**WinDivert 2.2: Windows Packet Divert**

**Table of Contents**

* [1. Introduction](https://reqrypt.org/windivert-doc.html" \l "introduction)
* [2. Building](https://reqrypt.org/windivert-doc.html#building)
  + [2.1 Driver Signing](https://reqrypt.org/windivert-doc.html#driver_signing)
* [3. Installing](https://reqrypt.org/windivert-doc.html#installing)
* [4. Uninstalling](https://reqrypt.org/windivert-doc.html#uninstalling)
* [5. Programming API](https://reqrypt.org/windivert-doc.html#programming_api)
  + [5.1 WINDIVERT\_LAYER](https://reqrypt.org/windivert-doc.html#divert_layers)
  + [5.2 WINDIVERT\_EVENT](https://reqrypt.org/windivert-doc.html#divert_events)
  + [5.3 WINDIVERT\_ADDRESS](https://reqrypt.org/windivert-doc.html#divert_address)
  + [5.4 WinDivertOpen](https://reqrypt.org/windivert-doc.html#divert_open)
  + [5.5 WinDivertRecv](https://reqrypt.org/windivert-doc.html#divert_recv)
  + [5.6 WinDivertRecvEx](https://reqrypt.org/windivert-doc.html#divert_recv_ex)
  + [5.7 WinDivertSend](https://reqrypt.org/windivert-doc.html#divert_send)
  + [5.8 WinDivertSendEx](https://reqrypt.org/windivert-doc.html#divert_send_ex)
  + [5.9 WinDivertShutdown](https://reqrypt.org/windivert-doc.html#divert_shutdown)
  + [5.19 WinDivertClose](https://reqrypt.org/windivert-doc.html#divert_close)
  + [5.11 WinDivertSetParam](https://reqrypt.org/windivert-doc.html#divert_set_param)
  + [5.12 WinDivertGetParam](https://reqrypt.org/windivert-doc.html#divert_get_param)
* [6. Helper Programming API](https://reqrypt.org/windivert-doc.html#helper_programming_api)
  + [6.1 WINDIVERT\_IPHDR](https://reqrypt.org/windivert-doc.html#divert_iphdr)
  + [6.2 WINDIVERT\_IPV6HDR](https://reqrypt.org/windivert-doc.html#divert_ipv6hdr)
  + [6.3 WINDIVERT\_ICMPHDR](https://reqrypt.org/windivert-doc.html#divert_icmphdr)
  + [6.4 WINDIVERT\_ICMPV6HDR](https://reqrypt.org/windivert-doc.html#divert_icmpv6hdr)
  + [6.5 WINDIVERT\_TCPHDR](https://reqrypt.org/windivert-doc.html#divert_tcphdr)
  + [6.6 WINDIVERT\_UDPHDR](https://reqrypt.org/windivert-doc.html#divert_udphdr)
  + [6.7 WinDivertHelperParsePacket](https://reqrypt.org/windivert-doc.html#divert_helper_parse_packet)
  + [6.8 WinDivertHelperHashPacket](https://reqrypt.org/windivert-doc.html#divert_helper_hash_packet)
  + [6.9 WinDivertHelperParseIPv4Address](https://reqrypt.org/windivert-doc.html#divert_helper_parse_ipv4_address)
  + [6.10 WinDivertHelperParseIPv6Address](https://reqrypt.org/windivert-doc.html#divert_helper_parse_ipv6_address)
  + [6.11 WinDivertHelperFormatIPv4Address](https://reqrypt.org/windivert-doc.html#divert_helper_format_ipv4_address)
  + [6.12 WinDivertHelperFormatIPv6Address](https://reqrypt.org/windivert-doc.html#divert_helper_format_ipv6_address)
  + [6.13 WinDivertHelperCalcChecksums](https://reqrypt.org/windivert-doc.html#divert_helper_calc_checksums)
  + [6.14 WinDivertHelperDecrementTTL](https://reqrypt.org/windivert-doc.html#divert_helper_dec_ttl)
  + [6.15 WinDivertHelperCompileFilter](https://reqrypt.org/windivert-doc.html#divert_helper_compile_filter)
  + [6.16 WinDivertHelperEvalFilter](https://reqrypt.org/windivert-doc.html#divert_helper_eval_filter)
  + [6.17 WinDivertHelperFormatFilter](https://reqrypt.org/windivert-doc.html#divert_helper_format_filter)
  + [6.18 WinDivertHelperNtoh\*](https://reqrypt.org/windivert-doc.html#divert_helper_ntoh)
  + [6.19 WinDivertHelperHton\*](https://reqrypt.org/windivert-doc.html#divert_helper_hton)
* [7. Filter Language](https://reqrypt.org/windivert-doc.html#filter_language)
  + [7.1 Filter Examples](https://reqrypt.org/windivert-doc.html#filter_examples)
  + [7.2 Filter Usage](https://reqrypt.org/windivert-doc.html#filter_usage)
* [8. Performance](https://reqrypt.org/windivert-doc.html#performance)
* [9. Samples](https://reqrypt.org/windivert-doc.html#samples)
* [10. Known Issues](https://reqrypt.org/windivert-doc.html#known_issues)
* [11. License](https://reqrypt.org/windivert-doc.html#license)

**1. Introduction**

WinDivert is a powerful user-mode capture/sniffing/modification/blocking/re-injection package for Windows 7, Windows 8 and Windows 10. WinDivert can be used to implement user-mode packet filters, packet sniffers, firewalls, NAT, VPNs, tunneling applications, etc., without the need to write kernel-mode code.

The main features of the WinDivert are:

* User-mode packet capture, sniffing, dropping, filtering, modification, re-injection, etc.
* Simple, high-level, programming API.
* Fully documented with sample programs.
* Full IPv6 support.
* Full loopback (localhost) support.
* A modern WDF/WFP driver implementation.
* Open source; Licensed under GNU Lesser General Public License (LGPL) version 3. See the [License](https://reqrypt.org/windivert-doc.html#license) for more information.

WinDivert provides similar functionality to divert sockets from FreeBSD/MacOS, NETLINK sockets from Linux.

**2. Building**

Note that pre-built WinDivert binary distributions are available from the [WinDivert website](https://reqrypt.org/windivert.html). Most users do not need to build their own version of WinDivert from source.

The source code for WinDivert is available for download at:

<https://github.com/basil00/Divert>

To build the WinDivert drivers from source:

1. Download and install [Windows Driver Kit 7.1.0](https://www.microsoft.com/whdc/devtools/wdk/default.mspx).
2. Open a *x86 Free Build Environment* console.
3. In the WinDivert package root directory, run the command:
4. wddk-build.bat

This will build the install\WDDK\i386\WinDivert32.sys driver.

1. Next, open a *x64 Free Build Environment* console.
2. Re-run the wddk-build.bat command to build the install\WDDK\amd64\WinDivert64.sys driver.

To build the WinDivert user-mode library (WinDivert.dll) and sample programs:

1. First, build the WinDivert drivers by running the wddk-build.bat command described above.
2. In Linux (with the MinGW cross-compilers installed) and in the WinDivert package root directory, run the command:
3. sh mingw-build.sh

This will the user-mode library and sample programs which will be placed in the install\MINGW subdirectory.

The generated WinDivert.dll/WinDivert.lib files should be compatible with all major compilers, including both MinGW and Visual Studio.

**2.1 Driver Signing**

If you built your own WinDivert32.sys/WinDivert64.sys drivers, they must be digitally signed before they can be used. See [Driver Signing Requirements for Windows](https://msdn.microsoft.com/en-us/windows/hardware/gg487317.aspx) for more information.

Note that the pre-built WinDivert32.sys/WinDivert64.sys drivers from the official WinDivert distribution are already digitally signed.

**3. Installing**

WinDivert does not require any special installation. Depending on your target configuration, simply place the following files in your application's home directory:

|  |  |  |
| --- | --- | --- |
| **Application Type** | **Target Windows Type** | **Files Required** |
| 32-bit | 32-bit Windows only | WinDivert.dll (32-bit version) and WinDivert32.sys |
| 64-bit | 64-bit Windows only | WinDivert.dll (64-bit version) and WinDivert64.sys |
| 32-bit | Both 32-bit and 64-bit Windows | WinDivert.dll (32-bit version), WinDivert32.sys, and WinDivert64.sys |

The WinDivert driver is automatically (and silently) installed on demand whenever your application calls [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open). The calling application must have Administrator privileges.

**4. Uninstalling**

To uninstall, simply delete the WinDivert.dll, WinDivert32.sys, and WinDivert64.sys files. If already running, the WinDivert driver will be automatically uninstalled during the next machine reboot. The WinDivert driver can also be manually removed by (1) terminating all processes that are using WinDivert, and (2) issuing the following commands at the command prompt

sc stop WinDivert

sc delete WinDivert

Alternatively, the WinDivert driver can be removed by using the windivertctl.exe [sample program](https://reqrypt.org/windivert-doc.html#samples) by issuing the following command:

windivertctl uninstall

**5. Programming API**

To use the WinDivert package, a program/application must:

1. Include the windivert.h header file
2. #include "windivert.h"
3. Link against or dynamically load the WinDivert.dll dynamic link library.

**5.1 WINDIVERT\_LAYER**

|  |
| --- |
| typedef enum  {  WINDIVERT\_LAYER\_NETWORK = 0,  WINDIVERT\_LAYER\_NETWORK\_FORWARD,  WINDIVERT\_LAYER\_FLOW,  WINDIVERT\_LAYER\_SOCKET,  WINDIVERT\_LAYER\_REFLECT,  } **WINDIVERT\_LAYER**, \***PWINDIVERT\_LAYER**; |

**Remarks**

WinDivert supports several *layers* for diverting or capturing network packets/events. Each layer has its own capabilities, such as the ability to block events or to inject new events, etc. The list of supported WinDivert layers is summarized below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer** | **Capability** | | | | **Description** |
|  | **Block?** | **Inject?** | **Data?** | **PID?** |  |
| WINDIVERT\_LAYER\_NETWORK | ✔ | ✔ | ✔ |  | Network packets to/from the local machine. |
| WINDIVERT\_LAYER\_NETWORK\_FORWARD | ✔ | ✔ | ✔ |  | Network packets passing through the local machine. |
| WINDIVERT\_LAYER\_FLOW |  |  |  | ✔ | Network flow established/deleted events. |
| WINDIVERT\_LAYER\_SOCKET | ✔ |  |  | ✔ | Socket operation events. |
| WINDIVERT\_LAYER\_REFLECT |  |  | ✔ | ✔ | WinDivert handle events. |

Here, the layer capabilities are:

* (Block?) the layer can block events/packets;
* (Inject?) the layer can inject new events/packets;
* (Data?) whether the layer returns packets/data or not; and
* (PID?) whether the ID for the process associated with an event/packet is available at this layer, or not.

The WINDIVERT\_LAYER\_NETWORK and WINDIVERT\_LAYER\_NETWORK\_FORWARD layers allow the user application to capture/block/inject network packets passing to/from (and through) the local machine. Due to technical limitations, process ID information is not available at these layers.

The WINDIVERT\_LAYER\_FLOW layer captures information about network flow establishment/deletion events. Here, a *flow* represents either (1) a TCP connection, or (2) an implicit flow created by the first sent/received packet for non-TCP traffic, e.g., UDP. Old flows are deleted when the corresponding connection is closed (for TCP), or based on an activity timeout (non-TCP). Flow-related events can be captured, but not blocked nor injected. Process ID information is also available at this layer. Due to technical limitations, the WINDIVERT\_LAYER\_FLOW layer cannot capture flow events that occurred before the handle was opened.

The WINDIVERT\_LAYER\_SOCKET layer can capture or block events corresponding to socket operations, such as bind(), connect(), listen(), etc., or the termination of socket operations, such as a TCP socket disconnection. Unlike the flow layer, most socket-related events can be blocked. However, it is not possible to inject new or modified socket events. Process ID information (of the process responsible for the socket operation) is available at this layer. Due to technical limitations, this layer cannot capture events that occurred before the handle was opened.

Finally, the WINDIVERT\_LAYER\_REFLECT layer can capture events relating to WinDivert itself, such as when another process opens a new WinDivert handle, or closes an old WinDivert handle. WinDivert events can be captured but not injected nor blocked. Process ID information (of the process responsible for opening the WinDivert handle) is available at this layer. This layer also returns data in the form of an object representation of the filter string used to open the handle. The object representation can be converted back into a human-readable filter string using the [WinDivertHelperFormatFilter()](https://reqrypt.org/windivert-doc.html#divert_helper_format_filter) function. This layer can also capture events that occurred before the handle was opened. This layer cannot capture events related to other WINDIVERT\_LAYER\_REFLECT-layer handles.

**5.2 WINDIVERT\_EVENT**

|  |
| --- |
| typedef enum  {  WINDIVERT\_EVENT\_NETWORK\_PACKET,  WINDIVERT\_EVENT\_FLOW\_ESTABLISHED,  WINDIVERT\_EVENT\_FLOW\_DELETED,  WINDIVERT\_EVENT\_SOCKET\_BIND,  WINDIVERT\_EVENT\_SOCKET\_CONNECT,  WINDIVERT\_EVENT\_SOCKET\_LISTEN,  WINDIVERT\_EVENT\_SOCKET\_ACCEPT,  WINDIVERT\_EVENT\_SOCKET\_CLOSE,  WINDIVERT\_EVENT\_REFLECT\_OPEN,  WINDIVERT\_EVENT\_REFLECT\_CLOSE,  } **WINDIVERT\_EVENT**, \***PWINDIVERT\_EVENT**; |

**Remarks**

Each layer supports one or more *events* summarized below:

* **WINDIVERT\_LAYER\_NETWORK** and **WINDIVERT\_LAYER\_NETWORK\_FORWARD**: Only a single event is supported:

|  |  |
| --- | --- |
| **Event** | **Description** |
| WINDIVERT\_EVENT\_NETWORK\_PACKET | A new network packet. |

* **WINDIVERT\_LAYER\_FLOW**: Two events are supported:

|  |  |
| --- | --- |
| **Event** | **Description** |
| WINDIVERT\_EVENT\_FLOW\_ESTABLISHED | A new flow is created. |
| WINDIVERT\_EVENT\_FLOW\_DELETED | An old flow is deleted. |

* **WINDIVERT\_LAYER\_SOCKET**: The following events are supported:

|  |  |
| --- | --- |
| **Event** | **Description** |
| WINDIVERT\_EVENT\_SOCKET\_BIND | A bind() operation. |
| WINDIVERT\_EVENT\_SOCKET\_CONNECT | A connect() operation. |
| WINDIVERT\_EVENT\_SOCKET\_LISTEN | A listen() operation. |
| WINDIVERT\_EVENT\_SOCKET\_ACCEPT | An accept() operation. |
| WINDIVERT\_EVENT\_SOCKET\_CLOSE | A socket endpoint is closed. This corresponds to a previous binding being released, or an established connection being terminated. The event cannot be blocked. |

* **WINDIVERT\_LAYER\_REFLECT**: Two events are supported:

|  |  |
| --- | --- |
| **Event** | **Description** |
| WINDIVERT\_EVENT\_REFLECT\_OPEN | A new WinDivert handle was opened. |
| WINDIVERT\_EVENT\_REFLECT\_CLOSE | An old WinDivert handle was closed. |

**5.3 WINDIVERT\_ADDRESS**

|  |
| --- |
| typedef struct  {  UINT32 IfIdx;  UINT32 SubIfIdx;  } **WINDIVERT\_DATA\_NETWORK**, \***PWINDIVERT\_DATA\_NETWORK**;  typedef struct  {  UINT64 Endpoint;  UINT64 ParentEndpoint;  UINT32 ProcessId;  UINT32 LocalAddr[4];  UINT32 RemoteAddr[4];  UINT16 LocalPort;  UINT16 RemotePort;  UINT8 Protocol;  } **WINDIVERT\_DATA\_FLOW**, \***PWINDIVERT\_DATA\_FLOW**;  typedef struct  {  UINT64 Endpoint;  UINT64 ParentEndpoint;  UINT32 ProcessId;  UINT32 LocalAddr[4];  UINT32 RemoteAddr[4];  UINT16 LocalPort;  UINT16 RemotePort;  UINT8 Protocol;  } **WINDIVERT\_DATA\_SOCKET**, \***PWINDIVERT\_DATA\_SOCKET**;  typedef struct  {  INT64 Timestamp;  UINT32 ProcessId;  WINDIVERT\_LAYER Layer;  UINT64 Flags;  INT16 Priority;  } **WINDIVERT\_DATA\_REFLECT**, \***PWINDIVERT\_DATA\_REFLECT**;  typedef struct  {  INT64 Timestamp;  UINT64 Layer:8;  UINT64 Event:8;  UINT64 Sniffed:1;  UINT64 Outbound:1;  UINT64 Loopback:1;  UINT64 Impostor:1;  UINT64 IPv6:1;  UINT64 IPChecksum:1;  UINT64 TCPChecksum:1;  UINT64 UDPChecksum:1;  union  {  WINDIVERT\_DATA\_NETWORK Network;  WINDIVERT\_DATA\_FLOW Flow;  WINDIVERT\_DATA\_SOCKET Socket;  WINDIVERT\_DATA\_REFLECT Reflect;  };  } **WINDIVERT\_ADDRESS**, \***PWINDIVERT\_ADDRESS**; |

**Fields**

* Timestamp: A timestamp indicating when event occurred.
* Layer: The handle's layer (WINDIVERT\_LAYER\_\*).
* Event: The captured event (WINDIVERT\_EVENT\_\*).
* Sniffed: Set to 1 if the event was sniffed (i.e., not blocked), 0 otherwise..
* Outbound: Set to 1 for *outbound* packets/event, 0 for *inbound* or otherwise.
* Loopback: Set to 1 for loopback packets, 0 otherwise
* Impostor: Set to 1 for impostor packets, 0 otherwise.
* IPv6: Set to 1 for IPv6 packets/events, 0 otherwise
* IPChecksum: Set to 1 if the IPv4 checksum is valid, 0 otherwise.
* TCPChecksum: Set to 1 if the TCP checksum is valid, 0 otherwise.
* UDPChecksum: Set to 1 if the UDP checksum is valid, 0 otherwise.
* Network.IfIdx: The interface index on which the packet arrived (for inbound packets), or is to be sent (for outbound packets).
* Network.SubIfIdx: The sub-interface index for IfIdx.
* Flow.EndpointId: The endpoint ID of the flow.
* Flow.ParentEndpointId: The parent endpoint ID of the flow.
* Flow.ProcessId: The ID of the process associated with the flow.
* Flow.LocalAddr, Flow.RemoteAddr, Flow.LocalPort, Flow.RemotePort, and Flow.Protocol: The network 5-tuple associated with the flow.
* Socket.EndpointId: The endpoint ID of the socket operation.
* Socket.ParentEndpointId: The parent endpoint ID of the socket operation.
* Socket.ProcessId: The ID of the process associated with the socket operation.
* Socket.LocalAddr, Socket.RemoteAddr, Socket.LocalPort, Socket.RemotePort, and Socket.Protocol: The network 5-tuple associated with the socket operation.
* Reflect.Timestamp: A timestamp indicating when the handle was opened.
* Reflect.ProcessId: The ID of the process that opened the handle.
* Reflect.Layer, Reflect.Flags, and Reflect.Priority: The [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open) parameters of the opened handle.

**Remarks**  
The [WINDIVERT\_ADDRESS](https://reqrypt.org/windivert-doc.html#divert_address) structure represents the "address" of a captured or injected packet. The address includes the packet's timestamp, layer, event, flags, and layer-specific data. All fields are set by [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) when the packet/event is captured. Only some fields are used by [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send) when a packet is injected.

The Timestamp indicates when the packet/event was first captured by WinDivert. It uses the same clock as [QueryPerformanceCounter()](https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx).

The Layer indicates the *layer* parameter ([WINDIVERT\_LAYER\_\*](https://reqrypt.org/windivert-doc.html#divert_layers)) that was passed to [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open). It is included in the address to make the structure self-contained.

The Event indicates the layer-specific *event* ([WINDIVERT\_EVENT\_\*](https://reqrypt.org/windivert-doc.html#divert_events)) that was captured.

The Outbound flag is set for *outbound* packets/events, and is cleared for *inbound* or direction-less packets/events.

The Loopback flag is set for *loopback* packets. Note that Windows considers any packet originating from, and destined to, the current machine to be a loopback packet, so loopback packets are not limited to localhost addresses. Note that WinDivert considers loopback packets to be *outbound only*, and will not capture loopback packets on the inbound path.

The Impostor flag is set for *impostor* packets. An impostor packet is any packet injected by another driver rather than originating from the network or Windows TCP/IP stack. Impostor packets are problematic since they can cause infinite loops, where a packet injected by [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send) is captured again by [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv). For more information, see [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send).

The IPv6 flag is set for *IPv6* packets/events, and cleared for *IPv4* packets/events.

The \*Checksum flags indicate whether the packet has valid checksums or not. When *IP/TCP/UDP checksum offloading* is enabled, it is possible that captured packets do not have valid checksums. Invalid checksums may be arbitrary values.

The Network.\* fields are only valid at the WINDIVERT\_LAYER\_NETWORK and WINDIVERT\_LAYER\_NETWORK\_FORWARD layers. The Network.IfIdx/Network.SubIfIdx indicate the packet's network adapter (a.k.a. interface) index. These values are ignored for *outbound* packets.

The Flow.\* fields are only valid at the WINDIVERT\_LAYER\_FLOW layer. The Flow.ProcessId is the *ID* of the process that created the flow (for outbound), or receives the flow (for inbound). The (Flow.LocalAddr, Flow.LocalPort, Flow.RemoteAddr, Flow.RemotePort, Flow.Protocol) fields form the network 5-tuple associated with the flow. For IPv4, the Flow.LocalAddr and Flow.RemoteAddr fields will be IPv4-mapped IPv6 addresses, e.g. the IPv4 address X.Y.Z.W will be represented by ::ffff:X.Y.Z.W.

The Socket.\* fields are only valid at the WINDIVERT\_LAYER\_SOCKET layer. The Socket.ProcessId is the *ID* of the process that executed the socket operation. The (Socket.LocalAddr, Socket.LocalPort, Socket.RemoteAddr, Socket.RemotePort, Socket.Protocol) fields form the network 5-tuple associated with the operation. For IPv4, the Socket.LocalAddr and Socket.RemoteAddr fields will be IPv4-mapped IPv6 addresses. The WINDIVERT\_EVENT\_SOCKET\_BIND and WINDIVERT\_EVENT\_SOCKET\_LISTEN events can occur before a connection attempt has been made, meaning that the Socket.RemoteAddr and Socket.RemotePort fields for these events will be zero.

The Reflect.\* fields are only valid at the WINDIVERT\_LAYER\_REFLECT layer. The Reflect.ProcessId is the *ID* of the process that opened the WinDivert handle. The Reflect.Timestamp field is a timestamp indicating when the handle was opened, using the same clock as [QueryPerformanceCounter()](https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx). The Reflect.Layer, Reflect.Flags, and Reflect.Priority fields correspond to the [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open) parameters of the opened handle.

Most address fields are ignored by [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send). The exceptions are Outbound (for WINDIVERT\_LAYER\_NETWORK only), Impostor, IPChecksum, TCPChecksum, UDPChecksum, Network.IfIdx and Network.SubIfIdx.

**5.4 WinDivertOpen**

|  |
| --- |
| HANDLE **WinDivertOpen**(  \_\_in const char \*filter,  \_\_in WINDIVERT\_LAYER layer,  \_\_in INT16 priority,  \_\_in UINT64 flags  ); |

**Parameters**

* filter: A packet filter string specified in the WinDivert [filter language](https://reqrypt.org/windivert-doc.html#filter_language).
* layer: The layer.
* priority: The priority of the handle.
* flags: Additional flags.

**Return Value**  
A valid WinDivert handle on success, or INVALID\_HANDLE\_VALUE if an error occurred. Use GetLastError() to get the reason for the error. Common errors include:

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** | **Description** |
| ERROR\_FILE\_NOT\_FOUND | 2 | The driver files WinDivert32.sys or WinDivert64.sys were not found. |
| ERROR\_ACCESS\_DENIED | 5 | The calling application does not have Administrator privileges. |
| ERROR\_INVALID\_PARAMETER | 87 | This indicates an invalid packet filter string, layer, priority, or flags. |
| ERROR\_INVALID\_IMAGE\_HASH | 577 | The WinDivert32.sys or WinDivert64.sys driver does not have a valid digital signature (see the [driver signing requirements](https://reqrypt.org/windivert-doc.html#driver_signing) above). |
| ERROR\_DRIVER\_FAILED\_PRIOR\_UNLOAD | 654 | An incompatible version of the WinDivert driver is currently loaded. |
| ERROR\_SERVICE\_DOES\_NOT\_EXIST | 1060 | The handle was opened with the WINDIVERT\_FLAG\_NO\_INSTALL flag and the WinDivert driver is not already installed. |
| ERROR\_DRIVER\_BLOCKED | 1275 | This error occurs for various reasons, including:   1. the WinDivert driver is blocked by security software; or 2. you are using a virtualization environment that does not support drivers. |
| EPT\_S\_NOT\_REGISTERED | 1753 | This error occurs when the *Base Filtering Engine* service has been disabled. |

**Remarks**  
Opens a WinDivert handle for the given filter. Unless otherwise specified by flags, any packet or event that matches the filter will be diverted to the handle. Diverted packets/events can be read by the application with [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv).

A typical application is only interested in a subset of all network traffic or events. In this case the filter should *match as closely as possible* to the subset of interest. This avoids unnecessary overheads introduced by diverting packets to the user-mode application. See the [filter language](https://reqrypt.org/windivert-doc.html#filter_language) section for more information.

The *layer* of the WinDivert handle is determined by the layer parameter. See [WINDIVERT\_LAYER](https://reqrypt.org/windivert-doc.html#divert_layers) for more information. Currently the following layers are supported:

|  |  |
| --- | --- |
| **Layer** | **Description** |
| WINDIVERT\_LAYER\_NETWORK = 0 | Network packets to/from the local machine. This is the default layer. |
| WINDIVERT\_LAYER\_NETWORK\_FORWARD | Network packets passing through the local machine. |
| WINDIVERT\_LAYER\_FLOW | Network flow established/deleted events. |
| WINDIVERT\_LAYER\_SOCKET | Socket operation events. |
| WINDIVERT\_LAYER\_REFLECT | WinDivert handle events. |

Different WinDivert handles can be assigned different priorities by the priority parameter. Packets are diverted to higher priority handles before lower priority handles. Packets injected by a handle are then diverted to the next priority handle, and so on, provided the packet matches the handle's filter. A packet is only diverted once per priority level, so handles should not share priority levels unless they use mutually exclusive filters. Otherwise it is not defined which handle will receive the packet first. Higher priority values represent higher priorities, with WINDIVERT\_PRIORITY\_HIGHEST being the highest priority, 0 the middle (and a good default) priority, and WINDIVERT\_PRIORITY\_LOWEST the lowest priority.

Different flags affect how the opened handle behaves. The following flags are supported:

|  |  |
| --- | --- |
| **Flag** | **Description** |
| WINDIVERT\_FLAG\_SNIFF | This flag opens the WinDivert handle in *packet sniffing* mode. In packet sniffing mode the original packet is not dropped-and-diverted (the default) but copied-and-diverted. This mode is useful for implementing packet sniffing tools similar to those applications that currently use Winpcap. |
| WINDIVERT\_FLAG\_DROP | This flag indicates that the user application does not intend to read matching packets with [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv), instead the packets should be silently dropped. This is useful for implementing simple packet filters using the WinDivert [filter language](https://reqrypt.org/windivert-doc.html#filter_language). |
| WINDIVERT\_FLAG\_RECV\_ONLY | This flags forces the handle into receive only mode which effectively disables [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send). This means that it is possible to block/capture packets or events but not inject them. |
| WINDIVERT\_FLAG\_READ\_ONLY | An alias for WINDIVERT\_FLAG\_RECV\_ONLY. |
| WINDIVERT\_FLAG\_SEND\_ONLY | This flags forces the handle into send only mode which effectively disables [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv). This means that it is possible to inject packets or events, but not block/capture them. |
| WINDIVERT\_FLAG\_WRITE\_ONLY | An alias for WINDIVERT\_FLAG\_SEND\_ONLY. |
| WINDIVERT\_FLAG\_NO\_INSTALL | This flags causes WinDivertOpen() to fail with ERROR\_SERVICE\_DOES\_NOT\_EXIST if the WinDivert driver is not already installed. This flag is useful for querying the WinDivert state using a WINDIVERT\_LAYER\_REFLECT handle. |
| WINDIVERT\_FLAG\_FRAGMENTS | If set, the handle will capture inbound IP fragments, but not inbound reassembled IP packets. Otherwise, if not set (the default), the handle will capture inbound reassembled IP packets, but not inbound IP fragments. This flag only affects inbound packets at the WINDIVERT\_LAYER\_NETWORK layer, else the flag is ignored. |

Note that any combination of (WINDIVERT\_FLAG\_SNIFF | WINDIVERT\_FLAG\_DROP) or (WINDIVERT\_FLAG\_RECV\_ONLY | WINDIVERT\_FLAG\_SEND\_ONLY) are considered invalid.

Some layers have mandatory flags, as listed below:

|  |  |
| --- | --- |
| **Layer** | **Required Flags** |
| WINDIVERT\_LAYER\_FLOW | WINDIVERT\_FLAG\_SNIFF | WINDIVERT\_FLAG\_RECV\_ONLY |
| WINDIVERT\_LAYER\_SOCKET | WINDIVERT\_FLAG\_RECV\_ONLY |
| WINDIVERT\_LAYER\_REFLECT | WINDIVERT\_FLAG\_SNIFF | WINDIVERT\_FLAG\_RECV\_ONLY |

**5.5 WinDivertRecv**

|  |
| --- |
| BOOL **WinDivertRecv**(  \_\_in HANDLE handle,  \_\_out\_opt PVOID pPacket,  \_\_in UINT packetLen,  \_\_out\_opt UINT \*pRecvLen,  \_\_out\_opt WINDIVERT\_ADDRESS \*pAddr  ); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* pPacket: An optional buffer for the captured packet.
* packetLen: The length of the pPacket buffer.
* pRecvLen: The total number of bytes written to pPacket. Can be NULL if this information is not required.
* pAddr: An optional buffer for the [address](https://reqrypt.org/windivert-doc.html#divert_address) of the captured packet/event.

**Return Value**  
TRUE if a packet/event was successfully received, or FALSE if an error occurred. Use GetLastError() to get the reason for the error.

Common errors include:

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** | **Description** |
| ERROR\_INSUFFICIENT\_BUFFER | 122 | The captured packet is larger than the pPacket buffer. |
| ERROR\_NO\_DATA | 232 | The handle has been shutdown using [WinDivertShutdown()](https://reqrypt.org/windivert-doc.html#divert_shutdown) and the packet queue is empty. |

**Remarks**  
Receives a single captured packet/event matching the filter passed to [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open). The received packet/event is guaranteed to match the filter.

Only some [layers](https://reqrypt.org/windivert-doc.html#divert_layers) can capture packets/data, as summarized below:

|  |  |  |
| --- | --- | --- |
| **Layer** | **Data?** | **Description** |
| WINDIVERT\_LAYER\_NETWORK | ✔ | Network packet. |
| WINDIVERT\_LAYER\_NETWORK\_FORWARD | ✔ | Network packet. |
| WINDIVERT\_LAYER\_FLOW |  | - |
| WINDIVERT\_LAYER\_SOCKET |  | - |
| WINDIVERT\_LAYER\_REFLECT | ✔ | Filter object. |

For layers that do support capturing, the captured packet/data will be written to the pPacket buffer. If non-NULL, then the total number of bytes written to pPacket will be written to pRecvLen. If the pPacket buffer is too small, the packet will be truncated and the operation will fail with the ERROR\_INSUFFICIENT\_BUFFER error code. This error can be ignored if the application only intends to receive part of the packet, e.g., the IP headers only. For layers that do not capture packets/data, the pPacket parameter should be NULL and packetLen should be zero.

If non-NULL, the [address](https://reqrypt.org/windivert-doc.html#divert_address) of the packet/event will be written to the pAddr buffer.

An application should call [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) *as soon as possible* after a successful call to [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open). When a WinDivert handle is open, any packet/event that matches the filter will be captured and queued until handled by [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv). Packets/events are not queued indefinitely, and if not handled in a timely manner, data may be lost. The amount of time a packet/event is queued can be controlled using the [WinDivertSetParam()](https://reqrypt.org/windivert-doc.html#divert_set_param) function.

Captured packets are guaranteed to have correct checksums or have the corresponding \*Checksum flag unset (see [WINDIVERT\_ADDRESS](https://reqrypt.org/windivert-doc.html#divert_address)).

[WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) should not be used on any WinDivert handle created with the WINDIVERT\_FLAG\_DROP set.

**5.6 WinDivertRecvEx**

|  |
| --- |
| BOOL **WinDivertRecvEx**(  \_\_in HANDLE handle,  \_\_out VOID \*pPacket,  \_\_in UINT packetLen,  \_\_out\_opt UINT \*pRecvLen,  \_\_in UINT64 flags,  \_\_out\_opt WINDIVERT\_ADDRESS \*pAddr,  \_\_inout\_opt UINT \*pAddrLen,  \_\_inout\_opt LPOVERLAPPED lpOverlapped  ); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* pPacket: A buffer for the captured packet(s).
* packetLen: The length of the pPacket buffer in bytes.
* pRecvLen: The total number of bytes written to pPacket. Can be NULL if this information is not required.
* flags: Reserved, set to zero.
* pAddr: The [WINDIVERT\_ADDRESS](https://reqrypt.org/windivert-doc.html#divert_address) of the captured packet(s).
* pAddrLen: Initially, a pointer to the length of the pAddr buffer in bytes. This value is updated to the total bytes written to pAddr. If NULL, a fixed length of sizeof(WINDIVERT\_ADDRESS) is assumed.
* lpOverlapped: An optional pointer to a OVERLAPPED structure.

**Return Value**  
TRUE if a packet was successfully received, or FALSE otherwise. Use GetLastError() to get the reason. The error code ERROR\_IO\_PENDING indicates that the overlapped operation has been successfully initiated and that completion will be indicated at a later time. All other codes indicate an error.

**Remarks**  
This function is equivalent to [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) except:

* *Overlapped I/O* is supported via the lpOverlapped parameter.
* *Batched I/O* (i.e., receiving multiple packets at once) is supported.

Batched I/O makes it possible to receive up to WINDIVERT\_BATCH\_MAX packets at once using a single operation, reducing the number of kernel/user-mode context switches and improving performance. To enable batched I/O:

1. pass an array of more than one WINDIVERT\_ADDRESS to pAddr;
2. set pAddrLen to be the total size (in bytes) of the pAddr buffer; and
3. ensure that pPacket points to a sufficiently large buffer capable of receiving multiple packets.

For example:

UINT8 packets[10 \* MTU]; // Space for up to 10 packets

WINDIVERT\_ADDRESS addr[10]; // Addresses for up to 10 packets

UINT addr\_len = sizeof(addr);

BOOL result = WinDivertRecvEx(handle, packets, ..., addr, &addr\_len, ...);

upon successful completion, the value pointed to by pAddrLen is updated to the total number of address bytes actually received. For example, if a total of 5 packets were received, then the value pointed to by pAddrLen will be set to (5\*sizeof(WINDIVERT\_ADDRESS)). The received packets are packed contiguously (i.e., no gaps) into the pPacket buffer.

**5.7 WinDivertSend**

|  |
| --- |
| BOOL **WinDivertSend**(  \_\_in HANDLE handle,  \_\_in const VOID \*pPacket,  \_\_in UINT packetLen,  \_\_out\_opt UINT \*pSendLen,  \_\_in const WINDIVERT\_ADDRESS \*pAddr  ); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* pPacket: A buffer containing a packet to be injected.
* packetLen: The total length of the pPacket buffer.
* pSendLen: The total number of bytes injected. Can be NULL if this information is not required.
* pAddr: The [address](https://reqrypt.org/windivert-doc.html#divert_address) of the injected packet.

**Return Value**  
TRUE if a packet was successfully injected, or FALSE if an error occurred. Use GetLastError() to get the reason for the error.

Common errors include:

|  |  |  |
| --- | --- | --- |
| **Name** | **Code** | **Description** |
| ERROR\_HOST\_UNREACHABLE | 1232 | This error occurs when an *impostor* packet (with pAddr->Impostor set to 1) is injected and the ip.TTL or ipv6.HopLimit field goes to zero. This is a defense of last resort against infinite loops caused by impostor packets. |

**Remarks**  
Injects a packet into the network stack. The injected packet may be one received from [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_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.

Only the WINDIVERT\_LAYER\_NETWORK and WINDIVERT\_LAYER\_NETWORK\_FORWARD [layers](https://reqrypt.org/windivert-doc.html#divert_layers) support packet injection, as summarized below:

|  |  |
| --- | --- |
| **Layer** | **Inject?** |
| WINDIVERT\_LAYER\_NETWORK | ✔ |
| WINDIVERT\_LAYER\_NETWORK\_FORWARD | ✔ |
| WINDIVERT\_LAYER\_FLOW |  |
| WINDIVERT\_LAYER\_SOCKET |  |
| WINDIVERT\_LAYER\_REFLECT |  |

For the WINDIVERT\_LAYER\_NETWORK layer the pAddr->Outbound value determines which direction the packet is injected. If the pAddr->Outbound field is 1, the packet will be injected into the *outbound* path (i.e. a packet leaving the local machine). Else, if pAddr->Outbound is 0, the packet is injected into the *inbound* path (i.e. a packet arriving to the local machine). Note that only the Outbound field, and *not* the IP addresses in the injected packet, determines the packet's direction.

For packets injected into the *inbound* path, the pAddr->Network.IfIdx and pAddr->Network.SubIfIdx fields are assumed to contain valid interface numbers. These may be retrieved from [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) (for packet modification), or from the [IP Helper API](https://msdn.microsoft.com/en-us/library/aa366073%28v=VS.85%29.aspx).

For *outbound* injected packets, the IfIdx and SubIfIdx fields are currently ignored and may be arbitrary values. Injecting an inbound packet on the outbound path *may* work (for some types of packets), however this should be considered "undocumented" behavior, and may be changed in the future.

For *impostor* packets (where pAddr->Impostor is set to 1) WinDivert will automatically decrement the ip.TTL or ipv6.HopLimit fields before reinjection. This is to mitigate infinite loops since WinDivert cannot prevent impostor packets from being captured again by [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv).

Injected packets must have the correct checksums or have the corresponding pAddr->\*Checksum flag unset. A packet/address pair captured by [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) is guaranteed to satisfy this condition, so can be reinjected unmodified without recalculating checksums. Otherwise, if a modification is necessary, checksums can be recalculated using the [WinDivertHelperCalcChecksums()](https://reqrypt.org/windivert-doc.html#divert_helper_calc_checksums) function.

**5.8 WinDivertSendEx**

|  |
| --- |
| BOOL **WinDivertSendEx**(  \_\_in HANDLE handle,  \_\_in const VOID \*pPacket,  \_\_in UINT packetLen,  \_\_out\_opt UINT \*pSendLen,  \_\_in UINT64 flags,  \_\_in const WINDIVERT\_ADDRESS \*pAddr,  \_\_in UINT addrLen,  \_\_inout\_opt LPOVERLAPPED lpOverlapped  ); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* pPacket: A buffer containing the packet(s) to be injected.
* packetLen: The total length of the buffer pPacket.
* pSendLen: The total number of bytes injected. Can be NULL if this information is not required.
* flags: Reserved, set to zero.
* pAddr: The [address(es)](https://reqrypt.org/windivert-doc.html#divert_address) of the injected packet(s).
* addrLen: The total length (in bytes) of the pAddr buffer.
* lpOverlapped: An optional pointer to a OVERLAPPED structure.

**Return Value**  
TRUE if a packet was successfully injected, or FALSE otherwise. Use GetLastError() to get the reason. The error code ERROR\_IO\_PENDING indicates that the overlapped operation has been successfully initiated and that completion will be indicated at a later time. All other codes indicate an error.

**Remarks**  
This function is equivalent to [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_recv) except:

* *Overlapped I/O* is supported via the lpOverlapped parameter.
* *Batched I/O* (i.e., sending multiple packets at once) is supported.

Batched I/O makes it possible to send up to WINDIVERT\_BATCH\_MAX packets at once using a single operation, reducing the number of kernel/user-mode context switches and improving performance. To use batched I/O:

1. pack *N* packets into the pPacket buffer (with no gaps between packets);
2. set packetLen to be the total sum of the *N* packet lengths;
3. pack the corresponding *N* WINDIVERT\_ADDRESS address structures into the pAddr buffer; and
4. set addrLen to be the total size (in bytes) of the pAddr buffer.

**5.9 WinDivertShutdown**

|  |
| --- |
| BOOL **WinDivertShutdown**(  \_\_in HANDLE handle,  \_\_in WINDIVERT\_SHUTDOWN how);  ); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* how: A WINDIVERT\_SHUTDOWN value to indicate how the handle should be shutdown.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
This operation causes all or part of a WinDivert handle to be shut down. The possible values for how are:

|  |  |
| --- | --- |
| **How** | **Description** |
| WINDIVERT\_SHUTDOWN\_RECV | Stop new packets being queued for [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv). |
| WINDIVERT\_SHUTDOWN\_SEND | Stop new packets being injected via [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send). |
| WINDIVERT\_SHUTDOWN\_BOTH | Equivalent to (WINDIVERT\_SHUTDOWN\_RECV | WINDIVERT\_SHUTDOWN\_SEND). |

Note that previously queued packets can still be received after WINDIVERT\_SHUTDOWN\_RECV. When the packet queue is empty, [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv) will fail with ERROR\_NO\_DATA.

**5.10 WinDivertClose**

|  |
| --- |
| BOOL **WinDivertClose**(  \_\_in HANDLE handle  ); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Closes a WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).

**5.11 WinDivertSetParam**

|  |
| --- |
| BOOL **WinDivertSetParam**(  \_\_in HANDLE handle,  \_\_in WINDIVERT\_PARAM param,  \_\_in UINT64 value); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* param: A WinDivert parameter name.
* value: The parameter's new value.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Sets a WinDivert parameter. Currently, the following WinDivert parameters are defined.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| WINDIVERT\_PARAM\_QUEUE\_LENGTH | Sets the maximum length of the packet queue for [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv). The default value is WINDIVERT\_PARAM\_QUEUE\_LENGTH\_DEFAULT, the minimum is WINDIVERT\_PARAM\_QUEUE\_LENGTH\_MIN, and the maximum is WINDIVERT\_PARAM\_QUEUE\_LENGTH\_MAX. |
| WINDIVERT\_PARAM\_QUEUE\_TIME | Sets the minimum time, in milliseconds, a packet can be queued before it is automatically dropped. Packets cannot be queued indefinitely, and ideally, packets should be processed by the application as soon as is possible. Note that this sets the *minimum* time a packet can be queued before it can be dropped. The actual time may be exceed this value. Currently the default value is WINDIVERT\_PARAM\_QUEUE\_TIME\_DEFAULT, the minimum is WINDIVERT\_PARAM\_QUEUE\_TIME\_MIN, and the maximum is WINDIVERT\_PARAM\_QUEUE\_TIME\_MAX. |
| WINDIVERT\_PARAM\_QUEUE\_SIZE | Sets the maximum number of bytes that can be stored in the packet queue for [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv). Currently the default value is WINDIVERT\_PARAM\_QUEUE\_SIZE\_DEFAULT, the minimum is WINDIVERT\_PARAM\_QUEUE\_SIZE\_MIN, and the maximum is WINDIVERT\_PARAM\_QUEUE\_SIZE\_MAX. |

**5.12 WinDivertGetParam**

|  |
| --- |
| BOOL **WinDivertGetParam**(  \_\_in HANDLE handle,  \_\_in WINDIVERT\_PARAM param,  \_\_out UINT64 \*pValue); |

**Parameters**

* handle: A valid WinDivert handle created by [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open).
* param: A WinDivert parameter name.
* value: The parameter's current value.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Gets a WinDivert parameter. This function supports all the parameters from [WinDivertSetParam()](https://reqrypt.org/windivert-doc.html#divert_set_param), and the following additional read-only parameters:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| WINDIVERT\_PARAM\_VERSION\_MAJOR | Returns the major version of the driver. |
| WINDIVERT\_PARAM\_VERSION\_MINOR | Returns the minor version of the driver. |

**6. Helper Programming API**

The WinDivert helper programming API is a collection of definitions and functions designed to make writing WinDivert applications easier. The use of the helper API is optional.

**6.1 WINDIVERT\_IPHDR**

|  |
| --- |
| typedef struct  {  UINT8 HdrLength:4;  UINT8 Version:4;  UINT8 TOS;  UINT16 Length;  UINT16 Id;  UINT16 *...*;  UINT8 TTL;  UINT8 Protocol;  UINT16 Checksum;  UINT32 SrcAddr;  UINT32 DstAddr;  } **WINDIVERT\_IPHDR**, \***PWINDIVERT\_IPHDR**; |

**Fields**  
See [here](https://en.wikipedia.org/wiki/IPv4#Packet_structure) for more information.

**Remarks**  
IPv4 header definition.

The following fields can only be get/set using the following macro definitions:

* *FragOff* with WINDIVERT\_IPHDR\_GET\_FRAGOFF(*hdr*) and WINDIVERT\_IPHDR\_SET\_FRAGOFF(*hdr*, *val*)
* *MF* with WINDIVERT\_IPHDR\_GET\_MF(*hdr*) and WINDIVERT\_IPHDR\_SET\_MF(*hdr*, *val*)
* *DF* with WINDIVERT\_IPHDR\_GET\_DF(*hdr*) and WINDIVERT\_IPHDR\_SET\_DF(*hdr*, *val*)
* *Reserved* with WINDIVERT\_IPHDR\_GET\_RESERVED(*hdr*) and WINDIVERT\_IPHDR\_SET\_RESERVED(*hdr*, *val*)

**6.2 WINDIVERT\_IPV6HDR**

|  |
| --- |
| typedef struct  {  UINT32 Version:4;  UINT32 ...:28;  UINT16 Length;  UINT8 NextHdr;  UINT8 HopLimit;  UINT32 SrcAddr[4];  UINT32 DstAddr[4];  } **WINDIVERT\_IPV6HDR**, \***PWINDIVERT\_IPV6HDR**; |

**Fields**  
See [here](https://en.wikipedia.org/wiki/IPv6_packet#Fixed_header) for more information.

**Remarks**  
IPv6 header definition.

The following fields can only be get/set using the following macro definitions:

* *TrafficClass* with WINDIVERT\_IPV6HDR\_GET\_TRAFFICCLASS(*hdr*) and WINDIVERT\_IPV6HDR\_SET\_TRAFFICCLASS(*hdr*, *val*)
* *FlowLabel* with WINDIVERT\_IPV6HDR\_GET\_FLOWLABEL(*hdr*) and WINDIVERT\_IPV6HDR\_SET\_FLOWLABEL(*hdr*, *val*)

**6.3 WINDIVERT\_ICMPHDR**

|  |
| --- |
| typedef struct  {  UINT8 Type;  UINT8 Code;  UINT16 Checksum;  UINT32 Body;  } **WINDIVERT\_ICMPHDR**, \***PWINDIVERT\_ICMPHDR**; |

**Fields**  
See [here](https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#ICMP_segment_structure) for more information.

**Remarks**  
ICMP header definition.

**6.4 WINDIVERT\_ICMPV6HDR**

|  |
| --- |
| typedef struct  {  UINT8 Type;  UINT8 Code;  UINT16 Checksum;  UINT32 Body;  } **WINDIVERT\_ICMPV6HDR**, \***PWINDIVERT\_ICMPV6HDR**; |

**Fields**  
See [here](https://en.wikipedia.org/wiki/ICMPv6#Packet_format) for more information.

**Remarks**  
ICMPv6 header definition.

**6.5 WINDIVERT\_TCPHDR**

|  |
| --- |
| typedef struct  {  UINT16 SrcPort;  UINT16 DstPort;  UINT32 SeqNum;  UINT32 AckNum;  UINT16 Reserved1:4;  UINT16 HdrLength:4;  UINT16 Fin:1;  UINT16 Syn:1;  UINT16 Rst:1;  UINT16 Psh:1;  UINT16 Ack:1;  UINT16 Urg:1;  UINT16 Reserved2:2;  UINT16 Window;  UINT16 Checksum;  UINT16 UrgPtr;  } **WINDIVERT\_TCPHDR**, \***PWINDIVERT\_TCPHDR**; |

**Fields**  
See [here](https://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_segment_structure) for more information.

**Remarks**  
TCP header definition.

**6.6 WINDIVERT\_UDPHDR**

|  |
| --- |
| typedef struct  {  UINT16 SrcPort;  UINT16 DstPort;  UINT16 Length;  UINT16 Checksum;  } **WINDIVERT\_UDPHDR**, \***PWINDIVERT\_UDPHDR**; |

**Fields**  
See [here](https://en.wikipedia.org/wiki/User_Datagram_Protocol#Packet_structure) for more information.

**Remarks**  
UDP header definition.

**6.7 WinDivertHelperParsePacket**

|  |
| --- |
| BOOL **WinDivertHelperParsePacket**(  \_\_in PVOID pPacket,  \_\_in UINT packetLen,  \_\_out\_opt PWINDIVERT\_IPHDR \*ppIpHdr,  \_\_out\_opt PWINDIVERT\_IPV6HDR \*ppIpv6Hdr,  \_\_out\_opt UINT8 \*pProtocol,  \_\_out\_opt PWINDIVERT\_ICMPHDR \*ppIcmpHdr,  \_\_out\_opt PWINDIVERT\_ICMPV6HDR \*ppIcmpv6Hdr,  \_\_out\_opt PWINDIVERT\_TCPHDR \*ppTcpHdr,  \_\_out\_opt PWINDIVERT\_UDPHDR \*ppUdpHdr,  \_\_out\_opt PVOID \*ppData,  \_\_out\_opt UINT \*pDataLen,  \_\_out\_opt PVOID \*ppNext,  \_\_out\_opt UINT \*pNextLen  ); |

**Parameters**

* pPacket: The packet(s) to be parsed.
* packetLen: The total length of the packet(s) pPacket.
* ppIpHdr: Output pointer to a WINDIVERT\_IPHDR.
* ppIpv6Hdr: Output pointer to a WINDIVERT\_IPV6HDR.
* pProtocol: Output transport protocol.
* ppIcmpHdr: Output pointer to a WINDIVERT\_ICMPHDR.
* ppIcmpv6Hdr: Output pointer to a WINDIVERT\_ICMPV6HDR.
* ppTcpHdr: Output pointer to a WINDIVERT\_TCPHDR.
* ppUdpHdr: Output pointer to a WINDIVERT\_UDPHDR.
* ppData: Output pointer to the packet's data/payload.
* pDataLen: Output data/payload length.
* ppNext: Output pointer to the next packet (if present).
* pNextLen: Output next packet length.

**Return Value**  
TRUE if successful, FALSE if an error occurred.

**Remarks**  
Parses a raw packet or batch of packets (e.g. from [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv)) into the various packet headers and/or payloads that may or may not be present.

Each output parameter may be NULL or non-NULL. For non-NULL parameters, this function will write the pointer to the corresponding header/payload if it exists, or will write NULL otherwise. Any non-NULL pointer that is returned:

1. Is a pointer into the original pPacket packet buffer; and
2. There is enough space in pPacket to fit the header.

This function does not do any verification of the header/payload contents beyond checking the header length and any other minimal information required for parsing. This function will always succeed provided the pPacket buffer contains at least one IPv4 or IPv6 header and the packetLen is correct.

By default this function will parse a single packet. However, if either ppNext or pNextLen are non-NULL, then the pPacket parameter can point to a batch (>1) of packets (and packetLen can be the total length of the batch). In this case, the function will parse the first packet, and a pointer to the remaining packet(s) will be written to ppNext, and the remaining length will be written to pNextLen. This makes it convenient to loop over every packet in the batch as follows:

while (WinDivertHelperParsePacket(pPacket, packetLen, ..., &pPacket, &packetLen))

{

...

}

**6.8 WinDivertHelperHashPacket**

|  |
| --- |
| UINT64 **WinDivertHelperHashPacket**(  \_\_in const VOID \*pPacket,  \_\_in UINT packetLen,  \_\_in UINT64 seed = 0  ); |

**Parameters**

* pPacket: The packet to be hashed.
* packetLen: The total length of the packet pPacket.
* seed: An optional seed value.

**Return Value**  
A 64bit hash value.

**Remarks**  
Calculates a 64bit hash value of the given packet. Note that the hash function depends on the *packet's IP and transport headers only*, and not the payload of the packet. That said, a weak dependency on the payload will exist if the TCP/UDP checksums are valid. The hash function itself is based on the [xxHash](https://cyan4973.github.io/xxHash/) algorithm and is **not** cryptographic.

The optional seed value is also incorporated into the hash.

**6.9 WinDivertHelperParseIPv4Address**

|  |
| --- |
| BOOL **WinDivertHelperParseIPv4Address**(  \_\_in const char \*addrStr,  \_\_out\_opt UINT32 \*pAddr  ); |

**Parameters**

* addrStr: The address string.
* pAddr: Output address.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Parses an IPv4 address stored in addrStr. If pAddr is non-NULL, the result is be stored in host-byte-order. Use [WinDivertHelperHtonl()](https://reqrypt.org/windivert-doc.html#divert_helper_hton) to convert the result into network-byte-order.

**6.10 WinDivertHelperParseIPv6Address**

|  |
| --- |
| BOOL **WinDivertHelperParseIPv6Address**(  \_\_in const char \*addrStr,  \_\_out\_opt UINT32 \*pAddr  ); |

**Parameters**

* addrStr: The address string.
* pAddr: Output address.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Parses an IPv6 address stored in addrStr. If pAddr is non-NULL, the buffer assumed to be large enough to hold a 16-byte IPv6 address. The result is stored in host-byte-order. Use [WinDivertHelperHtonIPv6Address()](https://reqrypt.org/windivert-doc.html#divert_helper_hton) to convert the result into network-byte-order.

**6.11 WinDivertHelperFormatIPv4Address**

|  |
| --- |
| BOOL **WinDivertHelperFormatIPv4Address**(  \_\_in UINT32 addr,  \_\_out char \*buffer,  \_\_in UINT bufLen  ); |

**Parameters**

* addr: The IPv4 address in host-byte order.
* buffer: The buffer to store the formatted string.
* bufLen: The length of buffer.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Convert an IPv4 address into a string.

**6.12 WinDivertHelperParseIPv6Address**

|  |
| --- |
| BOOL **WinDivertHelperFormatIPv6Address**(  \_\_in const UINT32 \*pAddr,  \_\_out char \*buffer,  \_\_in UINT bufLen  ); |

**Parameters**

* pAddr: The IPv6 address in host-byte order.
* buffer: The buffer to store the formatted string.
* bufLen: The length of buffer.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Convert an IPv6 address into a string.

**6.13 WinDivertHelperCalcChecksums**

|  |
| --- |
| BOOL **WinDivertHelperCalcChecksums**(  \_\_inout VOID \*pPacket,  \_\_in UINT packetLen,  \_\_out\_opt WINDIVERT\_ADDRESS \*pAddr,  \_\_in UINT64 flags  ); |

**Parameters**

* pPacket: The packet to be modified.
* packetLen: The total length of the packet pPacket.
* pAddr: Optional pointer to a [WINDIVERT\_ADDRESS](https://reqrypt.org/windivert-doc.html#divert_address) structure.
* flags: One or more of the following flags:
  + WINDIVERT\_HELPER\_NO\_IP\_CHECKSUM: Do not calculate the IPv4 checksum.
  + WINDIVERT\_HELPER\_NO\_ICMP\_CHECKSUM: Do not calculate the ICMP checksum.
  + WINDIVERT\_HELPER\_NO\_ICMPV6\_CHECKSUM: Do not calculate the ICMPv6 checksum.
  + WINDIVERT\_HELPER\_NO\_TCP\_CHECKSUM: Do not calculate the TCP checksum.
  + WINDIVERT\_HELPER\_NO\_UDP\_CHECKSUM: Do not calculate the UDP checksum.

**Return Value**  
TRUE if successful, FALSE if an error occurred.

**Remarks**  
(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 [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send).

By default this function will calculate each checksum from scratch, even if the existing checksum is correct. This may be inefficient for some applications. For better performance, incremental checksum calculations should be used instead (not provided by this API).

If pAddr is non-NULL, this function sets the corresponding \*Checksum flag (see [WINDIVERT\_ADDRESS](https://reqrypt.org/windivert-doc.html#divert_address)). Normally, pAddr should point to the address passed to [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send) for packet injection.

**6.14 WinDivertHelperDecrementTTL**

|  |
| --- |
| BOOL **WinDivertHelperDecrementTTL**(  \_\_inout VOID \*packet,  \_\_in packetLen  ); |

**Parameters**

* pPacket: The packet to be modified.
* packetLen: The total length of the packet pPacket.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Returns FALSE if the ip.TTL or ipv6.HopHimit fields go to 0.

**Remarks**  
Decrements the ip.TTL or ipv6.HopHimit field by 1, and returns TRUE only if the result is non-zero. This is useful for applications where packet loops may be a problem.

For IPv4, this function will preserve the validity of the IPv4 checksum. That is, if the packet had a valid checksum before the operation, the resulting checksum will also be valid after the operation. This function updates the checksum field incrementally.

**6.15 WinDivertHelperCompileFilter**

|  |
| --- |
| BOOL **WinDivertHelperCompileFilter**(  \_\_in const char \*filter,  \_\_in WINDIVERT\_LAYER layer,  \_\_out\_opt char \*object,  \_\_in UINT objLen,  \_\_out\_opt const char \*\*errorStr,  \_\_out\_opt UINT \*errorPos  ); |

**Parameters**

* filter: The packet filter string to be checked.
* layer: The layer.
* object: The compiled filter object.
* objLen: The length of the object buffer.
* errorStr: The error description.
* errorPos: The error position.

**Return Value**  
TRUE if the packet filter compilation is successful, FALSE otherwise.

**Remarks**  
Compiles the given packet filter string into a compact object representation that is optionally stored in object if non-NULL. The object representation is a valid null terminated C string, but is otherwise opaque and not meant to be human readable. The object representation can be passed to all WinDivert functions, such as [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open), in place of the human-readable filter string equivalent.

The compilation operation will succeed if the given filter string is valid with respect to the [filter language](https://reqrypt.org/windivert-doc.html#filter_language). Otherwise, if the filter is invalid, then a human readable description of the error is returned by errorStr (if non-NULL), and the error's position is returned by errorPos (if non-NULL).

Note that all strings returned through errorStr are global static objects, and therefore do not need to be deallocated.

**6.16 WinDivertHelperEvalFilter**

|  |
| --- |
| BOOL **WinDivertHelperEvalFilter**(  \_\_in const char \*filter,  \_\_in const VOID \*pPacket,  \_\_in UINT packetLen,  \_\_in const WINDIVERT\_ADDRESS \*pAddr  ); |

**Parameters**

* filter: The packet filter string to be evaluated.
* pPacket: The packet.
* packetLen: The total length of the packet pPacket.
* pAddr: The WINDIVERT\_ADDRESS of the packet pPacket.

**Return Value**  
TRUE if the packet matches the filter string, FALSE otherwise.

**Remarks**  
Evaluates the given packet against the given packet filter string. This function returns TRUE if the packet matches, and returns FALSE otherwise.

This function also returns FALSE if an error occurs, in which case GetLastError() can be used to get the reason for the error. Otherwise, if no error occurred, GetLastError() will return 0.

Note that this function is relatively slow since the packet filter string will be (re)compiled for each call. This overhead can be minimized by pre-compiling the filter string into the object representation using the [WinDivertHelperCompileFilter()](https://reqrypt.org/windivert-doc.html#divert_helper_compile_filter) function.

**6.17 WinDivertHelperFormatFilter**

|  |
| --- |
| BOOL **WinDivertHelperEvalFilter**(  \_\_in const char \*filter,  \_\_in WINDIVERT\_LAYER layer,  \_\_out char \*buffer,  \_\_in UINT bufLen  ); |

**Parameters**

* filter: The packet filter string to be evaluated.
* layer: The layer.
* buffer: A buffer for the formatted filter.
* bufLen: The length of buffer.

**Return Value**  
TRUE if successful, FALSE if an error occurred. Use GetLastError() to get the reason for the error.

**Remarks**  
Formats the given filter string or object. This function is mainly useful for decompiling the filter object representation back into a human-readable filter string representation. One application is the WINDIVERT\_LAYER\_REFLECT layer, where the filter object associated with the reflection event is returned by [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv).

**6.18 WinDivertHelperNtoh\***

|  |
| --- |
| UINT16 **WinDivertHelperNtohs**(  \_\_in UINT16 x  );  UINT32 **WinDivertHelperNtohl**(  \_\_in UINT32 x  );  UINT64 **WinDivertHelperNtohll**(  \_\_in UINT64 x  );  void **WinDivertHelperNtohIPv6Address**(  \_\_in const UINT \*inAddr,  \_\_out UINT \*outAddr  ); |

**Parameters**

* x: The input value in network byte-order.
* inAddr: The input IPv6 address in network byte-order.
* outAddr: A buffer for the output IPv6 address in host byte-order.

**Return Value**  
The output value in host byte order.

**Remarks**  
Converts a value/IPv6-address from network to host byte-order.

**6.19 WinDivertHelperHton\***

|  |
| --- |
| UINT16 **WinDivertHelperHtons**(  \_\_in UINT16 x  );  UINT32 **WinDivertHelperHtonl**(  \_\_in UINT32 x  );  UINT64 **WinDivertHelperHtonll**(  \_\_in UINT64 x  );  void **WinDivertHelperHtonIPv6Address**(  \_\_in const UINT \*inAddr,  \_\_out UINT \*outAddr  ); |

**Parameters**

* x: The input value in host byte-order.
* inAddr: The input IPv6 address in host byte-order.
* outAddr: A buffer for the output IPv6 address in network byte-order.

**Return Value**  
The output value in network byte order.

**Remarks**  
Converts a value/IPv6-address from host to network byte-order.

**7. Filter Language**

The [WinDivertOpen()](https://reqrypt.org/windivert-doc.html#divert_open) function accepts a string containing a *filter*. Only packets/events that match the filter will be blocked and/or captured. All other non-matching packets/events will be allowed to continue as normal.

The filter allows an application to select only a subset traffic that is of interest. For example, a HTTP blacklist filter is only interested in packets that might contain URLs. This could be achieved using the following filter.

HANDLE handle = WinDivertOpen(

"outbound and "

"tcp.PayloadLength > 0 and "

"tcp.DstPort == 80", 0, 0, 0);

This filter selects only the subset of all traffic that is:

1. outbound;
2. contains a non-empty payload; and
3. has TCP destination port 80 (i.e. HTTP web traffic).

A *filter* is a Boolean expression of the form:

*FILTER* := true | false | *FILTER* and *FILTER* | *FILTER* or *FILTER* | (*FILTER*) | (*FILTER*? *FILTER*: *FILTER*) | *TEST*

C-style syntax &&, ||, and ! may also be used instead of and, or, and not, respectively. C-style *conditional operators* are also supported, where the expression (A? B: C) evaluates to:

* B if A evaluates to true; or
* C if A evaluates to false.

A *test* is of the following form:

*TEST* := *TEST0* | not *TEST0*

*TEST0* := *FIELD* | *FIELD* op *VAL*

where op is one of the following:

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == or = | Equal |
| != | Not equal |
| < | Less-than |
| > | Greater-than |
| <= | Less-than-or-equal |
| >= | Greater-than-or-equal |

and *VAL* is a decimal number, hexadecimal number, IPv4 address, IPv6 address or a layer-specific macro. If the op *VAL* is missing, the test is implicitly *FIELD* != 0.

Finally, a *field* is some layer-specific property matching the packet or event. The possible fields are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Field** | **Layer** | | | | | **Description** |
|  | **NETWORK** | **FORWARD** | **FLOW** | **SOCKET** | **REFLECT** |  |
| zero | ✔ | ✔ | ✔ | ✔ | ✔ | The value zero |
| timestamp | ✔ | ✔ | ✔ | ✔ | ✔ | The packet/event timestamp |
| event | ✔ | ✔ | ✔ | ✔ | ✔ | The event |
| outbound | ✔ |  | ✔ |  |  | Is outbound? |
| inbound | ✔ |  | ✔ |  |  | Is inbound? |
| ifIdx | ✔ | ✔ |  |  |  | Interface index |
| subIfIdx | ✔ | ✔ |  |  |  | Sub-interface index |
| loopback | ✔ |  | ✔ | ✔ |  | Is loopback packet? |
| impostor | ✔ | ✔ |  |  |  | Is impostor packet? |
| fragment | ✔ | ✔ |  |  |  | Is IP fragment packet? |
| endpointId |  |  | ✔ | ✔ |  | Endpoint ID |
| parentEndpointId |  |  | ✔ | ✔ |  | Parent endpoint ID |
| processId |  |  | ✔ | ✔ | ✔ | Process ID |
| random8 | ✔ | ✔ |  |  |  | 8-bit random number |
| random16 | ✔ | ✔ |  |  |  | 16-bit random number |
| random32 | ✔ | ✔ |  |  |  | 32-bit random number |
| layer |  |  |  |  | ✔ | The handle's layer |
| priority |  |  |  |  | ✔ | The handle's priority |
| packet[i] | ✔ | ✔ |  |  |  | The ith 8-bit word of the packet |
| packet16[i] | ✔ | ✔ |  |  |  | The ith 16-bit word of the packet |
| packet32[i] | ✔ | ✔ |  |  |  | The ith 32-bit word of the packet |
| length | ✔ | ✔ |  |  |  | The packet length |
| ip | ✔ | ✔ | ✔ | ✔ |  | Is IPv4? |
| ipv6 | ✔ | ✔ | ✔ | ✔ |  | Is IPv6? |
| icmp | ✔ | ✔ | ✔ | ✔ |  | Is ICMP? |
| icmpv6 | ✔ | ✔ | ✔ | ✔ |  | Is ICMPv6? |
| tcp | ✔ | ✔ | ✔ | ✔ |  | Is TCP? |
| udp | ✔ | ✔ | ✔ | ✔ |  | Is UDP? |
| protocol | ✔ |  | ✔ | ✔ |  | The protocol |
| localAddr | ✔ |  | ✔ | ✔ |  | The local address |
| localPort | ✔ |  | ✔ | ✔ |  | The local port |
| remoteAddr | ✔ |  | ✔ | ✔ |  | The remote address |
| remotePort | ✔ |  | ✔ | ✔ |  | The remote port |
| ip.\* | ✔ | ✔ |  |  |  | IPv4 fields (see WINDIVERT\_IPHDR) |
| ipv6.\* | ✔ | ✔ |  |  |  | IPv6 fields (see WINDIVERT\_IPV6HDR) |
| icmp.\* | ✔ | ✔ |  |  |  | ICMP fields (see WINDIVERT\_ICMPHDR) |
| icmpv6.\* | ✔ | ✔ |  |  |  | ICMPV6 fields (see WINDIVERT\_ICMPV6HDR) |
| tcp.\* | ✔ | ✔ |  |  |  | TCP fields (see WINDIVERT\_TCPHDR) |
| tcp.PayloadLength | ✔ | ✔ |  |  |  | The TCP payload length |
| tcp.Payload[i] | ✔ | ✔ |  |  |  | The ith 8-bit word of the TCP payload |
| tcp.Payload16[i] | ✔ | ✔ |  |  |  | The ith 16-bit word of the TCP payload |
| tcp.Payload32[i] | ✔ | ✔ |  |  |  | The ith 32-bit word of the TCP payload |
| udp.\* | ✔ | ✔ |  |  |  | UDP fields (see WINDIVERT\_UDPHDR) |
| udp.PayloadLength | ✔ | ✔ |  |  |  | The UDP payload length |
| udp.Payload[i] | ✔ | ✔ |  |  |  | The ith 8-bit word of the UDP payload |
| udp.Payload16[i] | ✔ | ✔ |  |  |  | The ith 16-bit word of the UDP payload |
| udp.Payload32[i] | ✔ | ✔ |  |  |  | The ith 32-bit word of the UDP payload |

A *test* will also fails if the field is not relevant. For example, the test tcp.DstPort == 80 will fail if the packet does not contain a TCP header.

The processId field matches the ID of the process associated to an event. Due to technical limitations, this field is not supported by the WINDIVERT\_LAYER\_NETWORK\* layers. That said, it is usually possible to associate process IDs to network packets matching the same network 5-tuple. Note that a fundamental race condition exists between the processId and the termination of the corresponding process, see the [know issues](https://reqrypt.org/windivert-doc.html#known_issues) listed below.

The packet\*[i], tcp.Payload\*[i] and udp.Payload\*[i] fields take an *index* parameter (i). The following indexing schemes are supported:

* *Undecorated integer* (e.g., packet32[10]): evaluates to the ith word from the start of the packet/payload. This is essentially C-style array indexing;
* *Negative decorated integer* (e.g., packet32[-10]): evaluates to the ith word from the **end** of the packet/payload. Here the index (-1) is the first full word that fits; and
* *Byte decorated (negative) integer* (e.g., packet32[10b] or packet32[-10b]): evaluated to the word offset by i bytes from the start (or end) of the packet/payload.

These fields can be used to match filters against the contents of packets/payloads in addition to address/header information. Words are assumed to be in network-byte ordering. If the index is out-of-bounds then the corresponding *test* is deemed to have failed.

The random\* fields are not really random but use a deterministic hash value calculated using the [WinDivertHelperHashPacket()](https://reqrypt.org/windivert-doc.html#divert_helper_hash_packet) function.

Layer-specific macros make it possible to match events and layers symbolically, e.g., event == CONNECT or layer == SOCKET. The possible macros are:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Macro** | **Layer** | | | | | **Value** |
|  | **NETWORK** | **FORWARD** | **FLOW** | **SOCKET** | **REFLECT** |  |
| TRUE | ✔ | ✔ | ✔ | ✔ | ✔ | 1 |
| FALSE | ✔ | ✔ | ✔ | ✔ | ✔ | 0 |
| TCP | ✔ | ✔ | ✔ | ✔ | ✔ | IPPROTO\_TCP (6) |
| UDP | ✔ | ✔ | ✔ | ✔ | ✔ | IPPROTO\_UDP (17) |
| ICMP | ✔ | ✔ | ✔ | ✔ | ✔ | IPPROTO\_ICMP (1) |
| ICMPV6 | ✔ | ✔ | ✔ | ✔ | ✔ | IPPROTO\_ICMPV6 (58) |
| PACKET | ✔ | ✔ |  |  |  | WINDIVERT\_EVENT\_NETWORK\_PACKET |
| ESTABLISHED |  |  | ✔ |  |  | WINDIVERT\_EVENT\_FLOW\_ESTABLISHED |
| DELETED |  |  | ✔ |  |  | WINDIVERT\_EVENT\_FLOW\_DELETED |
| BIND |  |  |  | ✔ |  | WINDIVERT\_EVENT\_SOCKET\_BIND |
| CONNECT |  |  |  | ✔ |  | WINDIVERT\_EVENT\_SOCKET\_CONNECT |
| ACCEPT |  |  |  | ✔ |  | WINDIVERT\_EVENT\_SOCKET\_ACCEPT |
| LISTEN |  |  |  | ✔ |  | WINDIVERT\_EVENT\_SOCKET\_LISTEN |
| OPEN |  |  |  |  | ✔ | WINDIVERT\_EVENT\_REFLECT\_OPEN |
| CLOSE |  |  |  | ✔ | ✔ | WINDIVERT\_EVENT\_SOCKET\_CLOSE for the SOCKET layer, or WINDIVERT\_EVENT\_REFLECT\_CLOSE for the REFLECT layer. |
| NETWORK |  |  |  |  | ✔ | WINDIVERT\_LAYER\_NETWORK |
| NETWORK\_FORWARD |  |  |  |  | ✔ | WINDIVERT\_LAYER\_NETWORK\_FORWARD |
| FLOW |  |  |  |  | ✔ | WINDIVERT\_LAYER\_FLOW |
| SOCKET |  |  |  |  | ✔ | WINDIVERT\_LAYER\_SOCKET |
| REFLECT |  |  |  |  | ✔ | WINDIVERT\_LAYER\_REFLECT |

**7.1 Filter Examples**

1. Divert all outbound (non-local) web traffic:
2. HANDLE handle = WinDivertOpen(
3. "outbound and !loopback and "
4. "(tcp.DstPort == 80 or udp.DstPort == 53)",
5. 0, 0, 0
6. );
7. Divert all inbound TCP SYNs:
8. HANDLE handle = WinDivertOpen(
9. "inbound and "
10. "tcp.Syn",
11. 0, 0, 0
12. );
13. Divert all traffic:
14. HANDLE handle = WinDivertOpen("true", 0, 0, 0);
15. Divert no traffic:
16. HANDLE handle = WinDivertOpen("false", 0, 0, 0);

This is useful for packet injection.

**7.2 Filter Usage**

The purpose of the filter is to help applications select the subset of all network traffic that the application is interested in. Ideally the filter should be

1. As short as possible; and
2. As selective as possible.

For some applications these two objectives can conflict. That is, a selective filter is not short, and a short filter is not selective. For such applications the developer should experiment with different filter configurations and carefully measure the performance impact to find the optimal solution.

**8. Performance**

Using WinDivert to redirect network traffic to/from a user application incurs performance overheads, such as copying packet data and user/kernel mode context switching. Under heavy load (≥1Gbps) these overheads can be significant. The following techniques can be used to reduce overheads (in order of importance):

1. *Selective Filter*: Only select the subset of network traffic the user application is interested in. Non-matching traffic will continue to use the default path without incurring additional overheads.
2. *Batch Mode*: The [WinDivertRecvEx()](https://reqrypt.org/windivert-doc.html#divert_recv_ex) and [WinDivertSendEx()](https://reqrypt.org/windivert-doc.html#divert_send_ex) functions support *batching* that allows several packets to be received/sent at once. This can significantly reduce the overheads relating to user/kernel mode context switching.
3. *Multi-threading*: It is possible to spread packet processing over multiple threads ensuring that the user application does not become a bottleneck. That said, sometimes spawning too many threads can degrade performance.
4. *Small Buffers*: Large buffers generally incur more overhead compared to smaller buffers. In general, the buffer size should reflect the expected usage as closely as possible.
5. *Simple Filters*: Currently WinDivert does not optimize the filter compilation, so it is up to the user application to ensure the filter is simple/optimized.
6. *Overlapped I/O*: This allows the user application to do additional tasks at the same time as receive/send operations, which may improve performance for some applications. It is also possible for a single thread to initiate several receive/send operations at once. However, using overlapped I/O can be tricky, and it is important that all buffers passed to [WinDivertRecvEx()](https://reqrypt.org/windivert-doc.html#divert_recv_ex) or [WinDivertSendEx()](https://reqrypt.org/windivert-doc.html#divert_send_ex) (including the OVERLAPPED structure) are not modified by the user application until the operation completes.
7. *Queue length/size/time*: If these values are too small then some packets may be dropped under heavy load. These values can be controlled using the [WinDivertSetParam()](https://reqrypt.org/windivert-doc.html#divert_set_param) function.

The passthru.exe [sample program](https://reqrypt.org/windivert-doc.html#samples) can be used to experiment with different batch sizes and thread counts.

**9. Samples**

Some samples have been provided to demonstrate the WinDivert API. The sample programs are:

* webfilter.exe: A simple URL blacklist filter. This program monitors outbound HTTP traffic. If it finds a URL request that matches the blacklist, it hijacks the TCP connection, reseting the connection at the server's end, and sending a simple block-page to the browser. The blacklist(s) are specified at the command-line.
* netdump.exe: A simple packet sniffer based on the WinDivert filter language. This program takes a filter specified at the command line, and prints information about any packet that matches the filter. This example uses WinDivert in "packet sniffing" mode, similar to winpcap. However, unlike winpcap, WinDivert can see local (loopback) packets.
* netfilter.exe: A simple firewall based on the WinDivert filter language. This program takes a filter specified at the command line, and blocks any packet that matches the filter. It blocks TCP by sending a TCP reset, UDP by an ICMP message, and all other traffic it simply drops. This is similar to the Linux iptables command with the -j REJECT option.
* passthru.exe: A simple program that simply re-injects every packet it captures. This example has a configurable batch-size and thread count, and so is useful for performance testing or as a starting point for more interesting applications.
* streamdump.exe: A simple program that demonstrates how to handle streams using WinDivert. The basic idea is to divert outbound TCP connections to a local proxy server which can capture or manipulate the stream.
* flowtrack.exe: A program that tracks all network flows to and from the local machine, including information such as the ID of the responsible process. The flowtrack sample demonstrates the WINDIVERT\_LAYER\_FLOW layer.
* socketdump.exe: Dumps socket operations (bind(), connect(), etc.) and the ID of the responsible process. The socketdump sample demonstrates the WINDIVERT\_LAYER\_SOCKET layer.
* windivertctl.exe allows the user to query which processes are using WinDivert via the list or watch commands, or to terminate all such processes using the kill command. The windivertctl.exe can also forcibly remove the WinDivert driver using the uninstall command. The windivertctl sample demonstrates the WINDIVERT\_LAYER\_REFLECT layer.

The samples are intended for educational purposes only, and are not fully-featured applications.

The following basic template for a WinDivert application using the WINDIVERT\_LAYER\_NETWORK layer. The basic idea is to open a WinDivert handle, then enter a capture-modify-reinject loop:

HANDLE handle; // WinDivert handle

WINDIVERT\_ADDRESS addr; // Packet address

char packet[MAXBUF]; // Packet buffer

UINT packetLen;

// Open some filter

handle = WinDivertOpen("...", WINDIVERT\_LAYER\_NETWORK, 0, 0);

if (handle == INVALID\_HANDLE\_VALUE)

{

// Handle error

exit(1);

}

// Main capture-modify-inject loop:

while (TRUE)

{

if (!WinDivertRecv(handle, packet, sizeof(packet), &packetLen, &addr))

{

// Handle recv error

continue;

}

// Modify packet.

WinDivertHelperCalcChecksums(packet, packetLen, &addr, 0);

if (!WinDivertSend(handle, packet, packetLen, NULL, &addr))

{

// Handle send error

continue;

}

}

For applications that do not need to modify the packet, a better approach is to open the WinDivert handle with the WINDIVERT\_FLAG\_SNIFF flag set, and not re-inject the packet with [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send). See the netdump.exe sample program for an example of this usage.

**10. Known Issues**

WinDivert has some known limitations listed below:

* *Injecting inbound ICMP/ICMPv6 messages*: Calling [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send) will fail with an error for certain types of inbound ICMP/ICMPv6 messages. This is probably because the Windows TCP/IP stack does not handle such messages. Such errors are harmless and can be ignored.
* *The forward layer does not interact well with the Windows NAT*: It is not possible to block packets pre-NAT with WinDivert. As a general principle, you should not try and mix WinDivert at the forward layer with the Windows NAT implementation.
* *Re-injecting unmodified packets can lead to infinite loops*: If two or more Windows Filtering Platform (WFP) callout drivers (including WinDivert applications) block and inject unmodified copies of packets then this can lead to an infinite loop. If such a loop occurs, [WinDivertSend()](https://reqrypt.org/windivert-doc.html#divert_send) will eventually fail with error ERROR\_HOST\_UNREACHABLE. Unfortunately, such errors are not easy to fix. Some crude solutions include: (1) removing the incompatible driver, or (2) ignoring all packets with ip.TTL or ipv6.HopLimit less than the Windows DefaultTTL registry value. See [GitHub issue #41](https://github.com/basil00/Divert/issues/41) for more information.
* *WinDivert can cause the MSVC x86\_64 debugger to deadlock*: The deadlock occurs because the debugger uses local sockets. Thus: the debugger pauses the WinDivert application, which stops packets from being processed, which causes the debugger wait forever on input from a socket. The deadlock can be avoided by ignoring loopback traffic. See [GitHub issue #26](https://github.com/basil00/Divert/issues/26) for more information.
* *WinDivert can cause packets to be out-of-order*: Simply running the passthru.exe sample program can cause packets to become out-of-order. This is not a bug, since there is no requirement for packets to remain in-order. However, this may affect other buggy software (e.g. some buggy NAT implementations) that incorrectly assume packets to be in-order.
* *A race condition exists between addr.\*.processId and process termination.* By the time an event is received using [WinDivertRecv()](https://reqrypt.org/windivert-doc.html#divert_recv), it is possible that the process responsible for the event has already terminated. Furthermore, it is theoretically possible that the processId has been reassigned to an unrelated process. This problem can be partly mitigated by comparing the timestamp (addr.Timestamp) with the creation time of the process. If the process is newer, then the ID has been reassigned. This race condition does **not** affect the WINDIVERT\_EVENT\_REFLECT\_OPEN event. In this special case, the addr.Reflect.processId is guaranteed to be valid until the corresponding WINDIVERT\_EVENT\_REFLECT\_CLOSE event is received by the user application or is dropped (filter mismatch or timeout).