



One Identity Safeguard for Privileged Sessions 5.10

Creating custom Authentication and Authorization plugins

Copyright 2019 One Identity LLC.

ALL RIGHTS RESERVED.

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of One Identity LLC .

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 Authentication and Authorization plugins
Updated - March 2019
Version - 5.10

Contents

Introduction	4
How the Authentication and Authorization plugin works	5
Structure of a plugin	7
The manifest file	7
The main.py module	8
authenticate	8
authorize	15
session_ended	20
Examples	21
Tips and tricks	22
Including additional modules	23
The sample configuration file	23
Troubleshooting	24
Authorizing connections to the target hosts with a SPS plugin	25
Integrating SPS to ticketing systems	27
About us	28
Contacting us	28
Technical support resources	28

Introduction

The following sections provide an overview on creating custom plugins for One Identity Safeguard for Privileged Sessions (SPS) to authenticate your users to external services in addition to the authentication performed on the target server. For example, such plugins can implement two-factor (2FA) or multi-factor authentication (MFA) methods, or request the user to provide a valid ticket ID for the connection. 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.

Every SPS plugin is a Python module. SPS invokes the module to request the password 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 plugins as part of SPS's configuration. You do not need to create separate backups of your plugins.

How the Authentication and Authorization plugin works

If a Connection Policy has an Authentication and Authorization plugin (AA plugin) configured, SPS executes the plugin as the last step of the connection authorization phase. SPS can request the client to perform other type of authentication before executing the plugin. Using an AA plugin in a Connection Policy is treated as a gateway authentication if:

- the plugin authenticates the user,
- authentication is successful,
- the plugin returns the `gateway_user` and `gateway_groups` values, identifying the user it has authenticated.

Other types of gateway authentication will come before authentication by the AA plugin, so information from any other type of gateway authentication (for example, the username and usergroups of this authentication) will already be available (and therefore can be used by the plugin). If the AA plugin does perform gateway authentication, you can use a Credential Store as well.

However, for technical reasons, the web-based gateway authentication (that is, authenticating on the SPS web interface if the **Require Gateway Authentication on the SPS Web Interface** option is selected in the Connection Policy) is performed after the **AA plugin**, so using **AA plugin** and ticking **Require Gateway Authentication on the SPS Web Interface** at the same time is not a valid configuration.

SPS executes the `authorize` method after the authentication method and any inband gateway authentication or inband destination selection steps. As a result, the `authorize` method already has access to the IP address of the target server and the remote username (the username used in the server-side connection).

In the RDP, SSH, and Telnet protocols, the plugin can interactively request additional information from the client.

Optionally, the plugin can return the `gateway_user` and `gateway_groups` values. SPS will only update the gateway username and gateway groups fields in the connection database if the plugin returns the `gateway_user` and `gateway_groups` values. The returned `gateway_user` and `gateway_groups` values override any such attributes already available on SPS about the connection (that means that channel policy evaluations will be affected), so make sure that the plugin uses the original values appropriately.

If the plugin returns the `gateway_user` and `gateway_groups` values, you may have to configure an appropriate **Usermapping policy** in the **Connection Policy**. If the plugin returns a `gateway_user` that is different from the remote user, the connection will fail without a usermapping policy. For details on usermapping policies, see "[Configuring usermapping policies](#)" in the [Administration Guide](#).

Prerequisites

- SPS supports Authentication and Authorization plugins in the RDP, SSH, and TELNET protocols.
- In RDP, using an AA plugin together with Network Level Authentication in a Connection Policy has the same limitations as using Network Level Authentication without domain membership. For details, see "[Network Level Authentication without domain membership](#)" in the [Administration Guide](#).
- In RDP, using an AA plugin requires TLS-encrypted RDP connections. For details, see "[Enabling TLS-encryption for RDP connections](#)" in the [Administration Guide](#).

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.

i 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:

- [authenticate on page 8](#): Called to identify the user connecting through SPS.
- [authorize on page 15](#): Called when the remote username and the address of the target server are available (after the authentication hook and any inband gateway authentication or inband destination selection are completed).
- [session_ended on page 20](#): Called when the session is closed. It is called exactly once for the same session. For example, 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.

authenticate

The `authenticate` method performs the authentication of the session and returns a verdict that determines if SPS permits the connection to continue to the target server.

Example

```
def authenticate(self,
    session_id,
    protocol,
    connection_name,
    client_ip,
    client_port,
    key_value_pairs):
    return {
        'verdict': 'ACCEPT',
        'additional_metadata': 'my_metadata',
        'my_key': 'my_value',
    }
```

You must implement the authenticate method in the plugin.



TIP:

If you do not want to do anything in this method, include an empty method that returns the ACCEPT verdict.

Example

```
def authenticate (self):
    return {
        'verdict': 'ACCEPT',
    }
```

In addition, no gateway authentication has been performed by the plugin if the authenticate method returns:

- None.
- The dict {'verdict': 'NONE'}.

Input arguments

The order of the arguments does not make a difference, only their names do. Every argument is optional.

- 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 AA 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](#) on page 21.

- 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.

- connection_name

Type:	string
-------	--------

Description: The name of the Connection policy that handles the client's connection request.

- client_ip

Type:	string
-------	--------

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

- client_port

Type: int

Description: The port number of the client.

- gateway_user

Type: string

Description: Contains the gateway username of the client if already available (for example, if the user performed inband gateway authentication), otherwise its value is None.

- key_value_pairs

Type: dictionary

Description: A dictionary containing plugin-specific information (for example, it may include a token ID). This dictionary also contains any key-value pairs that the user specified. In the plugin, such fields are already parsed into separate key-value pairs. For details on how the user can provide such data during a connection, see ["Integrating external authentication and authorization systems" in the Administration Guide](#).

- protocol

Type: string

Description: The protocol used in the connection that the plugin is currently processing. Enter one of the following values: rdp, ssh, telnet.

- target_server

Type: string

Description: Contains information about the target server if already available (for example, if the user performed inband gateway authentication), otherwise its value is None.

- target_port

Type: integer

Description: Contains information about the target port if already available (for example, if the user performed inband gateway authentication), otherwise its value is None.

Returned values

The method must return a dictionary with the following (required or optional) elements.

The required elements are:

- `verdict`, which must contain one of the following returned values:
 - `ACCEPT`, which returns `gateway_user` and `gateway_groups` together.
 - `NEEDINFO`, which returns `question`.

The optional elements are:

- `verdict`, which contains one of the following values:
 - `DENY`
 - `NONE`
- `cookie`
- `session_cookie`
- `additional metadata`

The elements in more detail:

`verdict`

Type: string or None

Description: Must contain one of the following values:

- `ACCEPT`: The authentication was successful, the client can continue the connection. If the plugin returns both `gateway_users` and `gateway_groups` elements, it means that gateway authentication has been performed.
- `DENY`: Reject the connection.
- `NEEDINFO`: The authentication requires more information to be completed.
- `NONE`: No gateway authentication was performed by the plugin.

For example, the following sample code rejects the connection.

Example

```
return {'verdict': 'DENY'}
```

- gateway_user

Type: string

- gateway_groups

Type: list

Description: Optionally, the plugin can return the gateway_user and gateway_groups values. SPS will only update the gateway username and gateway groups fields in the connection database if the plugin returns the gateway_user and gateway_groups values. The returned gateway_user and gateway_groups values override any such attributes already available on SPS about the connection (which means that channel policy evaluations will be affected), so make sure that the plugin uses the original values appropriately.

NOTE:

If the plugin returns the gateway_user and gateway_groups values, you may have to configure an appropriate **Usermapping Policy** in the Connection Policy. If the plugin returns a gateway_user that is different from the remote user, the connection will fail without a Usermapping Policy. For details on Usermapping Policies, see ["Configuring usermapping policies" in the Administration Guide](#).

For example, the following sample code accepts the connection and sets the gateway_user and gateway_groups fields. (Naturally, you should write the plugin code that actually retrieves these data from the third-party system.) For details, see [Examples on page 21](#).

Example

```
return {
    'verdict': 'ACCEPT', 'gateway_user':
        'username-received-from-third-party',
    'gateway_groups': [
        'usergroup1-received-from-third-party',
        'usergroup2-received-from-third-party']
}
```

- 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 AA 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](#) on page 21.

- 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.

- additional_metadata

Type:	string
Required:	no

Description: The value of this string will be stored in the **Additional metadata** column of the SPS connection database, and will be available on the SPS search interface.

- question

Type:	tuple
Required:	no

Description: A tuple that contains key-question pairs and optionally a third element to disable echoing. You can use it to request additional information from the client when using the NEEDINFO verdict in RDP, Telnet, and SSH connections. For example, the following sample code displays a prompt (in this case, Enter your token number) to the user. For details, see [Examples](#) on page 21.

Example

```
return {
    'verdict': 'NEEDINFO', 'question':
    ('token', 'Enter your token number: ')
}
```

If the optional third element is True, the answer will not be echoed to the client.

TIP:

Set the third element to True if the answer to the question is sensitive information (for example, a password).

Example

```
return {
  'verdict': 'NEEDINFO', 'question':
    ('token', 'Enter your token number: ', True
  }
```

Note that in SPS version 4.3.0 and 4.3.1, question was a dictionary. Starting with version 4.3.2, it is a tuple.

Requesting more information from the client

To request additional information from the client (for example, a one-time password from a token, or a ticket ID), the authenticate method may return the NEEDINFO verdict and the question tuple containing key-question pairs. The questions are asked from the user in a protocol-specific way and the authenticate method is called again with a key_value_pairs argument containing the answers in key-answer pairs, where the key belongs to the corresponding question. Alternatively, you can also use the cookie to supply additional information to the plugin.

Currently only the Telnet and SSH protocols are supported.

authorize

The authorize method performs the authorization of the session and returns a verdict that determines if SPS permits the connection to continue to the target server. This method is executed only once. SPS executes the authorize method after the authentication method, and any inband gateway authentication or inband destination selection steps. As a result, the authorize method already has access to the IP address of the target server and the remote username (the username used in the server-side connection). You must implement the authorize method in the plugin.

TIP:

If you do not want to do anything in this method, include an empty method that returns the ACCEPT verdict. Otherwise, the connection will fail with the following log message: Calling Authorize hook of AA plugin failed..

Example

```
def authorize (self):  
    return {'verdict': 'ACCEPT'}
```

Input arguments

The order of the arguments does not make a difference, only their names do. Every argument is optional.

- 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 AA 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](#) on page 21.

- 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.

- connection_name

Type:	string
-------	--------

Description: The name of the Connection Policy that handles the client's connection request.

- client_ip

Type: string

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

- client_port

Type: int

Description: The port number of the client.

- gateway_groups

Type: string list

Description: The final gateway groups of the gateway user.

- key_value_pairs

Type: dictionary

Description:

A dictionary containing plugin-specific information (for example, it may include the username).

This dictionary also contains any key-value pairs that the user specified when establishing the connection. In the plugin, such fields are already parsed into separate key-value pairs. For details on how the user can provide such data during a connection, see ["Integrating external authentication and authorization systems" in the Administration Guide](#).

- protocol

Type: string

Description: The protocol used in the connection that the plugin is currently processing. Enter one of the following values: rdp, ssh, telnet.

- client_port

Type: int

Description: The port number of the client.

- target_server

Type: string

Description: Contains information about the target server if already available (for example, if the user performed inband gateway authentication), otherwise its value is None.

- target_port

Type: integer

Description: Contains information about the target port if already available (for example, if the user performed inband gateway authentication), otherwise its value is None.

- target_username

Type: string

Description: The username SPS uses to authenticate on the target server.

Returned values

The method must return a dictionary with the following (required or optional) elements.

The required elements are:

- verdict, which must contain one of the following returned values:
 - ACCEPT, which indicates that the authentication was successful and the client can continue the connection.
 - DENY, which rejects the connection.

The optional elements are:

- cookie
- session_cookie
- additional metadata

The elements in more detail:

- verdict

Type: string

Must contain one of the following values:

- ACCEPT: The authentication was successful, the client can continue the connection.
- DENY: Reject the connection.

For example, the following sample code rejects the connection.

Example

```
return {
    'verdict': 'DENY'
}
```

- 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 AA 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](#) on page 21.

- 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.

- additional_metadata

Type:	string
Required:	no

Description: The value of this string will be stored in the **Additional metadata** column of the SPS connection database, and will be available on the SPS search interface.

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
-------	--------

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 AA 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](#) on page 21.

- `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 including this information into the log file.

Example

```
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)
```

Examples

The following example checks if the user has entered the string good as the token number. If the value of the token number is anything other than good, the plugin displays a prompt to the user up to three times. After three unsuccessful attempts, the plugin terminates the connection.

Example

```
def authenticate(self, key_value_pairs, cookie):
    if key_value_pairs.get('token') == "good":
        return {
            'verdict': 'ACCEPT'}

    cookie['cnt'] = cookie.get('cnt', 0) + 1
    if cookie['cnt'] > 3:
        return { 'verdict': 'DENY' }

    return {'verdict': 'NEEDINFO', 'question':
            ('token', 'Enter token number: '), 'cookie': cookie
    }
```

The following example shows how to use the cookie to transfer data from the authenticate method to the session_ended method.

Example

```
import sys

class Plugin(object):

    def authenticate(self, session_id, cookie, protocol,
                    connection_name, client_ip, client_port, key_value_pairs):
        token = key_value_pairs.pop('token', None)

        # Accept the connection if the user provides a token number
        if token:
            # Write code here that validates the token number and
            # retrieves the username and usergroups of the user
            # We add the client_ip to the 'cookie' so it will be
            # available in the session_ended method as well
            return {'verdict': 'ACCEPT', 'gateway_user':
                    'username-received-from-third-party', 'gateway_groups':
                    ['usergroup1-received-from-third-party',
                    'usergroup2-received-from-third-party'],
                    'additional_metadata': token, 'cookie':
                    {'client_ip': client_ip}}

        # Display a prompt to the user to request a token number
        else:
            return {'verdict': 'NEEDINFO', 'question':
                    ('token', 'Enter your token number: ')}

    def session_ended(self, session_id, cookie):
        session_details = ',
        '.join([ '{0}={1}'.format(key, cookie[key]) for key in
                sorted(cookie.keys()) ])

        # Send a log message when the session ends, including the
        # client_ip address received in the cookie
        print "Session ended; session_id='{0}',
        session_details='{1}'".format(session_id, session_details)
```

Tips and tricks

If you need the public hostname of SPS in the plugin, the plugin can read it from the `/etc/hostnickname` file.

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.

For more information about module compatibility, see ["The available Python environments" in the Administration Guide](#).

The sample configuration file

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.

Authorizing connections to the target hosts with a SPS plugin

Purpose:

To configure SPS to use an Authentication and Authorization plugin before accessing the target host, complete the following steps.

Prerequisites:

- To use a custom plugin, you need to upload a working AA plugin to SPS. This plugin is a script that uses the SPS API to access an external system. If you want to create such a plugin, [contact our Support Team](#) for details and instructions.
- SPS supports Authentication and Authorization plugins in the RDP, SSH, and Telnet protocols.
- In RDP, using an AA plugin together with Network Level Authentication in a Connection Policy has the same limitations as using Network Level Authentication without domain membership. For details, see "[Network Level Authentication without domain membership](#)" in the [Administration Guide](#)
- In RDP, using an AA plugin requires TLS-encrypted RDP connections. For details, see "[Enabling TLS-encryption for RDP connections](#)" in the [Administration Guide](#)

Steps:

1. To upload the custom 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 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.

2. If your plugin supports configuration you can create multiple customized configuration instances of the plugin for your site. Create an instance by completing

the following steps:

- a. Go to **Policies > AA Plugin Configurations**. Select the plugin to use from the Plugin list.
- b. The **Configuration** textbox displays the example configuration of the plugin you selected. You can edit the configuration here if you wish to create a customized instance of the plugin.

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.

Figure 1. **Policies > AA Plugin Configurations** — Creating a customized plugin configuration instance

The screenshot shows a configuration window titled 'example_aa_plugin'. It features a 'Plugin:' dropdown menu with 'Ticket_query' selected. Below it is a 'Configuration:' text area containing the following text:

```
# This is a sample configuration.
# It must be an UTF-8-encoded test file but its interpretation is solely
# at the plugin's discretion, it can be in any format, like INI, YAML,
# JSON, and so on.
# This example is an INI style config.
[somesection]
somekey=somevalue
```

3. Navigate to the Connection Policy where you want to use the plugin (for example, to **RDP Control > Connections**), select the plugin configuration instance to use in the **AA plugin** field, then click  .

If the plugin sets or overrides the username of the connection, configure a Usermapping policy and use it in the Connection Policy. For details, see [Administration Guide](#).

Verify that the configuration works properly: try to establish a test connection. For details, see "[Performing authentication with AA plugin in Remote Desktop connections](#)" in the [Administration Guide](#). If the plugin is configured to store any metadata about the connection, these data will be available in the **Additional metadata** field of the SPS Search interface.

Integrating SPS to ticketing systems

From SPS 5 LTS and later, this functionality is available using the Authentication and Authorization (AA) plugin. SPS executes the `authorize` method after the authentication method, and any inband gateway authentication or inband destination selection selection steps. As a result, the `authorize` method already has access the IP address of the target server, and the remote username (that is, the username used in the server-side connection).

To use an AA plugin to integrate SPS to a ticketing system, note the following points.

- You can only request the ticket ID or other information from the user in the authentication hook ([authenticate](#) on page 8). For details on how the user can provide such data during a connection, see ["Integrating external authentication and authorization systems" in the Administration Guide](#).
- You must implement the actual authorization (for example, connecting and querying the ticketing system) in [authorize](#) on page 15. As a side effect, if the user submits an invalid ticket ID (or other invalid information) in the authentication hook, this error will not be recognized until the authorization hook. The user cannot correct this error and SPS will reject the connection. In this case, the user must initiate a new connection to provide the correct information.
- Only the Remote Desktop (RDP), Secure Shell (SSH), and Telnet protocols are supported.

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