DCE TCP Inspector

DCE TCP Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	None
Enabled	`true`

The DCE/RPC protocol allows processes on separate network hosts to communicate as if the processes were on the same host. These inter-process communications are commonly transported between hosts over TCP. Within the TCP transport, DCE/RPC might also be further encapsulated in the Windows Server Message Block (SMB) protocol or in Samba, an open-source SMB implementation used for inter-process communication in a mixed environment comprised of Windows, and UNIX or Linux operating systems.

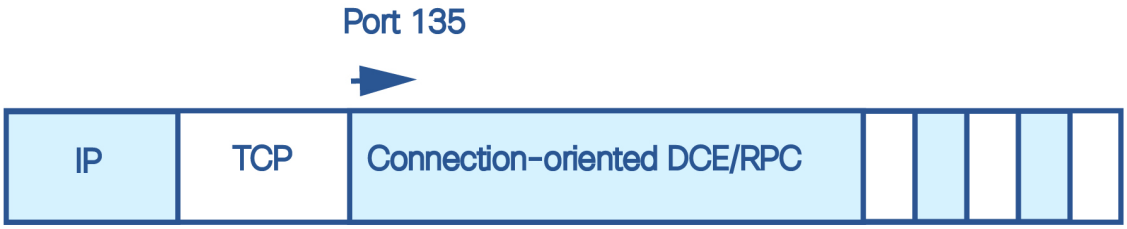
Although most DCE/RPC exploits occur in DCE/RPC client requests targeted for DCE/RPC servers, which could be practically any host on your network that is running Windows or Samba, exploits can also occur in server responses.

IP encapsulates all DCE/RPC transports. TCP transports all connection-oriented DCE/RPC. The DCE TCP inspector detects connection-oriented DCE/RPC and uses protocol-specific characteristics (such as header length and data fragment order) to:

- Detect DCE/RPC requests and responses encapsulated in TCP transports, including TCP-transported DCE/RPC using version 1 RPC over HTTP.
- Analyze DCE/RPC data streams and detect anomalous behavior and evasion techniques in DCE/RPC traffic.
- Defragment DCE/RPC.

- Normalize DCE/RPC traffic for processing by the rules engine.

The following diagram illustrates the point at which the DCE TCP inspector begins processing traffic for the TCP transport.



The well-known TCP port 135 identifies DCE/RPC traffic in the TCP transport. The figure does not include RPC over HTTP. For RPC over HTTP, connection-oriented DCE/RPC is transported directly over TCP as shown in the figure after an initial setup sequence over HTTP.

Target-Based Policies

Windows and Samba DCE/RPC implementations differ significantly. For example, all versions of Windows use the DCE/RPC context ID in the first fragment when defragmenting DCE/RPC traffic, and all versions of Samba use the context ID in the last fragment. As another example, Windows Vista uses the opnum (operation number) header field in the first fragment to identify a specific function call, and Samba and all other Windows versions use the opnum field in the last fragment.

For this reason, the `dce_tcp` inspector uses a target-based approach. When you configure a `dce_tcp` inspector instance, the `policy` parameter specifies a specific implementation of the DCE/RPC TCP protocol. This in combination with the host information establishes a default target-based server policy. Optionally, you can configure additional inspectors that target other hosts and DCE/RPC TCP implementations. The DCE/RPC TCP implementation specified by the default target-based server policy applies to any host not targeted by another `dce_tcp` inspector instance.

The DCE/RPC implementations the DCE TCP inspector can target with the `policy` parameter are:

- WinXP (default)
- Win2000
- WinVista
- Win2003
- Win2008
- Win7
- Samba
- Samba-3.0.37
- Samba-3.0.22
- Samba-3.0.20

Defragmentation

The DCE TCP inspector supports reassembling fragmented data packets before sending them to the detection engine. This feature is useful in inline mode to catch exploits early in the inspection process before full defragmentation is done, or to catch exploits that take advantage of fragmentation to conceal themselves. Be aware that disabling defragmentation may result in a large number of false negatives.

DCE TCP Inspector Parameters

DCE TCP port configuration

The `binder` inspector defines the DCE TCP port configuration. For more information, see the Binder Inspector Overview.

Example:

```
[
   {
        "when": {
            "role": "any",
            "proto": "tcp",
            "service": "dcerpc",
            "ports": ""
        },
        "use": {
            "type": "dce_tcp"
        }
    }
]
```

max_frag_len

Specifies the maximum fragment length in bytes allowed for defragmentation. For processing larger fragments the inspector considers packet content to the specified size before defragmenting.

Note The value specified in this parameter must be greater than or equal to the depth to which the rules need to examine the data to ensure detection. To ensure that all data goes through detection, use the default value.

Type: integer

Valid range: `1514` to `65535`

Default value: `65535`

disable_defrag

Specifies whether to defragment fragmented DCE/RPC traffic. When this parameter is `true`, the inspector still detects anomalies and sends DCE/RPC data to the rules engine, but at the risk of missing exploits in fragmented DCE/RPC data.

Although this parameter provides the flexibility of not defragmenting traffic and can speed processing, most DCE/RPC exploits attempt to take advantage of fragmentation to hide the exploit. Enabling this parameter would bypass most known exploits, resulting in a large number of false negatives.

Type: boolean

Valid values: `true`, `false`

Default value: `false`

limit_alerts

Specifies whether to limit DCE alerts to at most one per signature per flow.

Type: boolean

Valid values: `true`, `false`

Default value: `true`

reassemble_threshold

Specifies the minimum number of bytes in the DCE/RPC desegmentation and defragmentation buffers to queue before sending a reassembled packet to the rules engine. This parameter is useful in inline mode so as to potentially catch an exploit early before full defragmentation is done.

A low value increases the likelihood of early detection but could have a negative impact on performance. You should test for performance impact if you enable this parameter.

A value of `0` disables reassembly.

Type: integer

Valid range: `0` to `65535`

Default value: `0`

policy

Specifies the Windows or Samba DCE/RPC implementation used by the targeted host or hosts on your monitored network segment.

Type: enum

Valid values: A string selected from the following list: `Win2000`, `WinXP`, `WinVista`, `Win2003`, `Win2008`, `Win7`, `Samba`, `Samba-3.0.37`, `Samba-3.0.22`, `Samba-3.0.20`

Default value: `WinXP`

DCE TCP Inspector Rules

Enable the `dce_tcp` inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 1: DCE TCP Inspector Rules

GID:SID	Rule Message
133:27	connection oriented DCE/RPC - invalid major version
133:28	connection oriented DCE/RPC - invalid minor version
133:29	connection-oriented DCE/RPC - invalid PDU type

GID:SID	Rule Message
133:30	connection-oriented DCE/RPC - fragment length less than header size
133:31	connection-oriented DCE/RPC - remaining fragment length less than size needed
133:32	connection-oriented DCE/RPC - no context items specified
133:33	connection-oriented DCE/RPC -no transfer syntaxes specified
133:34	connection-oriented DCE/RPC - fragment length on non-last fragment less than maximum negotiated fragment transmit size for client
133:35	connection-oriented DCE/RPC - fragment length greater than maximum negotiated fragment transmit size
133:36	connection-oriented DCE/RPC - alter context byte order different from bind
133:37	connection-oriented DCE/RPC - call id of non first/last fragment different from call id established for fragmented request
133:38	connection-oriented DCE/RPC - opnum of non first/last fragment different from opnum established for fragmented request
133:39	connection-oriented DCE/RPC - context id of non first/last fragment different from context id established for fragmented request

DCE Inspectors Intrusion Rule Options

dce_iface

Specifies the following comma-separated elements:

- The UUID for a service interface.
- The interface version (optional). The default setting matches any version.
- An indicator of whether a rule should match on any fragment in a request (optional). The default setting matches only the first fragment.

In the DCE/RPC protocol, a client must bind to a service before making a call to it. When a client sends a bind request to the server, it can specify one or more service interfaces to bind to. Each interface is represented by a UUID, and each interface UUID is paired with a unique index (or context ID) that future requests can use to reference the service that the client is calling. The server responds with the interface UUIDs it accepts as valid and allows the client to make requests to those services. When a client makes a request, it will specify the context ID so the server knows to what service the client is making a request.

Using the `dce_iface` rule option, a rule can ask the inspector whether the client has bound to a specific interface UUID and whether this client request is making a request to that interface. This can eliminate false positives where more than one service is bound to successfully since the inspector can correlate the bind UUID to the context id used in the request.

The `dce_iface` option requires tracking client Bind and Alter Context requests as well as server Bind Ack and Alter Context responses for connection-oriented DCE/RPC in the inspector. For each Bind and Alter Context request, the client specifies a list of interface UUIDs along with a handle (or context id) for each interface UUID that will be used during the DCE/RPC session to reference the interface. The server response indicates which interfaces it allows the client to make requests to—it either accepts or rejects the client's wish to bind to a certain interface. This tracking is required so that when a request is processed, the context id used in the request can be correlated with the interface UUID for which it is a handle.

The `dce_iface` rule option matches if:

- the specified interface UUID matches the interface UUID (as referred to by the context ID) of the DCE/RPC request

and

- the `version` argument is not supplied, or the `version` argument is supplied and matches the interface UUID of the DCE/RPC request

and

- the `any_frag` argument is supplied, or the `any_frag` argument is absent and the `dce_iface` option matches the UUID and version criteria on the initial request fragment

Examples:

```
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188;
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, <2;
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188,any_frag;
dce_iface:4b324fc8-1670-01d3-1278-5a47bf6ee188, =1, any_frag;
```

dce_iface.uuid

A DCE/RPC request can specify whether UUID numbers are represented as big endian or little endian. The representation of the interface UUID in a request is different depending on the endianness specified in the request. The `dce_rpc` inspector normalizes the UUID. This means that the UUID specification in the `dce_iface` rule option must be written the same way regardless of the endianness of the request.

For example, a little endian Bind request would represent a UUID as follows:

```
|f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6 e6 fc|
```

and a big-endian Bind request would represent the same UUID as follows:

```
|5a 7b 91 f8 ff 00 11 d0 a9 b2 00 c0 4f b6 e6 fc|
```

In Snort 3 rules using the `dce_iface` option, the UUID must be represented in a string using big endian order regardless of the endian-ness of the request:

```
5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc
```

Type: string

Syntax: `dce_iface: <UUID>;`

Valid values: Where: *`UUID`* is 32 hexadecimal digits, displayed in five groups separated by hyphens, in the form: `xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`

Examples: `dce_iface: 5a7b91f8-ff00-11d0-a9b2-00c04fb6e6fc;`

dce_iface.version

A service interface has a version associated with it. Some versions of an interface may not be vulnerable to certain exploits. For this reason you can specify one or more version numbers in the `dce_iface` option, to determine whether it is necessary to check for a particular exploit.

Type: interval

Syntax: `dce_iface: <range_operator><positive integer>;` or `dce_iface: <positive integer><range_operator><positive integer>;`

Valid values: A set of one or more positive version numbers and a `range_operator` as specified in the Table 2: Range Formats.

Examples: `dce_iface: =6;`

dce_iface.any_frag

A DCE/RPC request can be broken up into one or more fragments. Flags are set in the DCE/RPC header to indicate whether the current fragment is the first, a middle, or the last fragment of the request. Many checks for data in the DCE/RPC request are relevant only if the DCE/RPC request is a first fragment (or full request). Thus, fragments that follow the first fragment contain data deeper into the DCE/RPC request. For example, a rule that looks for data in the first five bytes of the request (for example, a length field), finds the wrong data on a fragment other than the first. The start of subsequent fragments is offset some length from the beginning of the request. This can be a source of false positives in fragmented DCE/RPC traffic.

For this reason, by default the DCE_RPC inspector matches only the initial fragment in a request. To force the inspector to examine all fragments in a request for a match, add `any_frag` to the `dce_iface` rule option. Note that a defragmented DCE/RPC request is considered a full request.

Syntax: `dec_iface: any_frag;`

Examples: `dce_iface: any_frag;`

dce_opnum

Match a DCE RPC operation number, range of operation number, or list of operation number. This option represents one or more a specific function calls that can be made to an interface. After a client has bound to a specific service interface and makes a request to it, rules need to determine what function call the client is making to the service, to check for exploits that may lie within the function call. The functions call(s) are specified as a double-quote-enclosed list of operation numbers (opnums)

Type: string

Syntax: `dce_opnum: <opnum_list>;`

Where *`opnum_list`* is one of the following:

- A single integer.
- A comma-separated list of integers.
- A range of integers specified with a hyphen separating the lowest and highest numbers in the range.
- A combination of the above.

Valid values: A list of DCE/RPC request opnums.

Examples:

```
dce_opnum: "15";
dce_opnum: "15-18";
dce_opnum: "15, 18-20";
dce_opnum: "15, 17, 20-22";
```

dce_stub_data

This option places the detection cursor (used to traverse the packet payload in rules processing) at the beginning of the DCE/RPC stub data, regardless of preceding rule options. This option matches if there is DCE/RPC stub data present.

Syntax: `dce_stub_data;`

Examples: `dce_stub_data;`

Table 2: Range Formats

Range Format	Operator	Description
operator i		
	<	Less than
	>	Greater than
	=	Equal
	≠	Not equal
	≤	Less than or equal
	≥	Greater than or equal
j operator k		
	<>	Greater than j and less than k
	<=>	Greater than or equal to j and less than or equal to k