syslog-ng Premium Edition 7.0.23

Windows Event Collector Administration Guide

syslog-ng PE Windows Event Collector Administration Guide
Updated - 14 December 2020, 02:18
Version - 7.0.23

Contents

Introduction

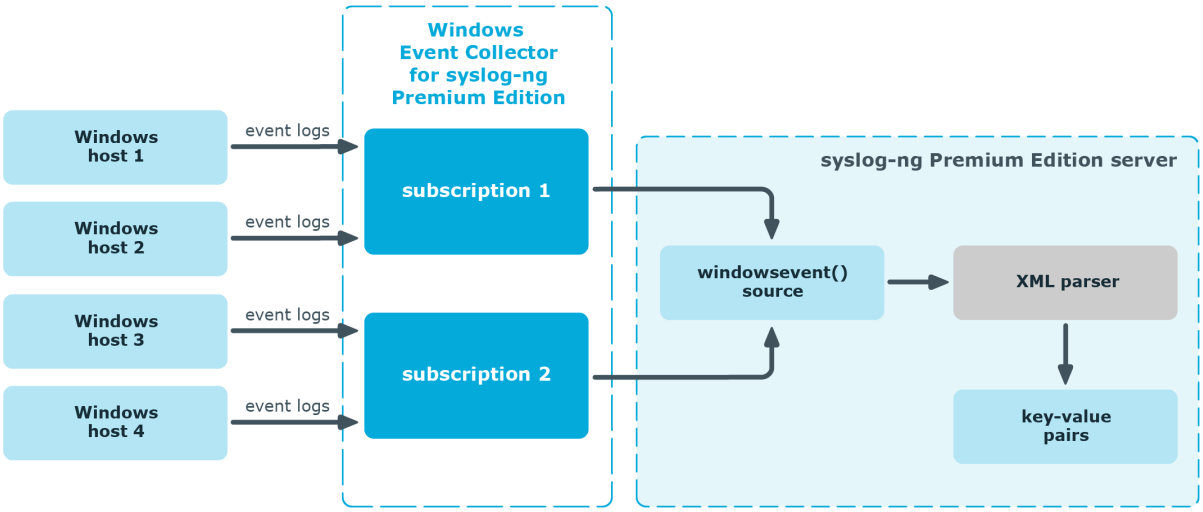
The Windows Event Collector (WEC) acts as a log collector and forwarder tool for the Microsoft Windows platform. It collects the log messages of Windows-based hosts over HTTPS (using TLS encryption and mutual authentication), and forwards them to a syslog-ng Premium Edition (syslog-ng PE) server. In Windows terminology, this tool allows you to define source-initiated push subscriptions, and have them forwarded to a syslog-ng PE server. For details on the limitations of WEC, see Limitations.

Unlike the syslog-ng Agent for Windows, the Windows Event Collector is a standalone tool that does not require installing on the Windows-based host itself. This can be an advantage when your organization's policies restrict or do not allow the installation of third-party tools.

Another difference between the Windows Event Collector tool and syslog-ng Agent for Windows is that WEC forwards logs only about Windows events, while syslog-ng Agent forwards both Windows event logs as well as files from Windows hosts to the syslog-ng PE server.

The Windows Event Collector sits between your Windows hosts and your syslog-ng Premium Edition server, accepting log messages from the remote Windows side with WinRM and feeding them to syslog-ng Premium Edition 7.0.

Figure 1: How Windows Event Collector works in syslog-ng PE 7.0



At a high level, this is how you can get Windows event logs to be forwarded to your syslog-ng Premium Edition server using the WEC tool:

1. Configure Windows event source computers.

   For details on how to configure your Windows hosts, see Configure event source computers.

2. Set up the Windows Event Collector as the server that collects and forwards event logs.

   For details on how to set up and configure the Windows Event Collector tool, see Install the Windows Event Collector, Generate SSL certificates for Windows Event Collector, and Configure Windows Event Collector.

3. The Windows Event Collector accepts incoming event log subscription requests from the Windows hosts.

4. The Windows Event Collector handshakes the event forwarding settings with the Windows hosts, for example, which events to forward.

5. The Windows Event Collector accepts the forwarded event logs, and writes the raw logs to a Unix datagram socket.

6. syslog-ng PE reads the Unix datagram socket using a source called `windowsevent()`.

   For details on how to configure your syslog-ng PE server, see Configure syslog-ng PE.

7. syslog-ng PE parses the logs into key-value pairs using the XML parser.

   For details on the XML parser, see "XML parser" in the Administration Guide.

Install the Windows Event Collector

Prerequisites

- syslog-ng PE version 7.0.6 or newer
- glibc version 2.12 or newer

  glibc version 2.12 is available on all platforms supported by syslog-ng Premium Edition 7.0. However, in the case of Red Hat Enterprise Linux, an upgrade to version 6.9 or newer is required.

The Windows Event Collector is bundled into the syslog-ng PE installers from version 7.0.6 onward. A SysV init script and a systemd service file are provided and installed automatically, so by installing syslog-ng PE, you also install WEC. However, `syslog-ng-wec` is not registered to start at boot.

To install the Windows Event Collector

1. To start `syslog-ng-wec` at boot, register the init script using the following commands:
   - *On systemd-based systems:* `systemctl enable syslog-ng-wec`
   - *On SysV-based systems:* `chkconfig` or `update-rc.d`

   For details on how to start `syslog-ng-wec` manually, see Start/stop Windows Event Collector.

3

Generate SSL certificates for Windows Event Collector

When the Windows-based host and the Windows Event Collector start communicating for the first time, they authenticate each other by exchanging and verifying each other's certificates. The process begins with the Windows host requesting and verifying the WEC tool's certificates. After successful verification, the Windows host sends its own certificates for verification to WEC.

NOTE: If the Windows host fails to authenticate the WEC tool's certificates for some reason, check the Windows event logs for details.

For details on which event logs to look at, see Troubleshoot Windows Event Collector.

The example described in this section uses OpenSSL for certificate generation. Note, however, that you can generate certificates using the Windows Public Key Infrastructure (PKI).

CAUTION:

The examples in this section illustrate the creation of certificates with a default value of 365 days for expiration. However, most deployments remain operational for several years, so refreshing your certificates every 365 days is unnecessary. To avoid refreshing your certificates every year, One Identity recommends that you setup your host certificates with an expiration time longer than the default value of 365 days (as seen in the examples).

To generate the SSL certificates for WEC, complete the following steps:

To generate SSL certificates for Windows Event Collector

1. Create two certificate template files for both the server and the client(s).

   NOTE: The templates shown here are examples only. Not all elements of the example `*opts.cnf` files are mandatory, for example, you do not need to define two DNS instances.

   The contents of `server-certopts.cnf`:

```
[req]
default_bits = 4096
default_md = sha256
req_extensions = req_ext
keyUsage = keyEncipherment,dataEncipherment
basicConstraints = CA:FALSE
distinguished_name = dn

[ req_ext ]
subjectAltName = @alt_names
extendedKeyUsage = serverAuth,clientAuth

[ alt_names ]
DNS.1 = <1st DNS hostname of server (preferably FQDN)>
...
DNS.<N> = <Nth DNS hostname of server (preferably FQDN)>
IP.1 = <1st IP of server>
...
IP.<N> = <Nth IP of server>

[dn]
```

For example:

```
[req]
default_bits = 4096
default_md = sha256
req_extensions = req_ext
keyUsage = keyEncipherment,dataEncipherment
basicConstraints = CA:FALSE
distinguished_name = dn

[ req_ext ]
subjectAltName = @alt_names
extendedKeyUsage = serverAuth,clientAuth

[ alt_names ]
DNS.1 = windowseventcollector.widgits
DNS.2 = wec.widgits
IP.1 = 10.64.10.2

[dn]
```

The contents of `client-certopts.cnf`:

```
[req]
default_bits = 4096
default_md = sha256
req_extensions = req_ext
keyUsage = keyEncipherment,dataEncipherment
basicConstraints = CA:FALSE
distinguished_name = dn

[ req_ext ]
subjectAltName = @alt_names
extendedKeyUsage = serverAuth,clientAuth

[ alt_names ]
DNS.1 = <1st DNS hostname of client (preferably FQDN)>
...
DNS.<N> = <Nth DNS hostname of client (preferably FQDN)>
IP.1 = <1st IP of client>
...
IP.<N> = <Nth IP of client>

[dn]
```

For example:

```
[req]
default_bits = 4096
default_md = sha256
req_extensions = req_ext
keyUsage = keyEncipherment,dataEncipherment
basicConstraints = CA:FALSE
distinguished_name = dn

[ req_ext ]
subjectAltName = @alt_names
extendedKeyUsage = serverAuth,clientAuth

[ alt_names ]
DNS.1 = windowsclient01.widgits
DNS.2 = client01.widgits
IP.1 = 10.64.10.11

[dn]
```

2. Generate the certificate authority (CA):

```
$ openssl genrsa -out ca.key 4096

$ openssl req -x509 -new -nodes -key ca.key -days 3650 -out ca.crt -subj
'<subject name for CA cert (must be formatted as
/type0=value0/type1=value1/type2=..., characters may be escaped by \
(backslash), no spaces are skipped)>'
```

For example:

```
$ openssl genrsa -out ca.key 4096

$ openssl req -x509 -new -nodes -key ca.key -days 3650 -out ca.crt -subj
'/C=AU/ST=Victoria/L=Melbourne/O=Internet Widgits Pty
Ltd/OU=Operations/CN=Operations Root CA'
```

Place a copy of the `ca.crt` file in a directory of your choice. Take a note of the directory because you need to reference it in the `cadir` option of the WEC configuration file.

3. Save the thumbprint of the CA:

```
$ openssl x509 -in ca.crt -fingerprint -sha1 -noout | sed -e 's/\://g'
```

You will need the fingerprint to configure the event source computers that send log messages to Windows Event Collector. For details, see Configure event source computers.

4. Create the server certificate:

> NOTE: The Common Name must be the FQDN (or IP address) of the Windows Event Collector server.

```
$ openssl req -new -newkey rsa:4096 -nodes -out server.csr -keyout server.key
-subj '<subject name for server cert (must be formatted as
/type0=value0/type1=value1/type2=..., characters may be escaped by \
(backslash), no spaces are skipped)>'

$ openssl x509 -req -in server.csr -out server.crt -CA ca.crt -CAkey ca.key -
CAcreateserial -extfile server-certopts.cnf -extensions req_ext -days 365
```

For example:

```
$ openssl req -new -newkey rsa:4096 -nodes -out server.csr -keyout server.key
-subj '/C=AU/ST=Victoria/L=Melbourne/O=Internet Widgits Pty
Ltd/OU=Operations/CN=windowseventcollector.widgits'

$ openssl x509 -req -in server.csr -out server.crt -CA ca.crt -CAkey ca.key -
CAcreateserial -extfile server-certopts.cnf -extensions req_ext -days 365
```

5. Create client(s') certificates:

   NOTE: The Common Name must be the FQDN (or IP address) of the client.

   ```
   $ openssl req -new -newkey rsa:4096 -nodes -out client.csr -keyout client.key
   -subj '<subject name for client cert (must be formatted as
   /type0=value0/type1=value1/type2=..., characters may be escaped by \
   (backslash), no spaces are skipped)>'

   $ openssl x509 -req -in client.csr -out client.crt -CA ca.crt -CAkey ca.key -
   CAcreateserial -extfile client-certopts.cnf -extensions req_ext -days 365
   ```

   For example:

   ```
   $ openssl req -new -newkey rsa:4096 -nodes -out client.csr -keyout client.key
   -subj '/C=AU/ST=Victoria/L=Melbourne/O=Internet Widgits Pty
   Ltd/OU=Operations/CN=windowsclient01.widgits'

   $ openssl x509 -req -in client.csr -out client.crt -CA ca.crt -CAkey ca.key -
   CAcreateserial -extfile client-certopts.cnf -extensions req_ext -days 365
   ```

6. Export the client(s') certificate(s) to the format recognized by the Windows Certificate Manager tool.

   ```
   $ openssl pkcs12 -export  -inkey client.key -in client.crt -certfile ca.crt -
   out client.p12
   ```

Configure event source computers

Prerequisites

- Microsoft Windows 7 or newer, up to Windows Server 2019

When collecting event logs from Windows hosts, the Windows clients sending logs act as the event source computers. The WEC tool collects and forwards messages from the standard Windows eventlog containers.

There is no restriction on the number of Windows hosts that can connect to the Windows Event Collector.

To configure your event sources, complete the following steps.

To configure event source computers

1. Open the **Microsoft Management Console** (`mmc.exe`), select **File > Add/Remove Snap-ins**, and add the **Certificates** snap-in.
2. Select **Computer Account**.
3. Right-click the **Personal** node, and select **All Tasks > Import**.
4. Find and select the client certificate (`client*.p12`) and import this file.
5. The PKCS #12 archive contains the CA certificate as well. Move the CA certificate to the **Trusted Root Certification Authorities** node after the import.

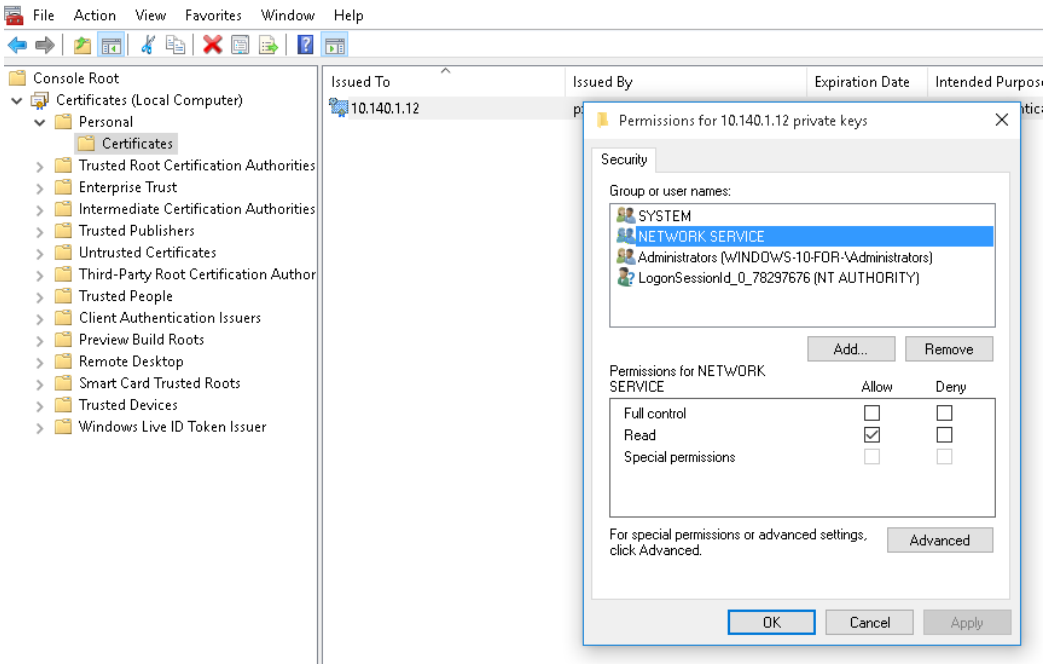
   NOTE: Make sure that you only move the CA certificate and not the client certificate.
6. Give NetworkService access to the private key file of the client authentication certificate:

   NOTE: Make sure that you modify the access rights of only the private key file of the client certificate and not the CA certificate.

   a. In certmgr, right-click the client certificate, select **All Tasks > Manage Private Keys...**.

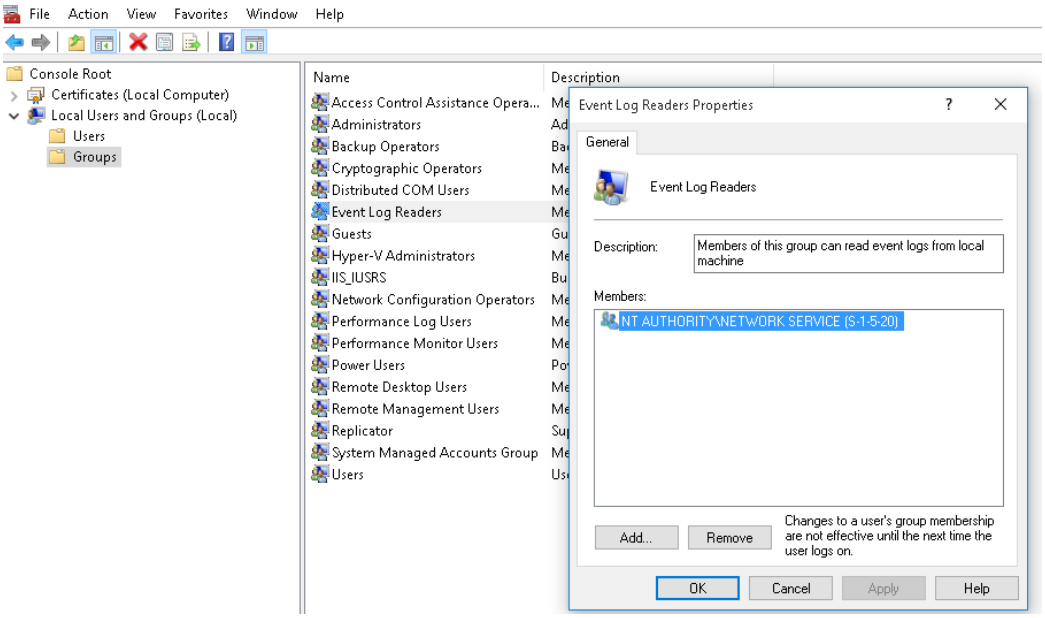
   b. Add read permission to "NETWORK SERVICE".

Figure 2: Adding read permission to "NETWORK SERVICE"



7. To forward security logs:
    a. In CompMgmt.msc, under **Local Users and Groups**, click **Groups > Event Log Readers** to open **Event Log Readers Properties**.
    b. Add the "NETWORK SERVICE" account to the **Event Log Readers** group.

Figure 3: Adding the "Network Service" account to the Event Log Readers group.



c. Reboot the client computer.

8. Run the following commands from an elevated privilege command prompt:

```
winrm qc -q
winrm set winrm/config/client/auth @{Certificate="true"}
```

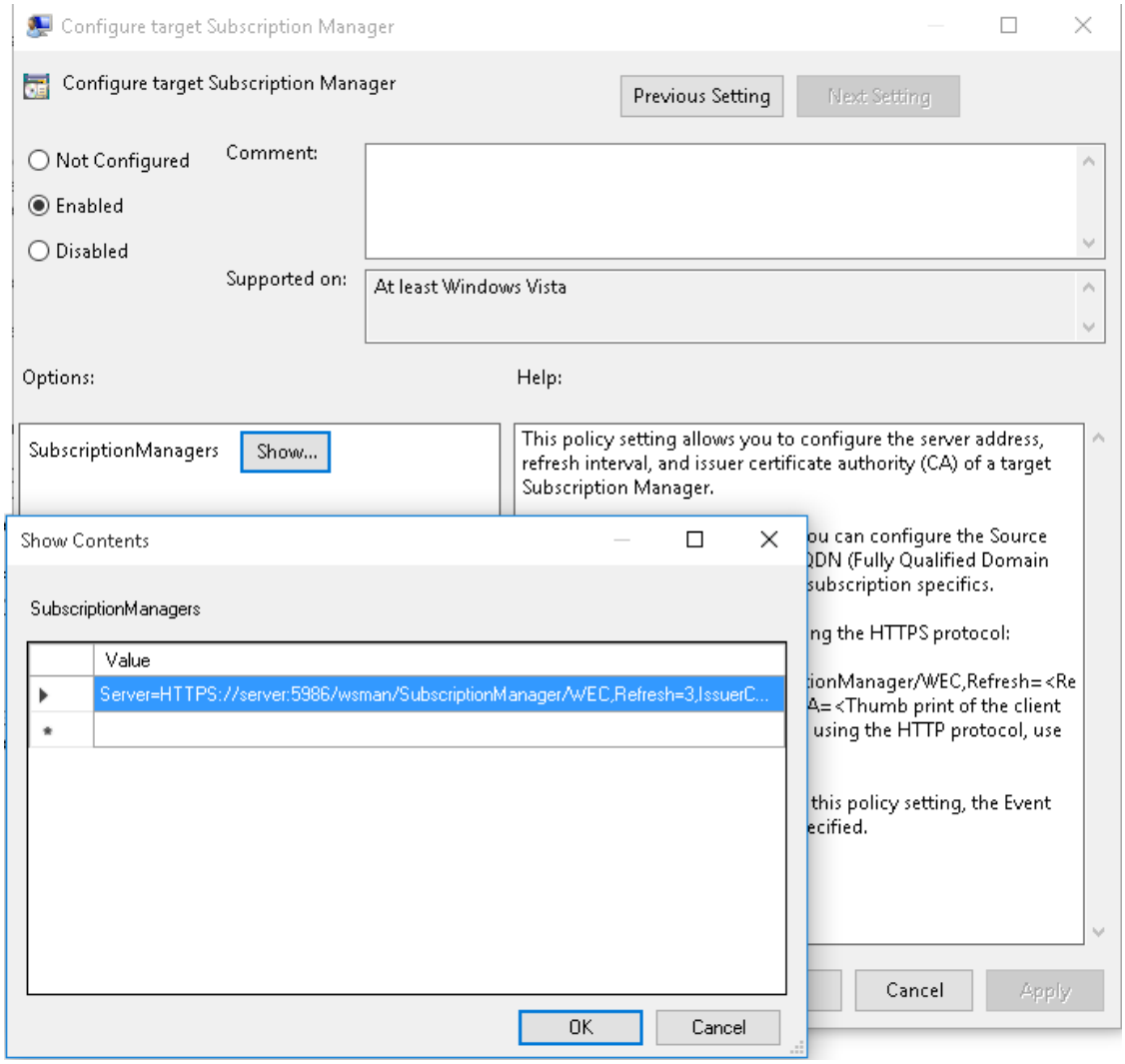
9. Open gpedit.msc.
10. Under the **Computer Configuration** node, expand the **Administrative Templates** node, then expand the **Windows Components** node, and then select the **Event Forwarding** node.
11. Select the **SubscriptionManagers** setting and enable it. Click the **Show** button to add a subscription (Generate SSL certificates for Windows Event Collector):

```
Server=https://<FQDN of the
collector>:5986/wsman/SubscriptionManager/WEC,Refresh=<Refresh interval in
seconds>,IssuerCA=<Thumbprint of the root CA>
```

For example:

```
Server=HTTPS://wec.balabit:5986/wsman/SubscriptionManager/WEC,Refresh=60,Issue
rCA=A814E609311FD3A89FFD0297974524E4F2D2BA9D
```

Figure 4: Adding the subscription in SubscriptionManagers



NOTE: If you wish to set up multiple subscriptions because you want to forward Windows events to multiple event collectors (such as WEC), then you can do that here.

Configure Windows Event Collector

Once you have configured your event source computer(s), the next step is to configure your event collector, in this case, the Windows Event Collector for syslog-ng PE.

NOTE: The configuration file of WEC is YAML based. Note that YAML uses spaces, not tabs, for indentation.

To configure WEC, use the following options.

For an example `wec.yaml` file, see WEC configuration example.

server

Type:	string
Default:	N/A

Description: The hostname, IP address, or FQDN of the server where WEC is running. It must match the Common Name of the SSL certificate.

port

Type:	integer
Default:	5986

Description: The port where the server running WEC is listening.

keyfile

Type:	string
Default:	N/A

Description: The path to the file that contains the unencrypted private key of the server running WEC. The file is in PEM format.

certfile

Type:	string
Default:	N/A

Description: The path to the file that contains the X.509 certificate of the server running WEC. The file is in PEM format.

cadir

Type:	string
Default:	N/A

Description: The path to the directory that contains the trusted CA certificates in PEM format.

log

Type:	map
Default:	N/A

Description: The options to specify how to handle the internal logs of WEC:

- level
- file

WEC sends internal log messages to stderr. You can also optionally specify a file to send logs to (in parallel with stderr). If you are using a systemd platform and start WEC using `systemctl`, then stderr is redirected to systemd-journal, and this is where you will find the internal logs of WEC.

level

Type:	debug|info
Default:	info

Description: The application log level of WEC.

Possible values are:

- `debug`: Information with the most details, useful when debugging WEC and diagnosing issues.
- `info`: Basic information about the initialization of WEC. Following initialization, no information is displayed on screen, unless an issue occurs.

file

Type:	string
Default:	N/A

Description: The path to the file where WEC should write internal log messages. The log file is automatically created by syslog-ng PE.

You can send this file to syslog-ng using a `file()` source.

eventdestination

Type:	map
Default:	N/A

Description: The options to specify how to store the event logs that are forwarded to WEC:

- file
- unixdatagram
- queuesize

file

Type:	string
Default:	N/A

Description: The path to the file where WEC should write the events received from the Windows host(s). Use this option for debug purposes only, when you wish to check what WEC is sending to syslog-ng PE.

It is possible to log both to a file and a Unix datagram socket in parallel.

unixdatagram

Type:	string
Default:	N/A

Description: The path to the Unix datagram socket receiving the Windows events. WEC writes the received events to the Unix datagram socket specified here. The listening socket is automatically created by syslog-ng PE.

It is possible to log both to a file and a Unix datagram socket in parallel.

queuesize

Type:	integer

Default:	10000

Description: The number of events that the destination memory queue can store. Note that the main purpose of `queuesize` is to speed up the writing of data into memory and smooth out peaks.

It is recommened to use the default value for this option.

For more information about flow control, see Flow control.

subscriptions

Type:	map
Default:	N/A

Description: The options related to the subscriptions you have set up on WEC:

- name
- computers
- contentformat
- heartbeats
- connectionretry
- batchsizelimit
- batchtimeoutlimit
- queries
- readexistingevents

NOTE: You can set up multiple subscriptions to events coming from the same Windows host. If an event matches more than one subscription, the event log comes in to WEC as many times as there is a match.

name

Type:	string
Default:	N/A

Description: The unique name of the subscription in WEC.

computers

Type:	list of strings
Default:	N/A

Description: A list of strings that specifies the DNS names of the non-domain computers that are allowed to initiate subscriptions. Specifies the Windows hosts from which you want WEC to receive event logs.

The names can be specified using the `*` and `?` wildcards, for example, `"*.mydomain.com"` or `"*"`.

contentformat

Type:	Events|RenderedText
Default:	N/A

Description: A value that specifies the format of the returned events.

Possible values are:

- `RenderedText`: Events are returned with the localized strings (such as event description strings) attached to the events
- `Events`: Events are returned without any of the localized strings

One Identity recommends setting this option to `RenderedText`.

heartbeats

Type:	integer
Default:	N/A

Description: A value that specifies the heartbeat interval for the subscription in seconds.

connectionretry

Type:	integer
Default:	N/A

*Description:*WEC attempts to reconnect to the Windows host(s) at the specified interval of time in seconds.

batchsizelimit

Type:	integer
Default:	0 (meaning that there is no limit)

Description: Specifies the maximum number of items for batched delivery in the event subscription.

Set this value to `1` if you wish to perform tests or debugging.

> NOTE: This option is not enforced on the Windows host side. Windows is handling this value only as a recommendation. The only exception is the value `1`.

batchtimeoutlimit

Type:	integer
Default:	N/A

Description: Specifies the maximum latency allowed in delivering a batch of events (in seconds).

NOTE: This option is not enforced on the Windows host side. Windows is handling this value only as a recommendation.

queries

Type:	string
Default:	N/A

Description: Specifies the query string for the subscription.

You can:

- type this value manually, or
- copy this value from the **XML** tab of the **Create Custom View** pop-up window in Windows

For examples of queries, check the following Windows blog posts:

- Microsoft TechNet blog post *Advanced XMl filtering in the Windows Event Viewer*
- Microsoft Developer Network article *Wecutil.exe*

readexistingevents

Type:	true|false
Default:	false

Description: When the value is `true`, all existing events are read from the subscription event source if the subscription in question has not been read yet. When the value is `false`, only future (arriving) events are delivered. If the subscription has a state in the persist file, this option will have no effect.

Configure syslog-ng PE

Prerequisites

syslog-ng Premium Edition (syslog-ng PE) version 7.0.6 or newer.

To enable syslog-ng PE to read and accept Windows events, you need to configure a source called `windowsevent()`.

To configure syslog-ng PE

1. Ensure that the default `5986` TCP port is accessible from WEC, as it is the Windows Event Collector that will initiate the event forwarding subscription toward the syslog-ng PE server.
2. Configure the `windowsevent()` source.

   ```
   source  s_wec  {
          windowsevent();
    };
   ```

   The `windowsevent()` source takes the following options:

   - `unix-domain-socket()`: The path to the Unix domain socket to read messages from.

     The default value is `/opt/syslog-ng/var/run/wec.sock`.
   - `prefix()`: The prefix that you wish to append to the key-value pairs.

     The default value is `".windowsevent."`.

     If you want to send Windows event logs to SDATA, then set `prefix(".SDATA.")`. This can be useful, for example, when you forward Windows event logs to a syslog-ng Store Box.

   For more information on the `windowsevent()` source, see "windowsevent: Collecting Windows event logs" in the Administration Guide.
3. Define a complete log path in `syslog-ng.conf` to enable the `windowsevent()` source, `s_wec`. Otherwise, the WEC process will not run (connection refused).

   For example:

```
source  s_wec  {
       windowsevent();
};

log  {
       source(s_wec);
       destination  {
              file("/var/log/example.log"
                     template("$(format-json  --scope  dot-nv-pairs)\n")
              );
       };
};
```

Start/stop Windows Event Collector

To start and stop the Windows Event Collector tool manually, use the following commands:

systemd service for systemd-based systems

- Start WEC: `systemctl start syslog-ng-wec`
- Stop WEC: `systemctl stop syslog-ng-wec`

SysV init for SysV-based systems

- Start WEC: `/etc/init.d/syslog-ng-wec start`
- Stop WEC: `/etc/init.d/syslog-ng-wec stop`

To start WEC in the foreground, execute:

```
wec -c /path/to/wec.yaml
```

Message format in Windows Event Collector for syslog-ng PE

The Windows Event Collector for syslog-ng Premium Edition (syslog-ng PE) is supported for Windows 7 or newer platforms. Starting with Windows 7, event logging is XML-based, meaning that event log messages reach WEC in XML format. When these are forwarded to the syslog-ng PE server, syslog-ng PE parses them into key-value pairs using the XML parser.

Once event log data is available in syslog-ng PE, you have the flexibility to modify and format data any way you want, using macros and rewrite rules.

Note that while event log data as processed by the WEC tool may differ from the data collected and made available by the syslog-ng Agent for Windows, the Windows Event Collector tool provides you with greater freedom and flexibility when it comes to manipulating your raw data.

9

Flow control

The Windows Event Collector tool applies flow control to minimize event log loss.

WEC regularly (in every second) polls the datagram socket that will receive the Windows events to check whether it exists already. Once the socket has been created (syslog-ng PE has started up), WEC connects to the socket and accepts the incoming connections from the Windows hosts. If the datagram socket does not exist, WEC refuses the incoming connections.

If the socket exists (syslog-ng PE is running) but syslog-ng PE does not read the Unix datagram socket, WEC fills up the kernel buffer of the datagram socket and stores `queuesize` amounts of log messages in the memory. When all buffers are full, WEC stops reading messages from the HTTP connections to prevent message loss.

The buffer size of a datagram socket is determined by certain Linux kernel parameters: the value of `rmem_*` (max/default) and the count of `net.unix.max_dgram_qlen`.

Reliability

WEC flags a message as delivered once it has put the message in the socket buffer. If syslog-ng PE does not read the socket for some reason (for example, because it is still flow-controlled) and syslog-ng PE is stopped, the contents of this socket (that is, the messages that are in the kernel buffer, unread by syslog-ng PE) will be lost.

This is why in cases when a restart is necessary, it is recommended to stop the Windows Event Collector and syslog-ng PE in the following order:

1. Windows Event Collector
2. syslog-ng PE

While it is not guaranteed that syslog-ng PE has read all sockets by the time you stop it, at least you can maximize the chances that it has.

10

Performance

Performance is dependent on the number of event log messages that the Windows hosts send to WEC and the capabilities of the XML parser.

Our performance measurements indicate that syslog-ng Premium Edition (syslog-ng PE)'s XML parser is capable of parsing 15,000-20,000 events/second. The exact capacity of the XML parser depends on the complexity of the Windows log messages, as well as the performance of the hardware that syslog-ng PE and WEC are running on. When the limit of 15,000-20,000 events/seconds is reached, a workaround is recommended.

As the value set in the `batchsizelimit` parameter is treated only as a recommendation by the Windows hosts, there is no direct way to control the amount of messages arriving from the event source computers.

A possible workaround is to launch multiple WEC servers and create multiple `windowsevent()` sources in syslog-ng PE. That way, you can distribute your Windows hosts across multiple WEC and syslog-ng PE servers, decreasing the load on individual servers.

To run multiple WEC services per syslog-ng PE service, you need to create your own init script. This is because the init script that comes with WEC enables you to run only a single WEC service per syslog-ng PE service.

11

Limitations

The Windows Event Collector for syslog-ng Premium Edition (syslog-ng PE) currently has the following limitations:

- Only source-initiated push subscriptions are supported (Windows hosts connect to the WEC server).

  An advantage of this, however, is that this requires less firewall rules.
- Only HTTPS and SSL certificate based authentication are supported. Kerberos authentication is not supported at the moment.
- The compression of events is not supported.
- The batchsizelimit and batchtimeoutlimit options are not enforced on the Windows host side: Windows is handling these values only as a recommendation.
- On Windows 7 and Windows 2008 platforms, there is a known issue. After several reconnects (if WEC is restarted quickly), the remote sender can stop forwarding the logs for a certain period of time. In this case, restarting the Windows RM service can help.

  This issue can also occur between two Windows machines. It has been reported to Microsoft and is awaiting resolution.

12

Troubleshoot Windows Event Collector

When you experience issues while using WEC, run WEC in debug mode to get detailed log messages.

1. Set the log level to `debug`:

   ```
   log:
     level: "debug"
   ```

2. Start WEC.

   At every refresh interval, the following debug messages should be displayed:

   ```
   DEBUG   subscriptionEndpoint    {"clientAddress": "..."}
   DEBUG   actionHandler   {"messageID": "...", "action":
   "http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate"}
   DEBUG   enumerate
   ```

   This means that the client has connected and requested the subscription list.

3. If you cannot see these messages within the refresh interval, you should check the following channels in the client's event viewer:
   - `Applications and Services Logs\Microsoft\Windows\Eventlog-ForwardingPlugin`
   - `Applications and Services Logs\Microsoft\Windows\Windows Remote Management`

Some common error codes and their explanations:

- *5004*: A channel specified in the query XML does not exist or cannot be read on the Windows client. This can be caused by the "Network Service" not having permission to read the security log.

  Add the "Network Service" account to the **Event Log Readers** group, and restart the computer for changes to take effect.
- *15008*: The query XML of the subscription is invalid.
- *995 (HTTP error 12186)*: The "Network Service" does not have permission to read the client certificate.

- *HTTP error 403*: If everything is set correctly, then it might be possible that a proxy is set and the forwarder tries to connect to the proxy instead of WEC.

  TIP:

  Sometimes proxy settings are not displayed in any GUI window. Check them using `netsh winhttp show proxy`. To reset proxy settings, use `netsh winhttp reset proxy`.

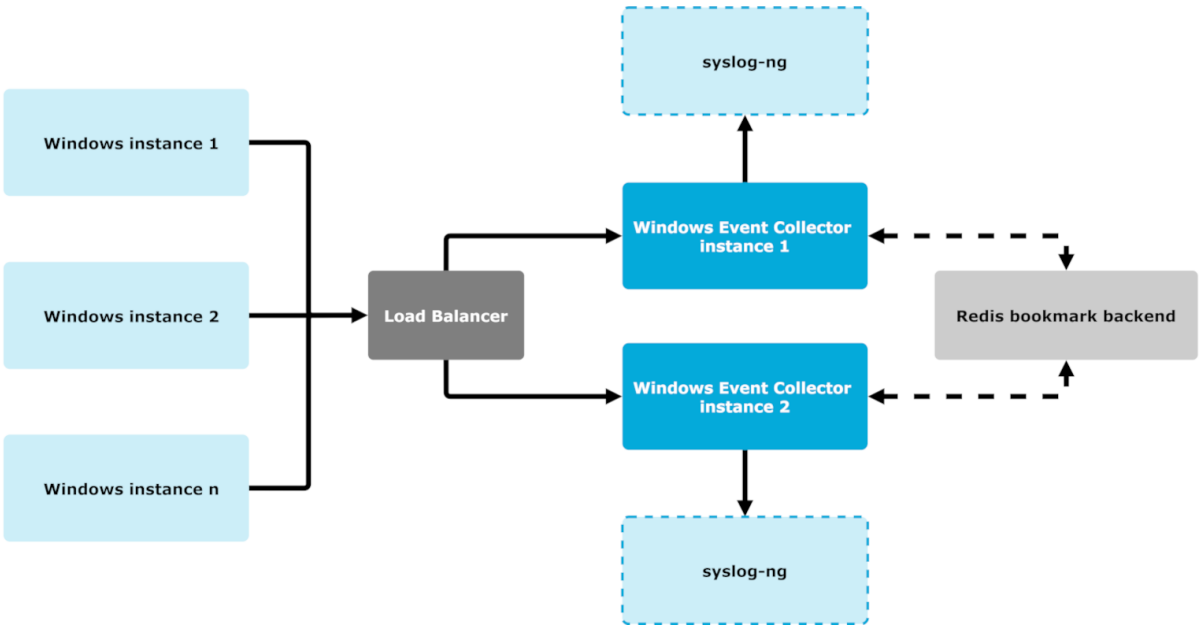
WEC configuration example

```
server: "wec.mydomain"
port: 5986
keyfile: "/opt/syslog-ng/etc/server.key"
certfile: "/opt/syslog-ng/etc/server.crt"
cadir: "/opt/syslog-ng/etc/cadir"

log:
  level: "info"
  file: "/opt/syslog-ng/var/wec.log"

eventdestination:
  unixdatagram: "/opt/syslog-ng/var/run/wec.sock"

subscriptions:
  - name: "ExampleDefaultSubscription"
    computers:
      - "windowsdc.mydomain.com"
      - "*.trusteddomain.com"

    contentformat: "RenderedText"
    heartbeats: 900.000
    connectionretry: 60.0
    batchtimeoutlimit: 900.000
    queries: |
      <QueryList>
        <Query Id="0">
          <Select Path="Application">*</Select>
          <Select Path="Security">*</Select>
          <Select Path="System">*</Select>
        </Query>
      </QueryList>
```

WEC clustering in syslog-ng PE

From version 7.0.23, syslog-ng Premium Edition (syslog-ng PE) supports distributing and monitoring Windows Event Forwarder connections with a TCP-level load balancer across multiple Windows Event Collector (WEC) instances (in short, WEC clustering). While using WEC clustering in syslog-ng PE, the configuration also uses a Redis key-value database to share WEC instances' state.

Figure 5: A schematic figure of WEC clustering



NOTE: There is no direct connection between WEC and Windows instances. Instead, the Windows machines are connected to load balancers, and the load balancers are connected to WEC instances. This also means that mutual authentication is performed between Windows hosts and load balancers, and then there is also a mutual authentication performed between load balancers and WEC instances.

Limitations

CAUTION:

Hazard of data loss!

In syslog-ng Premium Edition (syslog-ng PE) version 7.0.23, Redis Cluster is not supported. If you attempt to set up your configuration to enable the Redis Cluster feature, your WEC cluster will not function properly.

NOTE: The timers described the Timers on the WEC side section are not user configurable in version 7.0.23 of syslog-ng PE.

For more detailed information about WEC clustering, see the following sections:

The working mechanism of Windows Event Collector (WEC) clustering

This section describes the principles of the working mechanism behind Windows Event Collector (WEC) clustering.

If you are new to the concept of Windows Event Collectors, see the Introduction section of this guide, or the sections following the introduction for further details.

How does WEC clustering work?

Windows Event Forwarders are connected to load balancers, and the load balancers are connected to Windows Event Collector (WEC) instances. When a WEC instance is disconnected from a load balancer, the load balancer detects the disconnected WEC instance, and forwards events to a different WEC instance. The new WEC instance can continue event requests from the last position saved by the disconnected WEC. To solve the redirection problem, you must have WEC instances that can be configured to share their states. Because it is possible for WEC instances to store their state in a Redis key-value database, syslog-ng Premium Edition (syslog-ng PE) utilizes this Redis feature and shares WEC instances' state across such a Redis key-value database when performing WEC clustering.

> NOTE: The WEC clustering configuration operation is backward compatible and file states remain at their default as well. Converting file-based states to Redis states is also supported.

States in WEC clustering (file-based states and Redis states)

In Windows Event Forwarders, the subscription's status at a given time is recorded in a file (or a Redis key-value database, depending on your use case) that is commonly called a state. Accordingly, while discussing Windows Event Collector (WEC) clustering , these sections mention two types of states: file-based states and Redis states.

> NOTE: File-based states are local states and are not synchronized between multiple WEC instances (as opposed to shared states) File-based states can only be used locally, by a single WEC instance at a time.

The three data types within file-based states

States that are recorded in a file (that is, file-based states) consist of three data types:

- An event source (also called a host), where the events are received.
- A subscription ID.
- A bookmark (also called a position).

In case of file-based states, this information is stored within files (in the `stateDir` directory). Each file stored in the `stateDir` directory contains subscription IDs and positions to the given host, one file each.

Example: data structure within the stateDir directory

```
stateDir\win1: subscription_1 - position_1 subscription_2 - position_2 ...
subscription_n - position_n
```

NOTE: Real state files store these data in a different format. This format is only true for the information structure within the `stateDir` directory.

Mapping the file-based state structure to Redis states

If the user wants to use Redis states instead of file-based states while WEC clustering, the information has to be mapped to HSET data structures, a data structure native to Redis. Event sources are capable of identifying HSET data structures in file-based states' data structure contents, and the same state structure of subscription IDs and positions to the given host is mapped to Redis states as follows:

- In Redis, the Redis connections are mapped to the `stateDir` directory of the file-based state.
- In Redis, the HSETs are mapped to the file name of the event source (or host) of the file-based state.
- In Redis, the HKEYs within the HSETs are mapped to subscription IDs of the file-based state.
- In Redis, the HVALUEs within the HSETs are mapped to bookmarks (positions) of the file-based state.

What happens if the WEC instance is disconnected from a load balancer?

When a WEC instance is disconnected from a load balancer, the subscriptions connected to this instance are redirected to a different available WEC instance. When the new WEC instance receives the subscription, the WEC instance looks the subscription up in the Redis state, then populates the position from Redis. After this, the WEC instance continues processing requests from the last position loaded from the Redis backend. For each event (batch), the WEC instance writes bookmarks to Redis.

NOTE: Redis manages concurrent read or write operations.

What happens if Redis becomes unavailable for a WEC?

If for some reason, Redis becomes unavailable for a WEC istance, the WEC instance performs the following, in this order:

1. The WEC instance stops the listener.
2. The WEC instance stops all TCP connections established between the load balancer and the WEC instance.
3. The WEC instance stops sending messages to syslog-ng Premium Edition (syslog-ng PE).
4. The WEC instance starts to ping Redis.

NOTE: The load balancer is connected to the listener, so when the WEC instance is disconnected, the load balancer redirects the Windows event source to a different WEC instance. When Redis becomes available again, the load balancer detects it, and starts forwarding the events to the original WEC instance.

Converting file-based states to Redis states

This section describes the process of converting file-based states to Redis states, and the related potential limitations and issues.

For more information about file-based states and Redis states, see States in WEC clustering (file-based states and Redis states).

Conversion steps and limitations

During the conversion process, you move your existing file-based states into Redis states. The WEC binary supports moving the file-based states with a dedicated command line option (`-i`):

```
root@ubuntu1:/opt/syslog-ng/var/wecstate# /opt/syslog-ng/libexec/wec -h Usage of
/opt/syslog-ng/libexec/wec:
-c string
Configuration file name (default "/opt/syslog-ng/etc/wec.yaml")
-i Initializing shared state from stateDir
-s string
Persistent state directory (default "/opt/syslog-ng/var/wecstate")
-v WEC version
```

To convert file-based states to Redis states

1. Load your existing file-based states from your `stateDir` directory.
2. Connect to Redis server.

   NOTE: If your Redis server is not available, the WEC will not start at all, and the conversion process stops.
3. Save your previously loaded file-based states into a Redis HSET.
4. Move the files containing your WEC states to your `stateDir/.converted` folder.

CAUTION: If moving the WEC states to your `stateDir/.converted` folder is unsuccessful, your file-based state will contain out-of-date data. If the final conversion step is unsuccessful, and you have your `-i` option enabled, the out-of-date data in your file-based states overwrite the data in your Redis states after your next WEC restart (which results in message duplication). To avoid this, remove the `-i` command line option before you restart your WEC following your state conversion.

NOTE: If for some reason, your WEC cluster configuration does not work as expected, your previous file-base states will not be available if you delete them. Instead, One Identity recommends moving the files to the folder, where they will be available for recovery if needed.

NOTE: State conversion has two possible results:

- successful
- unsuccessful

If any of the state conversion steps is unsuccessful, the WEC instance stops with an error. As a result, even if the file state is successfully saved to Redis, but moving the files is unsuccessful, the WEC instance stops and prints an error message:

```
Failed to move state files, remove it manually
```

This is an expected behavior, developed to avoid unwanted, huge message duplication and related issues. Instead of message duplication, if the file state is converted, but moving the state files is unsuccessful for some reason, you can re-initialize Redis with the state files that contain old bookmarks.

An example use case for WEC clustering

This section describes an example use case for Windows Event Collector (WEC) clustering .

For more detailed information about the working mechanism of WEC clustering, see The working mechanism of Windows Event Collector (WEC) clustering.

Limitations

CAUTION:

Hazard of data loss!

In syslog-ng Premium Edition (syslog-ng PE) version 7.0.23, Redis Cluster is not supported. If you attempt to set up your configuration to enable the Redis Cluster feature, your WEC cluster will not function properly.

Configuration

In the example use case for WEC clustering, the following configuration is used:

```
bookmarks:
backend: redis
uri: 192.168.0.14:6379
password: pwd #optional
```

NOTE: In the example use case, multiple WEC instances are running behind the load balancer. Load balancing in this configuration will only work with multiple WEC instances if you have the same Redis backend configured on all WEC instances, with the same subscription used by all of the WEC instances. Otherwise, you will encounter message duplication.

Required components for a functional WEC cluster configuration

For a functional WEC cluster configuration, the required components are the following:

- WEC instances, with the following considerations:
  - The WEC instances must have a shared state (on the Redis backend).
  - The WEC instances must be configured to use the same subscription.

    The `subscriptions` part of the configuration should be the same in every WEC instance, for example:

    ```
    subscriptions:
     - batchsizelimit: 1
     batchtimeoutlimit: 1.0
     computers:
     - '*'
     connectionretry: 1.0
     contentformat: RenderedText
     heartbeats: 3.0
     name: wec-1-subscription
     queries: "<QueryList>\n   <Query Id=\"0\">\n      <Select
    Path=\"Application\">*</Select>\n\
     \   </Query>\n</QueryList>\n"readexistingevents: false
     * wec is connected to LoadBalancer (server and port is set to
    Load Balancer)
    ```

- Windows instances that forward requests to load balancers.
- An appropriately installed, set up, maintained, and monitored TCP level load balancer.

  The customer has the following responsibilities:
  - Installing the TCP level load balancer.
  - Setting up the TCP level load balancer.
  - Maintaining the TCP level load balancer.
  - Monitoring the TCP level load balancer.

- Configuring the SSL certificates appropriately.

  NOTE: When configuring your SSL certificates while WEC clustering, consider that the load balancer functions as a proxy in your configuration.

  For more information about configuring SSL certificates for Windows Event Collectors, see Generate SSL certificates for Windows Event Collector.

- An appropriately set up, configured, maintained, and monitored Redis server.

  The customer has the following responsibilities:

  - Setting up the Redis server.
  - Configuring the Redis server.
  - Maintaining the Redis server.
  - Monitoring the Redis server.

Troubleshooting for WEC clustering

This section provides troubleshooting information and solutions in connection with Windows Event Collector (WEC) clustering .

Log messages and why the WEC sends them

This section describes the possible log messages you may get while using Windows Event Collector (WEC) clustering with syslog-ng Premium Edition (syslog-ng PE), and why the WEC sends them.

- If Redis is not available during startup, the WEC instance cannot start. In this case, you will get a similar log message:

  ```
  2020-11-16T21:24:03.843Z        FATAL   state/redisstate.go:17  RedisConn:
  Error connecting to Redis    {"error": "RedisConn: connection failure: dial
  tcp 192.168.0.14:6379: connect: connection refused"}
  ```

- If Redis is disconnected, you will get a similar log message:

  ```
  2020-11-16T21:11:12.818Z          ERROR    state/redisconn.go:55
  RedisConn: dial failed   {"error": "dial tcp 192.168.0.14:6379: connect:
  connection refused"}
  ```

- If you are trying to ping Redis periodically (in this case, the ping period is 1 second), you will get a similar log message:

```
2020-11-16T21:11:12.818Z          DEBUG    state/redisconn.go:115   RedisConn
is still disconnected
 2020-11-16T21:11:13.819Z          ERROR    state/redisconn.go:55
RedisConn: dial failed   {"error": "dial tcp 192.168.0.14:6379: connect:
connection refused"}
```

- I Redis eventually becomes available, you will get a similar log message:

```
2020-11-16T21:13:59.829Z        DEBUG   state/redisconn.go:136  RedisConn is
connected
2020-11-16T21:13:59.829Z        INFO    wec/main.go:120 Redis connection
restored, starting server...
2020-11-16T21:13:59.830Z        INFO
eventstorage/datagrameventstorage.go:34 Trying to connect to unix datagram
socket       {"unix-datagram": "/home/vagrant/wec_unix_dgram"}
2020-11-16T21:13:59.830Z        INFO
eventstorage/datagrameventstorage.go:44 Connected to unix datagram socket
 {"unix-datagram": "/home/vagrant/wec_unix_dgram"}
```

Checking data stored in Redis

You can use the following commands to check your data stored in Redis.

NOTE: The following commands must be run from the CLI tool within Redis.

- Listing subscription IDs for an event source (or host):

  ```
  127.0.0.1:6379> HKEYS win1
  1) "FE14EC9A-A667-5375-B0B5-C4C4A9A6F745"
  ```

  The command lists the available subscriptions (in this case, `FE14EC9A-A667-5375-B0B5-C4C4A9A6F745` is the only one available) for the event source or host (in this case, `win1`).

- Getting bookmarks for a subscription ID:

  ```
  127.0.0.1:6379> HGET win1 FE14EC9A-A667-5375-B0B5-C4C4A9A6F745
  "<BookmarkList><Bookmark Channel=\"Application\" RecordId=\"11098\"
  IsCurrent=\"true\"/></BookmarkList>"
  ```

  The command lists the bookmark value (`RecordId=\"11098\"`) from the `FE14EC9A-A667-5375-B0B5-C4C4A9A6F745` subscription ID within the `win1` event source.

Timers on the WEC side

This section describes the predefined timers on the Windows Event Collector (WEC) side, and how they affect the interactions of syslog-ng PE and Redis during WEC clustering.

NOTE: The timers described in this section are not user-configurable in version 7.0.23 of syslog-ng PE.

In version 7.0.23 of syslog-ng PE, the following predefined timers are used during WEC clustering:

Timer	Description
`healthCheckInterval (periodical PING): 1 second`	When Redis is disconnected, the WEC instance is trying to PING Redis periodically. The period length is 1 second.
`connectTimeout: 10 seconds`	Connect operation fails when a connection to Redis cannot be established within 10 seconds.
`readTimeout: 5 seconds`	Redis is disconnected when a read operation cannot be finished within 5 seconds.
`writeTimeout: 5 seconds`	Redis is disconnected when a write operation cannot be finished within 5 seconds.

About us

One Identity solutions eliminate the complexities and time-consuming processes often required to govern identities, manage privileged accounts and control access. Our solutions enhance business agility while addressing your IAM challenges with on-premises, cloud and hybrid environments.

Contacting us

For sales and other inquiries, such as licensing, support, and renewals, visit https://www.oneidentity.com/company/contact-us.aspx.

Technical support resources

Technical support is available to One Identity customers with a valid maintenance contract and customers who have trial versions. You can access the Support Portal at https://support.oneidentity.com/.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to videos at www.YouTube.com/OneIdentity
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product