One Identity Safeguard for Privileged Sessions 5.10

Creating custom Credential Store plugins

The information in this document is provided in connection with One Identity products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of One Identity LLC products. EXCEPT AS SET FORTH IN THE TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, ONE IDENTITY ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ONE IDENTITY BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ONE IDENTITY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. One Identity makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. One Identity does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

One Identity LLC.
Attn: LEGAL Dept
4 Polaris Way
Aliso Viejo, CA 92656

Refer to our Web site (http://www.OneIdentity.com) for regional and international office information.

Patents

One Identity is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at http://www.OneIdentity.com/legal/patents.aspx.

Trademarks

One Identity and the One Identity logo are trademarks and registered trademarks of One Identity LLC. in the U.S.A. and other countries. For a complete list of One Identity trademarks, please visit our website at www.OneIdentity.com/legal. All other trademarks are the property of their respective owners.

Legend

> ⊗ **WARNING: A WARNING icon indicates a potential for property damage, personal injury, or death.**

> ⚠ **CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

> ⓘ IMPORTANT, NOTE, TIP, MOBILE, or VIDEO: An information icon indicates supporting information.

SPS Creating custom Credential Store plugins
Updated - March 2019
Version - 5.10

Contents

Introduction

The following sections provide an overview on creating custom Credential Store plugins that can be used to authenticate on the target servers using an external Credential Store server (for example, a password manager or SSH private key store). For details on using an existing plugin, see "Integrating external authentication and authorization systems" in the Administration Guide.

⚠ **CAUTION:**

Using custom plugins in SPS is recommended only if you are familiar with both Python and SPS. Product support applies only to SPS: that is, until the entry point of the Python code and passing the specified arguments to the Python code. One Identity is not responsible for the quality, resource requirements, or any bugs in the Python code, nor any crashes, service outages, or any other damage caused by the improper use of this feature, unless explicitly stated in a contract with One Identity. If you want to create a custom plugin, contact our Support Team for details and instructions.

The Credential Store plugin is a Python module.One Identity Safeguard for Privileged Sessions (SPS) invokes the module to request the password or the SSH private key of the target user. The plugin processes the request, returns the result to SPS, and exits. SPS then processes the result.

The backup and restore functionality of SPS handles the uploaded credential store plugin as part of SPS's configuration. You do not need to create separate backups of your Credential Store plugin.

2

Structure of a plugin

An SPS plugin is a .zip file that contains a MANIFEST file (with no extension) and a Python module named main.py in its root directory. The plugin .zip file may also contain an optional default.cfg file that serves to provide an example configuration that you can use as a basis for customization if you wish to adapt the plugin to your site's needs. The size of the .zip file is limited to 20 megabytes.

The MANIFEST file

The MANIFEST file is a YAML file and should conform to version 1.2 of the YAML specification. It should contain the following information about the plugin:

- api: The version number of the SPS API. Must be 1.0.
- type: The type of the plugin. It must be credential store for a Credential Store plugin, and authentication and authorization plugin for an Authentication and Authorization plugin.
- name: The name of the plugin.
- version: The version number of the plugin. Must be in <major-version>.<minor-version> format, for example, 0.4, 1.5, 3.3, and so on.
- description: The description of the plugin. This description is displayed on the SPS web interface.

Example

```
api: 1.0
type: credentialstore
name: MyCustomPlugin
version: 1.0
description: Example plugin for SPS
```

The `main.py` module

The main.py file is a Python module that the framework attempts to execute. The following restrictions apply:

- The main.py module must contain the Plugin class.
- The Plugin class must have member methods for all defined hooks.

The plugin is executed when a predefined entry point (hook method) is invoked. After returning the result, the plugin exits immediately.

> NOTE:
>
> Plugins have a global timeout limit. The plugin timeout is half of the timeout value of the protocol proxy that uses the plugin (configured on the **<Protocol name> Control > Settings** page of the SPS web interface). By default, the proxy timeout is 600 seconds, meaning the plugin timeout is 300 seconds.

Hooks can be defined with zero or more arguments and can usually return None or a dict with the appropriate keys. The order of the hook arguments is not defined. Instead, all arguments are passed by name.

All arguments are optional. Only the arguments actually used in the hook need to be specified.

No global state is preserved inbetween calls. Therefore, you have to use the cookie key in the returned dictionary to persist data between subsequent calls of the same plugin or between the different methods of a plugin. The cookie should be a dictionary containing simple data items. It has to be serializable to JSON. To persist data between two different plugins used in the same session, use the session_cookie key.

You can use (**kwargs) to get all possible call arguments in a hook, including the cookie argument.

The following hooks must all be implemented:

- get_password_list: Called when a password is required to login on the target.
- get_private_key_list: Called when a private key is required to login on the target.
- authentication_completed: Called after a successful login attempt.
- session_ended: A session signifies the lifetime of a logical connection: it starts with logging in to the target, and ends when the connection ends. The session_ended hook is the notification for the end of the session. It is called exactly once for the same session.

get_password_list

Called when a password is required to login on the target. Can be called multiple times for the same session.

Input arguments

- `session_id`

Type:	string
Required:	no

  Description: The unique identifier of the session.

- `cookie`

Type:	dictionary
Required:	no

  Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- `session_cookie`

Type:	dictionary
Required:	no

  Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

- `protocol`

Type:	string
Required:	no

  Description: The protocol name, in lowercase letters (`http`, `ica`, `rdp`, `ssh`, `telnet`, `vnc`).

- `client_ip`

Type:	string
Required:	no

Description: A string containing the IP address of the client.

- `gateway_username`

Type:	string
Required:	no

- `gateway_password`

Type:	string
Required:	no

- `gateway_groups`

Type:	list
Required:	no

- `gateway_domain`

Type:	string
Required:	no

- `target_username`

Type:	string
Required:	no

- `target_host`

Type:	string
Required:	no

- `target_port`

Type:	int
Required:	no

- `target_domain`

Type:	string
Required:	no

Returned values

- `cookie`

Type:	dictionary
Required:	no

Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- `session_cookie`

Type:	dictionary
Required:	no

Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

- `passwords`

Type:	string list
Required:	no

Description: If the plugin returns multiple passwords, SPS tries to use them to authenticate on the target server (in the order they are listed).

get_private_key_list

Called when an SSH private key is required to login on the target. Can be called multiple times for the same session.

Input arguments

- session_id

Type:	string
Required:	no

  Description: The unique identifier of the session.

- cookie

Type:	dictionary
Required:	no

  Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- session_cookie

Type:	dictionary
Required:	no

  Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

- protocol

Type:	string
Required:	no

Description: The protocol name, in lowercase letters (`http`, `ica`, `rdp`, `ssh`, `telnet`, `vnc`).

- `client_ip`

Type:	string
Required:	no

Description: A string containing the IP address of the client.

- `gateway_username`

Type:	string
Required:	no

- `gateway_password`

Type:	string
Required:	no

- `gateway_groups`

Type:	list
Required:	no

- `gateway_domain`

Type:	string
Required:	no

- `target_username`

Type:	string
Required:	no

- `target_host`

Type:	string
Required:	no

- `target_port`

Type:	int
Required:	no

- `target_domain`

Type:	string
Required:	no

Returned values

- `cookie`

Type:	dictionary
Required:	no

Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- `session_cookie`

Type:	dictionary
Required:	no

Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

- `private_keys`

Type:	tuple list
Required:	no

Description: A list of (<key type>, <private key>) tuples. If the plugin returns multiple private keys, SPS tries to use them to authenticate on the target server (in the order they are listed).

The key type must be `ssh-rsa` or `ssh-dss`. The private key must be a well-formatted private key blob in PKCS#1 or PKCS#8 in PEM (RFC 1421) format, and must include the corresponding headers. The Base64-formatted part must correspond to the RFC: "To represent the encapsulated text of a PEM message, the encoding function's output is delimited into text lines (using local conventions), with each line except the last containing exactly 64 printable characters and the final line containing 64 or fewer printable characters."

X.509 certificates are not supported, only private keys are.

authentication_completed

Called after a successful authentication attempt.

ℹ TIP:

You can use this hook to check-in the password to the Credential Store (since the user will not need it anymore) or to trigger a password change for the host.

Input arguments

- `session_id`

Type:	string
Required:	no

Description: The unique identifier of the session.

- `cookie`

Type:	dictionary
Required:	no

Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- session_cookie

Type:	dictionary
Required:	no

Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

Returned values

- cookie

Type:	dictionary
Required:	no

Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- session_cookie

Type:	dictionary
Required:	no

Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

session_ended

A session is the logical unit of user connections: it starts with logging in to the target, and ends when the connection ends. SPS executes the session_id hook when the session is closed. It is called exactly once for the same session.

TIP:

You can use this hook to send a log message related to the entire session or close the ticket related to the session if the plugin interacts with a ticketing system.

You must implement the `session_ended` method in the plugin.

Input parameters

- `session_id`

Type:	string
Required:	no

  Description: The unique identifier of the session.

- `cookie`

Type:	dictionary
Required:	no

  Description: The cookie returned by the previous hook in the session. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by one of the previous calls in this particular custom Credential Store plugin. You can use the cookie to maintain the state for each particular connection or to transfer information between the different methods of the plugin. For an example that transfers information in the cookie between two methods, see "Examples" in the Creating custom Authentication and Authorization plugins.

- `session_cookie`

Type:	dictionary
Required:	no

  Description: You can use the session cookie to maintain global state between plugins for each particular connection. If this is the first call for that session, it is initialized as an empty dictionary, otherwise it has the value returned by a previous plugin hook in the session.

Returned values

This hook does not return values.

`session_ended` example

The following example formats every information received in the cookie into key-value pairs and prints a log message that includes this information.

Key-value pairs in log message

```
def session_ended(self, session_id, session_cookie, cookie):
session_details = ','.join([ '{0}={1}'.format(key, cookie[key]) for key in
sorted(cookie.keys()) ])
print "Session ended; session_id='{0}',
session_details='{1}'".format(session_id, session_details)
```

Plugin modification examples

The following example shows a simple plugin that can return both passwords and private keys based on usernames:

Example: return passwords and username-based private keys

```
class Plugin(object):
    passdb = {
        "user": ["password"],
    }
    privkeydb = {
        "user1": [('ssh-rsa', """
-----BEGIN RSA PRIVATE KEY-----
ISNFNFIASNFIANSFINSDIIISLLERfEJW++SppInNHlL89wTymILaxgln7FfQ2vr6
aBHymY/+Xwf08GiuLg2hFmfLNGZlJNnF9YB4+3o7MfjPDZJR1ne8Vr9hkte/SuK2
OhZbAeWbxHLsdOv0+ZCm7h5/nEM1gj4va+uKgpShVbxqEH7RglyUDvKUgQ7KwUZE
GW+RPApnXFN3OVjFdAqOpzeayH0kA52A3W/ske81JFGEHvfP54EePJx1qncJAX1z
jFPllYjPlMSLujbH7sabL0+LbnZDfMxOw2NXwnaKPgVlJ7I7YQDE11NLhiWbC2f1
pTLIerTOG9lovC3caa7TaIRs8VfZLjjNXWnS5wIDAQABAoIBAB6HLgz5eXIFT+ai
ISNFNFIASNFIANSFINSDIIISLLERfEJW++SppInNHlL89wTymILaxgln7FfQ2vr6
QScd2MYvJ9dIdumxbk5dK7+5I3fGHroXTRgUF6AIKI2FCsnQtDyTY1mjZ99+dGjH
AjOKnIbKPuaj+Mpx3dLhlhDgi+DncGSizhOtb3jK1tq++YLoA7W/7n9av5Ybz8c0
iqF0WUwcd6KYphuL9583OPP6Gv33Br4jP729EkqXnJa8PcniX8y3ZlFcVmxOGqnL
ISNFNFIASNFIANSFINSDIIISLLERfEJW++SppInNHlL89wTymILaxgln7FfQ2vr6
UumxiQECgYEA9yPcGBo/R/2IyjyKBXjYcd/1u0kYZRWvloahjNoWQjs/EHvbBMlM
xmtowOHbbEg4BgymPmVR8Ux24B3XJR6SbAPMF15wJ7oD1WwG8djQSw0RrbuPgP4s
OJnRpCn4blpa15n5qUF8wCwnEJow+UUaYY1znMlmAyeWjaK1VHV7tEUCgYEA8MH1
guHR+hHyZcLTT2+QTuL2Pu2MrwLhXNz5hPcCRH72dKBdfrvpRwLKj3XJKBK4r4gN
hByiT2sJKCNks4LkyOlWQtd0khRuan/xkliH7a6Fcx+d5odQsZrRbrjpsUQFlnTB
AFv6kSnhAtmJVDalYWfPSQCuE0nwB9TaDU6UGzsCgYAItvwA4ZQPrtIPB5l6XeuM
ISNFNFIASNFIANSFINSDIIISLLERfEJW++SppInNHlL89wTymILaxgln7FfQ2vr6
QDIHNO5RiE6wTPHlv1aA/wH7lVyXGN9oU4w/9Lbs9US0y5oxLL0Abc4m2LkXYSdv
ISNFNFIASNFIANSFINSDIIISLLERfEJW++SppInNHlL89wTymILaxgln7FfQ2vr6
FykNgS4dhrCG3NmpP4zQbKnS+VDQrLJ/qbSG59Ida8nIs74yanQX17EPuzqD/iJT
LoahB2128G7BiEfcIpFVCgI0OqikYQkM4oOQD3sUw8ySfi/rZMxGtT34uf7398FH
bBRnAoGBANRNw9oTcSh/ScLNqhB1pld81UX8jf+4+9hj9U+gpQCkujVxTs7xil8R
ISNFNFIASNFIANSFINSDIIISLLERfEJW++SppInNHlL89wTymILaxgln7FfQ2vr6
31nME0D1kojABIMeW8cITVHx4PD7I8jp+3sIPRXzCr8bfTzGSOAA
-----END RSA PRIVATE KEY-----
""")],
    }
    def get_private_key_list(self, session_id, cookie, protocol, client_ip,
                            gateway_username, gateway_password,
                            target_username, target_host, target_port,
                            target_domain=None, gateway_domain=None,
                            gateway_groups=None):
        keylist = []
        if target_username in self.privkeydb:
            keylist = self.privkeydb[target_username]
            print "Retrieved private keys;"
            print keylist
        else:
            print "User not found;"
        return {
            "private_keys": keylist,
        }
    def get_password_list(self, session_id, cookie, protocol, client_ip,
                        gateway_username, gateway_password,
                        target_username, target_host, target_port,
                        target_domain=None, gateway_domain=None
                        gateway_groups=None):
        pwlist = []
        if target_username in self.passdb:
            pwlist = self.passdb[target_username]
            print "Retrieved passwords;"
        else:
            print "User not found;"
        return {
            "passwords": pwlist,
        }
    def authentication_completed(self, session_id, cookie):
        return None
        def session_ended(self, session_id, cookie):
            return None
```

The following example demonstrates how the predefined hooks can be enhanced with additional logic:

Example: enhance predefined hooks

```
import inspect

class Plugin(object):
    passdb = {
        "joe": ["joespw1", "joespw2", ],
        "jack": ["jackspw", ],
    }

    def get_password_list(self, session_id, cookie, protocol, client_ip,
                        gateway_username, gateway_password,
                        target_username, target_host, target_port,
                        target_domain=None, gateway_domain=None, gateway_
groups=None):

        # Discard "None" parameters, log all other returned parameters
        args = list(inspect.getargvalues(inspect.currentframe()).args)
        logkws = ["{arg}='{value}'".format(arg=arg, value=locals()[arg])
        for arg in args if arg != 'self' and locals()[arg] is not None]

        if "call_count" in cookie:
            call_count = cookie["call_count"]
        else:
            call_count = 0

        logkws.append("call_count='{0}'".format(call_count))

        print ("Retrieving passwords, non-null parameters follow; " + ', '.join
(logkws))

        # Return the password list for the user
        pwlist = []
        if target_username in self.passdb:
            pwlist = self.passdb[target_username]
            print "Retrieved passwords;"
        else:
            print "User not found;"

        return {
            "passwords": pwlist,
            "cookie": {"call_count": call_count + 1}
        }
    def authentication_completed(self, session_id, cookie):
        call_count = cookie["call_count"] if "call_count" in cookie else None
        print ("Received notification about completed authentication; "
            "call_count='{call_count}'").format(call_count=call_count)
        return None

    def session_ended(self, session_id, cookie):
        call_count = cookie["call_count"] if "call_count" in cookie else None
        print ("Received notification about session end; "
            "call_count='{call_count}'").format(call_count=call_count)
        return None
```

Including additional modules

You can invoke additional Python modules from `main.py`, provided that the total size of the `.zip` bundle does not exceed 20 megabytes and all calls are executed within the plugin timeout.

The modules must be compatible with Python version 2.7.12. All built-in modules of the The Python Standard Library are included in the environment, plus the following additional modules:

- DNS
- OpenSSL

The sample configuration file (`default.cfg`)

Your plugin `.zip` file may contain an optional `default.cfg` sample configuration file. This file serves to provide an example configuration that you can use as a basis for customization if you wish to adapt the plugin to your site's needs.

The only prerequisites for this file are as follows:

- It must be a UTF-8 encoded text file.
- The size of the file must not exceed 10 KiB.

Other than these prerequisites, the contents of the file are not restricted in any way.

Troubleshooting

On the default log level, SPS logs everything that the plugin writes to `stdout` and `stderr`. Log message lines are prefixed with the session ID of the proxy, which makes it easier to find correlating messages.

To transfer information between the methods of a plugin (for example, to include data in a log message when the session is closed), you can use a cookie.

If an error occurs while executing the plugin, SPS automatically terminates the session.

NOTE:

This error is not visible in the verdict of the session. To find out why the session was terminated, you have to check the logs.

Using a custom Credential Store plugin to authenticate on the target hosts

The following describes how to configure SPS to retrieve the credentials used to login to the target host using a custom plugin.

Prerequisites

To use a custom Credential Store plugin, you have to upload a working Credential Store plugin to SPS. This plugin is a script that can be used to access an external Credential Store or Password Manager. If you want to create such a custom Credential Store plugin, contact our Support Team.

> NOTE:
>
> Users accessing connections that use Credential Stores to authenticate on the target server must authenticate on SPS using gateway authentication. Therefore, gateway authentication must be configured for these connections. For details, see "Configuring gateway authentication" in the Administration Guide.

To upload the custom Credential Store plugin you received, navigate to **Basic Settings > Plugins > Upload/Update Plugins**, browse for the file and click **Upload**.

> NOTE:
>
> It is not possible to upload or delete Credential Store plugins if SPS is in "Sealed mode" in the Administration Guide.

Your plugin `.zip` file may contain an optional sample configuration file. This file serves to provide an example configuration that you can use as a basis for customization if you wish to adapt the plugin to your site's needs.

To configure SPS to retrieve the credentials used to login to the target host using a custom plugin

1. Navigate to **Policies > Credential Stores**.
2. Click + and enter a name for the Credential Store.

3. Select **External Plugin**, then select the plugin to use from the **Plugin** list.
4. If your plugin supports configuration, then you can create multiple customized configuration instances of the plugin for your site. The **Configuration** textbox displays the example configuration of the plugin you selected. If you wish to create a customized configuration instance of the plugin for your site, then edit the configuration here.

   > NOTE:
   >
   > Plugins created and issued before the release of SPS 5 F1 do not support configuration. If you create a configuration for a plugin that does not support this, the affected connection will stop with an error message.

5. Click .
6. Navigate to the Connection policy where you want to use the Credential Store (for example, to **SSH Control > Connections**), select the Credential Store configuration instance to use in the **Credential Store** field, then click .

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 or other inquiries, visit https://www.oneidentity.com/company/contact-us.aspx or call +1-800-306-9329.

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