

PyDivert API Reference

class `pydivert.WinDivert`(`filter='true'`, `layer=<Layer.NETWORK: 0>`, `priority=0`, `flags=<Flag.DEFAULT: 0>`)

A WinDivert handle that can be used to capture packets. The main methods are `.open()`, `.recv()`, `.send()` and `.close()`.

Use it like so:

```
with pydivert.WinDivert() as w:
    for packet in w:
        print(packet)
        w.send(packet)
```

static `register()`

An utility method to register the service the first time. It is usually not required to call this function, as WinDivert will register itself when opening a handle.

static `is_registered()`

Check if the WinDivert service is currently installed on the system.

static `unregister()`

Unregisters the WinDivert service. This function only requests a service stop, which may not be processed immediately if there are still open handles.

static `check_filter`(`filter`, `layer=<Layer.NETWORK: 0>`)

Checks if the given packet filter string is valid with respect to the filter language.

The remapped function is WinDivertHelperCheckFilter:

```
BOOL WinDivertHelperCheckFilter(
    __in const char *filter,
    __in WINDIVERT_LAYER layer,
    __out_opt const char **errorStr,
    __out_opt UINT *errorPos
);
```

See: https://reqrypt.org/windivert-doc.html#divert_helper_check_filter

Returns: A tuple (res, pos, msg) with check result in 'res' human readable description of the error in 'msg' and the error's position in 'pos'.

`open()`

Opens a WinDivert handle for the given filter. Unless otherwise specified by flags, any packet that matches the filter will be diverted to the handle. Diverted packets can be read by the application with `receive()`.

The remapped function is WinDivertOpen:

```
HANDLE WinDivertOpen(
    __in const char *filter,
    __in WINDIVERT_LAYER layer,
    __in INT16 priority,
    __in UINT64 flags
);
```

For more info on the C call visit: http://reqrypt.org/windivert-doc.html#divert_open

`is_open`

Indicates if there is currently an open handle.

`close()`

Closes the handle opened by `open()`.

The remapped function is WinDivertClose:

```

BOOL WinDivertClose(
    __in HANDLE handle
);

```

For more info on the C call visit: http://reqrypt.org/windivert-doc.html#divert_close

recv(bufsize=1500)

Receives a diverted packet that matched the filter.

The remapped function is WinDivertRecv:

```

BOOL WinDivertRecv(
    __in HANDLE handle,
    __out PVOID pPacket,
    __in UINT packetLen,
    __out_opt PWINDIVERT_ADDRESS pAddr,
    __out_opt UINT *recvLen
);

```

For more info on the C call visit: http://reqrypt.org/windivert-doc.html#divert_recv

Returns: The return value is a *pydivert.Packet*.

send(packet, recalculate_checksum=True)

Injects a packet into the network stack. Recalculates the checksum before sending unless recalculate_checksum=False is passed.

The injected packet may be one received from recv(), or a modified version, or a completely new packet. Injected packets can be captured and diverted again by other WinDivert handles with lower priorities.

The remapped function is WinDivertSend:

```

BOOL WinDivertSend(
    __in HANDLE handle,
    __in PVOID pPacket,
    __in UINT packetLen,
    __in PWINDIVERT_ADDRESS pAddr,
    __out_opt UINT *sendLen
);

```

For more info on the C call visit: http://reqrypt.org/windivert-doc.html#divert_send

Returns: The return value is the number of bytes actually sent.

get_param(name)

Get a WinDivert parameter. See pydivert.Param for the list of parameters.

The remapped function is WinDivertGetParam:

```

BOOL WinDivertGetParam(
    __in HANDLE handle,
    __in WINDIVERT_PARAM param,
    __out UINT64 *pValue
);

```

For more info on the C call visit: http://reqrypt.org/windivert-doc.html#divert_get_param

Returns: The parameter value.

set_param(name, value)

Set a WinDivert parameter. See pydivert.Param for the list of parameters.

The remapped function is DivertSetParam:

```

BOOL WinDivertSetParam(
    __in HANDLE handle,
    __in WINDIVERT_PARAM param,
    __in UINT64 value
);

```

For more info on the C call visit: http://reqrypt.org/windivert-doc.html#divert_set_param

class pydivert.Packet(*raw, interface, direction*)

A single packet, possibly including an IP header, a TCP/UDP header and a payload. Creation of packets is cheap, parsing is done on first attribute access.

is_outbound

Indicates if the packet is outbound. Convenience method for `.direction`.

is_inbound

Indicates if the packet is inbound. Convenience method for `.direction`.

is_loopback

- True, if the packet is on the loopback interface.
- False, otherwise.

address_family

The packet address family:

- `socket.AF_INET`, if IPv4
- `socket.AF_INET6`, if IPv6
- `None`, otherwise.

protocol

- A (ipproto, proto_start) tuple.
`ipproto` is the IP protocol in use, e.g. `Protocol.TCP` or `Protocol.UDP`.
`proto_start` denotes the beginning of the protocol data.
 If the packet does not match our expectations, both `ipproto` and `proto_start` are `None`.

ipv4

- An `IPv4Header` instance, if the packet is valid IPv4.
- `None`, otherwise.

ipv6

- An `IPv6Header` instance, if the packet is valid IPv6.
- `None`, otherwise.

ip

- An `IPHeader` instance, if the packet is valid IPv4 or IPv6.
- `None`, otherwise.

icmpv4

- An `ICMPv4Header` instance, if the packet is valid ICMPv4.
- `None`, otherwise.

icmpv6

- An `ICMPv6Header` instance, if the packet is valid ICMPv6.
- `None`, otherwise.

icmp

- An `ICMPHeader` instance, if the packet is valid ICMPv4 or ICMPv6.
- `None`, otherwise.

tcp

- An TCPHeader instance, if the packet is valid TCP.
- None, otherwise.

udp

- An TCPHeader instance, if the packet is valid UDP.
- None, otherwise.

src_addr

- The source address, if the packet is valid IPv4 or IPv6.
- None, otherwise.

dst_addr

- The destination address, if the packet is valid IPv4 or IPv6.
- None, otherwise.

src_port

- The source port, if the packet is valid TCP or UDP.
- None, otherwise.

dst_port

- The destination port, if the packet is valid TCP or UDP.
- None, otherwise.

payload

- The payload, if the packet is valid TCP, UDP, ICMP or ICMPv6.
- None, otherwise.

recalculate_checksums(flags=0)

(Re)calculates the checksum for any IPv4/ICMP/ICMPv6/TCP/UDP checksum present in the given packet. Individual checksum calculations may be disabled via the appropriate flag. Typically this function should be invoked on a modified packet before it is injected with WinDivert.send(). Returns the number of checksums calculated.

See: https://reqrypt.org/windivert-doc.html#divert_helper_calc_checksums

wd_addr

Gets the interface and direction as a WINDIVERT_ADDRESS structure. :return: The WINDIVERT_ADDRESS structure.

matches(filter, layer=<Layer.NETWORK: 0>)

Evaluates the packet against the given packet filter string.

The remapped function is:

```
BOOL WinDivertHelperEvalFilter(
    __in const char *filter,
    __in WINDIVERT_LAYER layer,
    __in PVOID pPacket,
    __in UINT packetLen,
    __in PWINDIVERT_ADDRESS pAddr
);
```

See: https://reqrypt.org/windivert-doc.html#divert_helper_eval_filter

Parameters: • filter – The filter string.
• layer – The network layer.

Returns: True if the packet matches, and False otherwise.

class pydivert.packet.TCPHeader(packet, start=0)

ns

Indicates if the NS flag is set.

cwr

Indicates if the CWR flag is set.

ece

Indicates if the ECE flag is set.

urg

Indicates if the SYN flag is set.

ack

Indicates if the ACK flag is set.

psh

Indicates if the PSH flag is set.

rst

Indicates if the RST flag is set.

syn

Indicates if the SYN flag is set.

fin

Indicates if the FIN flag is set.

header_len

The TCP header length.

seq_num

The sequence number field.

ack_num

The acknowledgement number field.

window_size

The size of the receive window in bytes.

cksum

The TCP header checksum field.

urg_ptr

The Urgent Pointer field.

data_offset

The size of TCP header in 32bit words.

dst_port

The destination port.

payload

The packet payload data.

raw

The raw header, possibly including payload.

reserved

The reserved field.

src_port

The source port.

control_bits

The Control Bits field.

```
class pydivert.packet.UDPHeader(packet, start=0)
```

payload

The packet payload data.

dst_port

The destination port.

raw

The raw header, possibly including payload.

src_port

The source port.

cksum

The UDP header checksum field.

```
class pydivert.packet.IPv4Header(packet, start=0)
```

header_len

The IP header length in bytes.

hdr_len

The header length in words of 32bit.

packet_len

The total packet length, including *all* headers, as reported by the IP header.

tos

The Type Of Service field (six-bit DiffServ field and a two-bit ECN field).

ident

The Identification field.

reserved

Indicates if the RESERVED flag is set.

evil

Just an april's fool joke for the RESERVED flag.

df

Indicates if the DF flag is set.

mf

Indicates if the MF flag is set.

ttl

The Time To Live field.

protocol

The Protocol field.

cksum

The IP header Checksum field.

flags

The flags field: RESERVED (the evil bit), DF (don't fragment), MF (more fragments).

frag_offset

The Fragment Offset field in blocks of 8 bytes.

dscp

The Differentiated Services Code Point field (originally defined as Type of Service) also known as DiffServ.

diff_serv

The Differentiated Services Code Point field (originally defined as Type of Service) also known as DiffServ.

ecn

The Explicit Congestion Notification field.

dst_addr

The packet destination address.

raw

The raw header, possibly including payload.

src_addr

The packet source address.

```
class pydivert.packet.IPv6Header(packet, start=0)
```

dst_addr

The packet destination address.

payload_len

The Payload Length field.

raw

The raw header, possibly including payload.

src_addr

The packet source address.

next_hdr

The Next Header field. Replaces the Protocol field in IPv4.

hop_limit

The Hop Limit field. Replaces the TTL field in IPv4.

packet_len

The total packet length, including *all* headers, as reported by the IP header.

traffic_class

The Traffic Class field (six-bit DiffServ field and a two-bit ECN field).

flow_label

The Flow Label field.

diff_serv

The DiffServ field.

ecn

The Explicit Congestion Notification field.

```
class pydivert.packet.ICMPv4Header(packet, start=0)
```

cksum

The ICMP header checksum field.

code

The ICMP message code.

payload

The packet payload data.

raw

The raw header, possibly including payload.

type

The ICMP message type.

class `pydivert.packet.ICMPv6Header`(*packet*, *start*=0)

cksum

The ICMP header checksum field.

code

The ICMP message code.

payload

The packet payload data.

raw

The raw header, possibly including payload.

type

The ICMP message type.

class `pydivert.Layer`

See https://reqrypt.org/windivert-doc.html#divert_open

NETWORK = 0

NETWORK_FORWARD = 1

class `pydivert.Flag`

See https://reqrypt.org/windivert-doc.html#divert_open

DEFAULT = 0

SNIFF = 1

DROP = 2

NO_CHECKSUM = 1024

class `pydivert.Param`

See https://reqrypt.org/windivert-doc.html#divert_set_param

QUEUE_LEN = 0

QUEUE_TIME = 1

QUEUE_SIZE = 2

class `pydivert.CalcChecksumsOption`

See https://reqrypt.org/windivert-doc.html#divert_helper_calc_checksums

NO_IP_CHECKSUM = 1

NO_ICMP_CHECKSUM = 2

NO_ICMPV6_CHECKSUM = 4

NO_TCP_CHECKSUM = 8

NO_UDP_CHECKSUM = 16

NO_REPLACE = 2048

class `pydivert.Direction`

See https://reqrypt.org/windivert-doc.html#divert_address

OUTBOUND = 0**INBOUND = 1****class pydivert.Protocol**

Transport protocol values define the layout of the header that will immediately follow the IPv4 or IPv6 header. See <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>

HOPOPT = 0**ICMP = 1****TCP = 6****UDP = 17****ROUTING = 43****FRAGMENT = 44****AH = 51****ICMPV6 = 58****NONE = 59****DSTOPTS = 60**