst

FlowInfo.**getAddressDst**(*self:* pypeafowl.FlowInfo) → *pypeafowl.IpAddress*

Returns the destination address.

Returns: The destination address.

pypeafowl.FlowInfo.getAddressSrc

FlowInfo.**getAddressSrc**(*self:* pypeafowl.FlowInfo) → *pypeafowl.IpAddress*

Returns the source address.

Returns: The source address.

pypeafowl.FlowInfo.getId

FlowInfo.**getId**(*self:* pypeafowl.FlowInfo) → int

Returns a unique identifier of the flow. If multithreaded version is used, id is per-thread unique, i.e. two different flows, managed by two different threads may have the same id. If multithreaded Peafowl is used, the unique identifier will be the pair <thread_id, id>

Returns: The identifier of the flow.

pypeafowl.FlowInfo.getPortDst

FlowInfo.**getPortDst**(*self:* pypeafowl.FlowInfo) → int

Returns the destination port.

Returns: The destination port.

pypeafowl.FlowInfo.getPortSrc

FlowInfo.**getPortSrc**(*self:* pypeafowl.FlowInfo) → int

Returns the source port.

Returns: The source port.

pypeafowl.FlowInfo.getProtocolL2

FlowInfo.**getProtocolL2**(*self:* pypeafowl.FlowInfo) → peafowl::ProtocolL2
Returns the L2 protocol of this flow.

Returns: The L2 protocol of this flow.

pypeafowl.FlowInfo.getProtocolL3

FlowInfo.**getProtocolL3**(*self:* pypeafowl.FlowInfo) → peafowl::ProtocolL3
Returns the L3 protocol of this flow.

Returns: The L3 protocol of this flow.

pypeafowl.FlowInfo.getProtocolL4

FlowInfo.**getProtocolL4**(*self:* pypeafowl.FlowInfo) → peafowl::ProtocolL4
Returns the L4 protocol of this flow.

Returns: The L4 protocol of this flow.

pypeafowl.FlowInfo.getProtocolsL7

FlowInfo.**getProtocolsL7**(*self:* pypeafowl.FlowInfo) → List[peafowl::ProtocolL7]
Some L7 protocols may be carried by other L7 protocols. For example, Ethereum may be carried by JSON-RPC, which in turn may be carried by HTTP. If such a flow is found, we will have:

protocols[0] = HTTP

protocols[1] = JSON-RPC

protocols[2] = Ethereum

i.e., protocols are shown by the outermost to the innermost. Similarly, if Ethereum is carried by plain JSON-RPC, we would have:

protocols[0] = JSON-RPC

protocols[1] = Ethereum

This encapsulation can also hold over different packets of a given flow. E.g.IMAP over SSL has a few packet exchanged with plain IMAP and then the subsequent packets encapsulated within SSL. In such a case, the first IMAP packets will only have protocols[0] = IMAP. However, when the first SSL packet for the flow is received, we will have protocols[0] = IMAP and protocols[1] = SSL for that packet and for all the subsequent packets. Indeed, it is important to remark that protocols are associated to flows and not to packets. This call returns the list of L7 protocols of this flow.

Returns: The list of L7 protocols of this flow.

pypeafowl.FlowInfo.getStatistic

FlowInfo.**getStatistic**(*self:* pypeafowl.FlowInfo, *stat:* pypeafowl.Statistic, *dir:* pypeafowl.Direction) → float

Returns a statistic of this flow for a specific direction.

Args:

stat The type of statistic to get.

dir The direction.

Returns: The required statistics of this flow.

pypeafowl.FlowInfo.getThreadId

FlowInfo.**getThreadId**(*self:* pypeafowl.FlowInfo) → int

Returns the identifier of the thread that managed this flow.

Returns: The identifier of the thread that managed this flow.

pypeafowl.FlowInfo.getUserData

FlowInfo.**getUserData**(*self:* pypeafowl.FlowInfo) → capsule

Returns the user data associated to this flow.

Returns: The user data associated to this flow.

pypeafowl.FlowInfo.setUserData

FlowInfo.**setUserData**(*self:* pypeafowl.FlowInfo, *udata: capsule*) → None

Associates to this flow some user data.

Args:

udata The user data.

pypeafowl.FlowManager

class pypeafowl.**FlowManager**

This class wraps the function which is called when the flow terminates.

FlowManager.onTermination(self, info)	This function is called when the flow terminates

pypeafowl.FlowManager.onTermination

FlowManager.**onTermination**(*self: pypeafowl.FlowManager*, *info: peafowl::FlowInfo*) → None
This function is called when the flow terminates

Args:

info The flow information.

pypeafowl.FlowsStrategy

class pypeafowl.**FlowsStrategy**
Members:

FLOWS_STRATEGY_NONE

FLOWS_STRATEGY_SKIP

FLOWS_STRATEGY_EVICT

FlowsStrategy.FLOWS_STRATEGY_EVICT
FlowsStrategy.FLOWS_STRATEGY_NONE
FlowsStrategy.FLOWS_STRATEGY_SKIP
FlowsStrategy.name
FlowsStrategy.value

pypeafowl.FlowsStrategy.FLOWS_STRATEGY_EVICT

FlowsStrategy.**FLOWS_STRATEGY_EVICT = <FlowsStrategy.FLOWS_STRATEGY_EVICT: 2>**

pypeafowl.FlowsStrategy.FLOWS_STRATEGY_NONE

FlowsStrategy.**FLOWS_STRATEGY_NONE = <FlowsStrategy.FLOWS_STRATEGY_NONE: 0>**

pypeafowl.FlowsStrategy.FLOWS_STRATEGY_SKIP

FlowsStrategy.**FLOWS_STRATEGY_SKIP = <FlowsStrategy.FLOWS_STRATEGY_SKIP: 1>**

pypeafowl.FlowsStrategy.name

property FlowsStrategy.**name**

pypeafowl.FlowsStrategy.value

property FlowsStrategy.**value**

---

pypeafowl.IpAddress

class pypeafowl.**IpAddress**
IP address.

IpAddress.getIPv4(self)	Returns the IPv4 address, in network byte order.
IpAddress.getIPv6(self)	Returns the IPv6 address, in network byte order.
IpAddress.isIPv4(self)	Checks if this address is an IPv4 address.
IpAddress.isIPv6(self)	Checks if this address is an IPv6 address.
IpAddress.toString(self)	Returns a string representation of the IP address, in host byte order.

pypeafowl.IpAddress.getIPv4

IpAddress.**getIPv4**(*self:* pypeafowl.IpAddress) → int
Returns the IPv4 address, in network byte order.

Returns: The IPv4 address, in network byte order.

pypeafowl.IpAddress.getIPv6

IpAddress.**getIPv6**(*self:* pypeafowl.IpAddress) → in6_addr
Returns the IPv6 address, in network byte order.

Returns: The IPv6 address, in network byte order.

pypeafowl.IpAddress.isIPv4

IpAddress.**isIPv4**(*self:* pypeafowl.IpAddress) → bool
Checks if this address is an IPv4 address.

Returns: True if the address is an IPv4 address, false otherwise.

pypeafowl.IpAddress.isIPv6

IpAddress.**isIPv6**(*self:* pypeafowl.IpAddress) → bool
Checks if this address is an IPv6 address.

Returns: True if the address is an IPv4 address, false otherwise.

pypeafowl.IpAddress.toString

IpAddress.**toString**(*self:* pypeafowl.IpAddress) → str

Returns a string representation of the IP address, in host byte order.

Returns: A string representation of the IP address, in host byte order.

pypeafowl.Peafowl

class pypeafowl.**Peafowl**

Handle to the Peafowl library.

Peafowl.dissectFromL2(self, pkt, ts, dlt)	Dissects the packet starting from the beginning of the L2 (datalink) header.
Peafowl.dissectFromL3(self, pkt, ts)	Dissects the packet starting from the beginning of the L3 (IP) header.
Peafowl.fieldAddL7(self, field)	Enables the extraction of a specific L7 field for a given protocol.
Peafowl.fieldRemoveL7(self, arg0)	Enables the extraction of a specific L7 field for a given protocol.
Peafowl.setExpectedFlows(self, flows, strategy)	Sets the number of simultaneously active flows to be expected.
Peafowl.setFlowManager(self, flowManager)	Sets the functor object which is called when the flow terminates.
Peafowl.setTimestampUnit(self, unit)	Sets the unit of the timestamps used in the dissect* calls.

pypeafowl.Peafowl.dissectFromL2

Peafowl.**dissectFromL2**(*self:* pypeafowl.Peafowl, *pkt: str*, *ts: float*, *dlt:* pypeafowl.ProtocolL2) → *pypeafowl.DissectionInfo*

Dissects the packet starting from the beginning of the L2 (datalink) header.

Args:

pkt A string containing the packet.

ts The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.

dlt The datalink type. You can convert a PCAP datalink type to a Peafowl datalink type by calling the function 'convertPcapDlt'.

Returns: The result of the dissection from L2 to L7.

pypeafowl.Peafowl.dissectFromL3

Peafowl.**dissectFromL3** (*self:* pypeafowl.Peafowl, *pkt: str*, *ts: float*) → *pypeafowl.DissectionInfo*

Dissects the packet starting from the beginning of the L3 (IP) header.

Args:

pkt A string containing the packet.

ts The current time. The time unit depends on the timers used by the caller and can be set through the setTimestampUnit call. By default it is assumed that the timestamps unit is 'seconds'.

Returns: The result of the dissection from L3 to L7.

pypeafowl.Peafowl.fieldAddL7

Peafowl.**fieldAddL7** (*self:* pypeafowl.Peafowl, *field:* pypeafowl.FieldId) → None

Enables the extraction of a specific L7 field for a given protocol. When a protocol is identified, the default behavior is to not inspect the packets belonging to that flow anymore and keep simply returning the same protocol identifier.

If at least one field extraction is enabled for a certain protocol, then we keep inspecting all the new packets of that flow to extract such field. Moreover, if the application protocol uses TCP, then we have the additional cost of TCP reordering for all the segments. Is highly recommended to enable TCP reordering if it is not already enabled (remember that is enabled by default). Otherwise the informations extracted could be erroneous/incomplete.

Please note that this is only a suggestion given by the user to peafowl, and that in some cases the dissector could still extract the field, even if this has not been requested by the user. Indeed, in some cases the extraction of some fields may be needed for the correct identification of the protocol.

Args:

field The field to extract.

pypeafowl.Peafowl.fieldRemoveL7

Peafowl.**fieldRemoveL7** (*self:* pypeafowl.Peafowl, *arg0:* pypeafowl.FieldId) → None

Enables the extraction of a specific L7 field for a given protocol. When a protocol is identified, the default behavior is to not inspect the packets belonging to that flow anymore and keep simply returning the same protocol identifier.

If at least one field extraction is enabled for a certain protocol, then we keep inspecting all the new packets of that flow to extract such field. Moreover, if the application protocol uses TCP, then we have the additional cost of TCP reordering for all the segments. Is highly recommended to enable TCP reordering if it is not already enabled (remember that is enabled by default). Otherwise the informations extracted could be erroneous/incomplete.

Please note that this is only a suggestion given by the user to peafowl, and that in some cases the dissector could still extract the field, even if this has not been requested by the user. Indeed, in some cases the extraction of some fields may be needed for the correct identification of the protocol.

Args:

field The field to extract.

pypeafowl.Peafowl.setExpectedFlows

Peafowl.**setExpectedFlows**(*self:* pypeafowl.Peafowl, *flows:* *int*, *strategy:* pypeafowl.FlowsStrategy) → None

Sets the number of simultaneously active flows to be expected.

Args:

flows The number of simultaneously active flows.

strategy If PFWL_FLOWS_STRATEGY_NONE, there will not be any limit to the number of simultaneously active flows. However, this could lead to slowdown when retrieving flow information. If PFWL_FLOWS_STRATEGY_SKIP, when that number of active flows is reached, if a new flow is created an error will be returned (PFWL_ERROR_MAX_FLOWS) and new flows will not be created. If PFWL_FLOWS_STRATEGY_EVICT, when when that number of active flows is reached, if a new flow is created the oldest flow will be evicted.

pypeafowl.Peafowl.setFlowManager

Peafowl.**setFlowManager**(*self:* pypeafowl.Peafowl, *flowManager:* pypeafowl.FlowManager) → None

Sets the functor object which is called when the flow terminates.

Args:

flowManager The functor object.

pypeafowl.Peafowl.setTimestampUnit

Peafowl.**setTimestampUnit**(*self:* pypeafowl.Peafowl, *unit: pfwl_timestamp_unit_t*) → None

Sets the unit of the timestamps used in the dissect* calls.

Args:

unit The unit of the timestamps.

pypeafowl.ProtocolL2

class pypeafowl.**ProtocolL2**

L2 protocol.

ProtocolL2.getId(self)	Returns the protocol identifier.
ProtocolL2.getName(self)	Returns the protocol name.

pypeafowl.ProtocolL2.getId

ProtocolL2.**getId**(*self:* pypeafowl.ProtocolL2) → pfwl_datalink_type
Returns the protocol identifier.

Returns: The protocol identifier.

pypeafowl.ProtocolL2.getName

ProtocolL2.**getName**(*self:* pypeafowl.ProtocolL2) → str
Returns the protocol name.

Returns: The protocol name.

pypeafowl.ProtocolL3

class pypeafowl.**ProtocolL3**
L3 protocol.

ProtocolL3.getId(self)	Returns the protocol identifier.
ProtocolL3.getName(self)	Returns the protocol name.

pypeafowl.ProtocolL3.getId

ProtocolL3.**getId**(*self:* pypeafowl.ProtocolL3) → pfwl_protocol_l3_t
Returns the protocol identifier.

Returns: The protocol identifier.

pypeafowl.ProtocolL3.getName

ProtocolL3.**getName**(*self:* pypeafowl.ProtocolL3) → str
Returns the protocol name.

Returns: The protocol name.

pypeafowl.ProtocolL4

class pypeafowl.**ProtocolL4**
L4 protocol.

ProtocolL4.getId(self)	Returns the protocol identifier.
ProtocolL4.getName(self)	Returns the protocol name.

pypeafowl.ProtocolL4.getId

ProtocolL4.**getId**(*self:* pypeafowl.ProtocolL4) → int
Returns the protocol identifier.

Returns: The protocol identifier.

pypeafowl.ProtocolL4.getName

ProtocolL4.**getName**(*self:* pypeafowl.ProtocolL4) → str
Returns the protocol name.

Returns: The protocol name.

pypeafowl.ProtocolL7

class pypeafowl.**ProtocolL7**
L7 protocol.

ProtocolL7.getId(self)	Returns the protocol identifier.
ProtocolL7.getName(self)	Returns the protocol name.

pypeafowl.ProtocolL7.getId

ProtocolL7.**getId**(*self:* pypeafowl.ProtocolL7) → pfwl_protocol_l7_t
Returns the protocol identifier.

Returns: The protocol identifier.

pypeafowl.ProtocolL7.getName

ProtocolL7.**getName**(*self:* pypeafowl.ProtocolL7) → str
Returns the protocol name.

Returns: The protocol name.

pypeafowl.Statistic

class pypeafowl.**Statistic**
Members:

PACKETS

BYTES

TIMESTAMP_FIRST

TIMESTAMP_LAST

L4_TCP_RTT_SYN_ACK

L4_TCP_COUNT_SYN

L4_TCP_COUNT_FIN

L4_TCP_COUNT_RST

L4_TCP_COUNT_RETRANSMISSIONS

L4_TCP_COUNT_ZERO_WINDOW

L4_TCP_WINDOW_SCALING

L7_PACKETS

L7_BYTES

STAT_NUM

Statistic.BYTES
Statistic.L4_TCP_COUNT_FIN
Statistic.L4_TCP_COUNT_RETRANSMISSIONS
Statistic.L4_TCP_COUNT_RST
Statistic.L4_TCP_COUNT_SYN
Statistic.L4_TCP_COUNT_ZERO_WINDOW
Statistic.L4_TCP_RTT_SYN_ACK
Statistic.L4_TCP_WINDOW_SCALING
Statistic.L7_BYTES
Statistic.L7_PACKETS
Statistic.PACKETS
Statistic.STAT_NUM
Statistic.TIMESTAMP_FIRST
Statistic.TIMESTAMP_LAST
Statistic.name
Statistic.value

pypeafowl.Statistic.BYTES

Statistic.**BYTES = <Statistic.BYTES: 1>**

pypeafowl.Statistic.L4_TCP_COUNT_FIN

Statistic.**L4_TCP_COUNT_FIN = <Statistic.L4_TCP_COUNT_FIN: 6>**

pypeafowl.Statistic.L4_TCP_COUNT_RETRANSMISSIONS

Statistic.**L4_TCP_COUNT_RETRANSMISSIONS = <Statistic.L4_TCP_COUNT_RETRANSMISSIONS: 8>**

pypeafowl.Statistic.L4_TCP_COUNT_RST

Statistic.**L4_TCP_COUNT_RST = <Statistic.L4_TCP_COUNT_RST: 7>**

pypeafowl.Statistic.L4_TCP_COUNT_SYN

Statistic.**L4_TCP_COUNT_SYN = <Statistic.L4_TCP_COUNT_SYN: 5>**

pypeafowl.Statistic.L4_TCP_COUNT_ZERO_WINDOW

Statistic.**L4_TCP_COUNT_ZERO_WINDOW = <Statistic.L4_TCP_COUNT_ZERO_WINDOW: 9>**

pypeafowl.Statistic.L4_TCP_RTT_SYN_ACK

Statistic.**L4_TCP_RTT_SYN_ACK = <Statistic.L4_TCP_RTT_SYN_ACK: 4>**

pypeafowl.Statistic.L4_TCP_WINDOW_SCALING

Statistic.**L4_TCP_WINDOW_SCALING = <Statistic.L4_TCP_WINDOW_SCALING: 10>**

pypeafowl.Statistic.L7_BYTES

Statistic.**L7_BYTES = <Statistic.L7_BYTES: 12>**

pypeafowl.Statistic.L7_PACKETS

Statistic.**L7_PACKETS = <Statistic.L7_PACKETS: 11>**

pypeafowl.Statistic.PACKETS

Statistic.**PACKETS = <Statistic.PACKETS: 0>**

pypeafowl.Statistic.STAT_NUM

Statistic.**STAT_NUM = <Statistic.STAT_NUM: 13>**

pypeafowl.Statistic.TIMESTAMP_FIRST

Statistic.**TIMESTAMP_FIRST = <Statistic.TIMESTAMP_FIRST: 2>**

pypeafowl.Statistic.TIMESTAMP_LAST

Statistic.**TIMESTAMP_LAST = <Statistic.TIMESTAMP_LAST: 3>**

pypeafowl.Statistic.name

property Statistic.**name**

pypeafowl.Statistic.value

property Statistic.**value**

---

pypeafowl.Status

class pypeafowl.**Status**
Status of the dissection.

Status.getMessage(self)	Returns the message associated to this status.

pypeafowl.Status.getMessage

Status.**getMessage**(*self:* pypeafowl.Status) → str
Returns the message associated to this status.

Returns: The message associated to this status.

pypeafowl.String

class pypeafowl.**String**
This class represents a string extracted from the packet.

String.getLength(self)	Returns the string length
String.getValue(self)	Returns the string content

pypeafowl.String.getLength

String.**getLength**(*self:* pypeafowl.String) → int
Returns the string length

Returns: The string length

pypeafowl.String.getValue

String.**getValue**(*self:* pypeafowl.String) → int
Returns the string content

Returns: The string content

Functions

convertPcapDlt(arg0)	Converts a pcap datalink type (which can be obtained with the pcap_datalink(...) call), to a pfwl_datalink_type_t.

pypeafowl.convertPcapDlt

pypeafowl.**convertPcapDlt**(*arg0: int*) → *pypeafowl.ProtocolL2*
Converts a pcap datalink type (which can be obtained with the pcap_datalink(...) call), to a pfwl_datalink_type_t.

Args:

dlt The pcap datalink type.

Returns: The peafowl datalink type. PFWL_DLT_NOT_SUPPORTED is returned if the specified datalink type is not supported by peafowl.

4.9 Adding New Protocols

If you want to add the support for new protocols, you can do it by following some simple steps. Protocols must be added in the **C** implementation. They will then be automatically available to the **C++** and **Python** interfaces as well.

For example, if you want to add the support for the Telnet protocol:

1. Define the protocol and give to it the next available numeric identifier (file `include/peafowl/peafowl.h`).

```
/** Protocols. **/
typedef enum{
  PFWL_PROTO_L7_HTTP = 0,
  PFWL_PROTO_L7_BGP,
  PFWL_PROTO_L7_SMTP,
  PFWL_PROTO_L7_POP3,
```

(continues on next page)

(continued from previous page)

```
  PFWL_PROTO_L7_TELNET, // <--- Insert this line right before 'PFWL_PROTO_L7_
↪NUM' to assign an identifier to the new protocol
  PFWL_PROTO_L7_NUM
}pfwl_protocol_l7_t;
```

2. Create a new inspector, by implementing a C function with the following signature and semantic:

```
uint8_t check_telnet(pfwl_state_t* state,                              ///< The␣
↪state of the library.
                     const unsigned char* app_data,                    ///< A␣
↪pointer to the beginning of the
                                                                       ///<␣
↪application (layer 7) data.
                     uint32_t data_length,                             ///< The␣
↪lenght of the application data.
                     pfwl_dissection_info_t* dissection_info,          ///<␣
↪Dissection data collected up to L4.
                     pfwl_flow_info_private_t* flow_info_private);     ///<␣
↪Information about the flow the packet belongs to
```

The function declaration must be put in `include/peafowl/inspectors/inspectors.h`, while its definition can be put in a new source file in inspectors folder (e.g. `src/inspectors/telnet.c`).

This function, after analyzing "app_data" and using the knowledge about the current state of the flow can return one of four different values:

- PFWL_PROTOCOL_MATCHES: If the protocol matches for sure
- PFWL_PROTOCOL_NO_MATCHES: If the protocol doesn't matches for sure
- PFWL_PROTOCOL_MORE_DATA_NEEDED: If the inspector needs more data to be sure that the protocol matches
- PFWL_PROTOCOL_ERROR: If an error occurred

You can look at one of the existing inspectors to see some examples.

If the inspector needs to store information about the application flow, add an appropriate structure in the `pfwl_flow_info_private_t` structure (in file `include/peafowl/flow_table.h`). This data will be flow-specific and will be preserved between different packets for the same flow.

```
typedef struct pfwl_flow_info_private{
    [...]
    /*************************************/
    /* Protocol inspectors support data */
    /*************************************/
    [...]
    /*************************************/
    /** Telnet Tracking informations. **/
    /*************************************/
    void* telnet_state;
}pfwl_flow_info_private_t;
```

These data can be then used by the inspector by accessing the parameter `flow_info_private`.

3. In file `src/parsing_l7.c`, create a descriptor for the new protocol, by adding a descriptor struct to the `protocols_descriptors` array. The descriptor has the following fields:

- name: A string representation for the protocol (e.g. `"TELNET"`).

- dissector: The function to detect the if the packet is carrying data for the given protocol. (Described in point 2)
- transport: PFWL_L7_TRANSPORT_TCP if the protocol can only be carried by TCP packets, PFWL_L7_TRANSPORT_UDP if the protocol can only be carried by UDP packets, PFWL_L7_TRANSPORT_TCP_OR_UDP if the protocol can be carried by both TCP and UDP packets.
- dependencies_fields: Array of fields (of other protocols) needed to identify this protocol. Last value in the array must always be PFWL_FIELDS_L7_NUM

4. If the protocol usually run on one or more predefined ports, specify the association between the ports and the protocol identifier (`src/parsing_l7.c`).

   ATTENTION: The ports must be specified in Network Byte Order! Check `include/peafowl/inspectors/protocols_identifiers.h` for some example.

```
static const pfwl_protocol_l7 const
  pfwl_known_ports_tcp[PFWL_MAX_UINT_16+1] =
    {[0 ... PFWL_MAX_UINT_16] = PFWL_PROTOCOL_UNKNOWN
    ,[port_http] = PFWL_PROTOCOL_HTTP
    ,[port_bgp] = PFWL_PROTOCOL_BGP
    ,[port_smtp_1] = PFWL_PROTOCOL_SMTP
    ,[port_smtp_2] = PFWL_PROTOCOL_SMTP
    ,[port_pop3] = PFWL_PROTOCOL_POP3
    ,[port_telnet] = PFWL_PROTOCOL_TELNET};
```

   In this way, when the framework receives a protocol on the telnet port, it will first check if the carried protocol is Telnet and, if this is not the case, it will check the other protocols. In a similar way, if the protocol runs over UDP instead of TCP, you have to add it to `pfwl_known_ports_udp` array.

5. Add unit tests for the protocol. Suppose you are adding the support for the `TELNET` protocol. First, you need to add a `testTelnet.cpp` file under `./test/`. This file will be automatically compiled and executed when the tests are run. In this file you should put the code for checking that the protocol `TELNET` is correctly identified. You can check correctness in the way you prefer.

   However, the suggested (and simplest) way is the following:

   - Place a .pcap file containing some packets for the protocol under the `./test/pcaps` folder. Suppose this file is called `TELNET.pcap`. If the protocol is a TCP-based protocol, check that the .pcap contains the SYN packets which open the TCP connection.
   - Peafowl relies on [googletest](https://github.com/google/googletest). In the `testTelnet.cpp` file you can check the correctness of the identification by running the following code:

```
#include "common.h"

TEST(TELNETTest, Generic) {
    std::vector<uint> protocols;
    getProtocols("./pcaps/TELNET.pcap", protocols);
    EXPECT_EQ(protocols[PFWL_PROTOCOL_TELNET], (uint) 42);
}
```

   Where `42` is the number of `TELNET` packets you expect to be identified by the protocol inspector. Of course, you can check the correctness of the protocol in any other way.

6. Recompile the framework with testing option enabled and run the tests to check that the unit tests succeed:

```
$ cd build
$ rm -rf *
$ cmake -DENABLE_TESTS=ON ../
```

(continues on next page)

(continued from previous page)

```
$ make
$ make test
```

If you implemented the support for some other protocols please consider opening a Pull Request.

4.10 Adding New Fields

As described before, besides protocol identification, it is possible to seamlessly provide data and metadata carried by the protocols to the application that uses the framework. To add this capability to existing inspector you need to follow some simple steps.

For example, let us assume that POP3 dissector is available in Peafowl but no field extraction capabilities are provided yet. To extract POP3 fields the following steps must be followed:

1. Add to the `field_L7_descriptors` array in the `src/parsing_l7.c` source file, the descriptors for the fields you want to extract, for example:

```
typedef enum{
  [...]
  {PFWL_PROTO_L7_POP3 , "SRC_ADDR", PFWL_FIELD_TYPE_STRING, "POP3 source␣
↪address"},
  {PFWL_PROTO_L7_POP3 , "DST_ADDR", PFWL_FIELD_TYPE_STRING, "POP3␣
↪destination address"},
  [...]
}pfwl_field_id_t;
```

   The elements specified are defined as follows:

   - The first element is the protocol for which we want to extract the field
   - The second element is the short name of the field. Enum values called `PFWL_FIELDS_L7_POP3_SRC_ADDR` and `PFWL_FIELDS_L7_POP3_DST_ADDR` will be automatically generated when compiling the code, and could be used by the user inside the application.
   - The third element is the type of field. In this case both addresses are strings.
   - The fourth and last field is a textual description of the field (just used for documentation purposes).

2. In the protocol dissector, set the fields once you find them in the packet. Different types of fields are supported, and some helper functions (e.g. `pfwl_field_string_set(...)`) are provided to simplify setting the fields. Peafowl guarantees that the fields are valid only until the next packet for the same flow is received. Accordingly, to avoid data copying, for STRING fields you can just set a pointer to the position in the original packet. Instead of copying the data. Moreover, you could inspect and process some parts of the packet only if the user required that field.

   E.g. suppose you want to set a field corresponding to the source mail address:

```
if(pfwl_protocol_field_required(state, PFWL_FIELDS_L7_POP3_SRC_ADDR)){
  pfwl_field_string_set(dissection_info, PFWL_FIELDS_L7_POP3_SRC_ADDR,␣
↪[pointer to the address start in the packet], [length of the address])
}
```

3. Now, inside the application that is using Peafowl, it is possible to check the fields that have been extracted. Helper function are provided.

For example:

```
if(pfwl_dissect_from_L2(state, packet, header.caplen, time(NULL), dlt, &
↪dissection_info) >= PFWL_STATUS_OK){
  if(dissection_info.l7.protocol == PFWL_PROTOCOL_POP3){
    pfwl_string_t src_addr;
    if(!pfwl_field_string_get(&dissection_info.l7.protocol_fields, PFWL_
↪FIELDS_L7_POP3_SRC_ADDR, &src_addr)){
      // Use src_addr string
    }
  }
}
```

If you implemented the extraction of some other fields please consider opening a Pull Request.

4.11 Low-level Configuration

Peafowl can be tuned by modifying some low-level configuration parameters, by modifying some `#define` in the `include/config.h` file before compiling and installing the library. The most important are the following:

- `PFWL_HTTP_MAX_HEADERS`: The maximum number of headers that can be extracted for a single HTTP packet [default = 256].
- `PFWL_DEFAULT_FLOW_TABLE_AVG_BUCKET_SIZE`: Default value for the average bucket size of the flow table.
- `PFWL_DEFAULT_EXPECTED_FLOWS`: Default value for the expected flows
- `PFWL_CACHE_LINE_SIZE`: Size of L1 cache line
- `PFWL_FLOW_TABLE_USE_MEMORY_POOL`: If 1 a certain amount of memory is preallocated for the hash table. That amount of memory can be specified using macros `` PFWL_FLOW_TABLE_MEMORY_POOL_DEFAULT_SIZE_v4`` and `PFWL_FLOW_TABLE_MEMORY_POOL_DEFAULT_SIZE_v6` respectively for IPv4 and IPv6 hash tables.
- `PFWL_USE_MTF`: If 1, when a packet is received, the information about its flow are moved on the top of the corresponding collision list. Experiments shown that this can be very useful in most cases.
- `PFWL_NUMA_AWARE`: Experimental macro for NUMA machine support
- `PFWL_NUMA_AWARE_FLOW_TABLE_NODE`: Experimental macro for NUMA machine support
- `PFWL_DEFAULT_MAX_TRIALS_PER_FLOW`: Maximum number of attempts before declaring the protocol of the flow as "Unknown". 0 means infinite.
- `PFWL_ENABLE_L3_TRUNCATION_PROTECTION` and `PFWL_ENABLE_L4_TRUNCATION_PROTECTION`: To protect from the cases in which the packet is truncated for some reasons
- `PFWL_FLOW_TABLE_HASH_VERSION`: Hash function used for the hash table where the flows are stored. Can be one of: PFWL_SIMPLE_HASH, PFWL_FNV_HASH, PFWL_MURMUR3_HASH, PFWL_BKDR_HASH. Experiments shown that PFWL_SIMPLE_HASH is a good choice for most cases.
- `PFWL_IPv4_FRAGMENTATION_DEFAULT_TABLE_SIZE`: Size of the table containing IPv4 fragments when IPv4 fragmentation is enabled.
- `PFWL_IPv4_FRAGMENTATION_DEFAULT_PER_HOST_MEMORY_LIMIT`: Maximum amount of memory that can be allocated to any source for fragmentation purposes.

- PFWL_IPv4_FRAGMENTATION_DEFAULT_TOTAL_MEMORY_LIMIT: Maximum amount of memory (global) that can be allocated for fragmentation purposes.
- PFWL_IPv4_FRAGMENTATION_DEFAULT_REASSEMBLY_TIMEOUT: Maximum amount of time (seconds) which can elapse from when the first fragment for a datagram is received to the moment when it is completely rebuilt. If after this amount of time there is still some missing fragment, the fragments saved by the framework will be removed.
- PFWL_IPv6_FRAGMENTATION_DEFAULT_TABLE_SIZE: As for IPv4
- PFWL_IPv6_FRAGMENTATION_DEFAULT_PER_HOST_MEMORY_LIMIT: As for IPv4
- PFWL_IPv6_FRAGMENTATION_DEFAULT_TOTAL_MEMORY_LIMIT: As for IPv4
- PFWL_IPv6_FRAGMENTATION_DEFAULT_REASSEMBLY_TIMEOUT: As for IPv4

PYTHON MODULE INDEX

p

INDEX

S

T

V