One Identity Safeguard for Privileged Sessions 6.0.11

Creating custom Credential Store plugins

Legend

WARNING: A WARNING icon highlights a potential risk of bodily injury or property damage, for which industry-standard safety precautions are advised. This icon is often associated with electrical hazards related to hardware.

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

SPS Creating custom Credential Store plugins
Updated - 19 August 2021, 11:42
Version - 6.0.11

Contents

1

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

Plugin packaging

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, which 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.

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 the selected Python environment. For more information, see the available Python environments.

The MANIFEST file

The MANIFEST file is a YAML file and should conform to version 1.2 of the YAML specification.

It must contain the following information about the plugin:

- name: The identifier of the plugin during the upload to SPS. The initial character must be an alphabetical character, while the rest may be alphabetical characters, numerals or '_'. While case sensitivity is supported, special characters (for example, '@' or '&') are not permitted.
- description: The description of the plugin. This description is displayed on the SPS web interface.
- version: The version number of the plugin. It must begin with a numeral (for example, 2.0.3).
- type: The type of the plugin. It must be credentialstore for a Credential Store plugin and aa for an Authentication and Authorization plugin.
- api: The version number of the required SPS API. The current version number is 1.1.

It may contain the following elements:

- entry_point: main.py: The custom entry point of the plugin. If ommitted, the plugin will be executed with Python2 interpreter. If included, the plugin will be executed with an interpreter specified on the first line of the main.py file. For more information, see the available Python environments.
- scb_min_version: The minimum syslog-ng Store Box product version compatible with the plugin. For example, 5.10.0 means 5F10.
- scb_max_version: The maximum compatible syslog-ng Store Box product version. To allow any version below a certain value, add the ~charater. For example, 5.11.0~ means "any version up till, but not including, 5.11.0".

Example

```
name: name: SPS_TPAM
description: OneIdentity TPAM plugin
version: 2.0.1
type: credentialstore
api: 1.1
entry_point: main.py
```

API versioning

SPS supports only a single version of the plugin API.

The required version of SPS API must be in <major number>.<minor number> format.

NOTE:

SPS uses semantic versioning for the API. That is, if the plugin requires API version <x>.<y>, the API version's <major number> must be equal to <x> and the <minor number> must be equal to, or greater than, <y>. Otherwise the plugin cannot be uploaded.

For example, if the API version of SPS is `1.3`, SPS can use plugins with the required API version numbers `1.0`, `1.1`, `1.2`, and `1.3`. Versions `1.4` and `2.0` will not work.

Currently the API version number is 1.1.

Plugin versioning with Python2 legacy plugins

For Python2 legacy plugins the `api: version` should be `1.0`.

Plugin versioning for Python3 plugins using the Plugin SDK module

For Python3 plugins using the Plugin SDK module the `api: version` should be the same as the <major number>.<minor number> version of the Plugin SDK. That is, if the Plugin SDK version is `1.2`, write `api: 1.2` in the `MANIFEST` file.

NOTE:

The plugin does not need to be upgraded as long as the <major number> version remains the same, therefore the plugin should work with `1.3`, `1.4` or higher API versions.

The available Python environments

If you have no `entry_point` in the `MANIFEST` file

The plugins must be compatible with Python version 2.6.5, and have access to the following Python modules:

- `dns`
- `httplib`
- `json`
- `lxml`
- `openssl`
- `urllib`
- `urllib2`
- `xml`
- `xmllib`
- `xmlrpclib`

If you have `entry_point: main.py` in the `MANIFEST` file (the `main.py` starting with '#!/usr/bin/env pluginwrapper3')

In this case, the plugin must be Python 3.6.7 compatible. The plugin has access to these Python 3 modules:

oneidentity_safeguard_sessions_plugin_sdk (version == 1.1.2, https://oneidentity.github.io/safeguard-sessions-plugin-sdk/1.1.2/)

> NOTE:
>
> The <major> and <minor> version number of Plugin SDK is always equal to the SPS API version of the same release.

The Plugin SDK module mentioned above is a tool that allows you to reliably access SPS features and can be downloaded from Downloads page. In addition, the Plugin SDK module also allows you to develop or test plugins outside SPS. For more detailed information about the Plugin SDK module, see the Developer's Guide here.

- `pyOpenSSL (version >= 17.5.0, https://pyopenssl.org/en/17.5.0/index.html)`
- `python-ldap (version >= 3.0.0, https://www.python-ldap.org/en/python-ldap-3.0.0/)`
- `requests (version >= 2.18.4, http://docs.python-requests.org/en/master/)`
- `urllib3 (version >= 1.22, https://urllib3.readthedocs.io/en/latest/)`
- `pyyaml (version >= 3.12, https://pyyaml.org/)`

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. SPS searches for the plugin hook implementations under the `Plugin` class. SPS instantiates this class and invokes the hooks on the resulting instance.
- The `Plugin` class must have an `__init__(self, configuration="")` method. This is how the **Configuration** (for example, at **Policies >AA Plugin Configuration > Configuration** or **Policies > Credential Stores > Configuration**) is passed to the `Plugin` instance as string.
- 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,therefore the default 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 is the logical unit of user connections: 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

  Description: The unique identifier of the session.

- `cookie`

  Type: dictionary

  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

  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

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

- `client_ip`

Type: string

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

- `gateway_username`

  Type: string

- `gateway_password`

  Type: string

- `gateway_groups`

  Type: list

- `gateway_domain`

  string

- `target_username`

  string

- `target_host`

  string

- `target_port`

  Type: int

- `target_domain`

  Type: string

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

  Description: The unique identifier of the session.

- `cookie`

  Type: dictionary

  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

  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

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

- `client_ip`

  Type: string

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

- `gateway_username`

  Type: string

- `gateway_password`

  Type: string

- `gateway_groups`

  Type: list

- `gateway_domain`

  Type: string

- `target_username`

  Type: string

- `target_host`

  Type: string

- `target_port`

  Type: int

- `target_domain`

  Type: string

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.

> 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

  Description: The unique identifier of the session.

- `cookie`

  Type: dictionary

  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

  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 arguments

- `session_id`

  Type: string

  Description: The unique identifier of the session.

- `cookie`

  Type: dictionary

  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

  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
```

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.

3

Plugin troubleshooting

On the default log level, One Identity Safeguard for Privileged Sessions (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.

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