certsrv Documentation

Release 2.1.0

Magnus Watn

Dec 09, 2018

Contents

A Python client for the Microsoft AD Certificate Services web page

It is quite normal to have an internal PKI based on the Microsoft AD Certificate Services, which work great with Windows, but not so much on other OSes. Users of other OSes must often manually create a CSR and then use the Certificate Services web page (certsrv) to get a certificate. This is not ideal, as it is a manual and time consuming (and creating a csr with OpenSSL on the command line is confusing and complicated.)

This is a simple litle Python client for the certsrv page, so that Python programs can get certificates without manual operation.

CHAPTER 1

Prerequisites

The IIS server must listen on HTTPS with a valid (trusted by the client) certificate. It is recommended to enable basic auth on IIS, but NTLM is also supported through the requests_ntlm package

CHAPTER 2

Disclaimer

The certsrv page is not an API, so this is obviously a little hackish and a little fragile. It will break if Microsoft does any changes to the certsrv application.

Luckily (or sadly?) they haven't changed much in the last 19 years...

Certsrv has been tested against Windows 2008R2, 2012R2 and 2016, but I'm sure it works on everything from 2003 to 2019.

CHAPTER 3

GitHub

magnuswatn/certsrv

CHAPTER 4

Module Documentation

4.1 certsrv

A Python client for the Microsoft AD Certificate Services web page.

https://github.com/magnuswatn/certsrv

exception `certsrv.`**`CertificatePendingException`**(*req_id*)

Bases: `exceptions.Exception`

Signifies that the request needs to be approved by a CA admin.

class `certsrv.`**`Certsrv`**(*server*, *username*, *password*, *auth_method='basic'*, *cafile=None*, *timeout=30*)

Bases: `object`

Represents a Microsoft AD Certificate Services web server.

Parameters

- **`server`** – The FQDN to a server running the Certification Authority Web Enrollment role (must be listening on https).
- **`username`** – The username for authentication.
- **`password`** – The password for authentication.
- **`auth_method`** – The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (SSL client certificate).
- **`cafile`** – A PEM file containing the CA certificates that should be trusted.
- **`timeout`** – The timeout to use against the CA server, in seconds. The default is 30.

Note: If you use a client certificate for authentication (auth_method=cert), the username parameter should be the path to a certificate, and the password parameter the path to a (unencrypted) private key.

`check_credentials`()
Checks the specified credentials against the ADCS server.

Returns True if authentication succeeded, False if it failed.

`get_ca_cert`(*encoding='b64'*)
Gets the (newest) CA certificate from the ADCS server.

Parameters **`encoding`** – The desired encoding for the returned certificate. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).

Returns The newest CA certificate from the server.

`get_cert`(*csr*, *template*, *encoding='b64'*, *attributes=None*)
Gets a certificate from the ADCS server.

Parameters

- **`csr`** – The certificate request to submit.
- **`template`** – The certificate template the cert should be issued from.
- **`encoding`** – The desired encoding for the returned certificate. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).
- **`attributes`** – Additional Attributes (request attibutes) to be sent along with the request.

Returns The issued certificate.

Raises

- *`RequestDeniedException`* – If the request was denied by the ADCS server.
- *`CertificatePendingException`* – If the request needs to be approved by a CA admin.
- *`CouldNotRetrieveCertificateException`* – If something went wrong while fetching the cert.

`get_chain`(*encoding='bin'*)
Gets the CA chain from the ADCS server.

Parameters **`encoding`** – The desired encoding for the returned certificates. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).

Returns The CA chain from the server, in PKCS#7 format.

`get_existing_cert`(*req_id*, *encoding='b64'*)
Gets a certificate that has already been created from the ADCS server.

Parameters

- **`req_id`** – The request ID to retrieve.
- **`encoding`** – The desired encoding for the returned certificate. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).

Returns The issued certificate.

Raises *`CouldNotRetrieveCertificateException`* – If something went wrong while fetching the cert.

`update_credentials`(*username*, *password*)
Updates the credentials used against the ADCS server.

Parameters

- **username** – The username for authentication.
- **password** – The password for authentication.

exception certsrv.**CouldNotRetrieveCertificateException**(*message*, *response*)

Bases: `exceptions.Exception`

Signifies that the certificate could not be retrieved.

exception certsrv.**RequestDeniedException**(*message*, *response*)

Bases: `exceptions.Exception`

Signifies that the request was denied by the ADCS server.

certsrv.**check_credentials**(*server*, *username*, *password*, ***kwargs*)

Checks the specified credentials against the specified ADCS server.

Parameters

- **ca** – The FQDN to a server running the Certification Authority Web Enrollment role (must be listening on https).
- **username** – The username for authentication.
- **pasword** – The password for authentication.
- **auth_method** – The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (ssl client certificate).
- **cafile** – A PEM file containing the CA certificates that should be trusted.

Returns True if authentication succeeded, False if it failed.

Note: This method is deprecated.

certsrv.**get_ca_cert**(*server*, *username*, *password*, *encoding='b64'*, ***kwargs*)

Gets the (newest) CA certificate from a Microsoft AD Certificate Services web page.

Parameters

- **server** – The FQDN to a server running the Certification Authority Web Enrollment role (must be listening on https).
- **username** – The username for authentication.
- **pasword** – The password for authentication.
- **encoding** – The desired encoding for the returned certificate. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).
- **auth_method** – The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (ssl client certificate).
- **cafile** – A PEM file containing the CA certificates that should be trusted.

Returns The newest CA certificate from the server.

Note: This method is deprecated.

certsrv.**get_cert**(*server*, *csr*, *template*, *username*, *password*, *encoding='b64'*, ***kwargs*)

Gets a certificate from a Microsoft AD Certificate Services web page.

Parameters

- **server** – The FQDN to a server running the Certification Authority Web Enrollment role (must be listening on https).
- **csr** – The certificate request to submit.
- **template** – The certificate template the cert should be issued from.
- **username** – The username for authentication.
- **pasword** – The password for authentication.
- **encoding** – The desired encoding for the returned certificate. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).
- **auth_method** – The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (ssl client certificate).
- **cafile** – A PEM file containing the CA certificates that should be trusted.

Returns The issued certificate.

Raises

- *RequestDeniedException* – If the request was denied by the ADCS server.
- *CertificatePendingException* – If the request needs to be approved by a CA admin.
- *CouldNotRetrieveCertificateException* – If something went wrong while fetching the cert.

Note: This method is deprecated.

`certsrv.`**get_chain**(*server*, *username*, *password*, *encoding='bin'*, ***kwargs*)

Gets the chain from a Microsoft AD Certificate Services web page.

Parameters

- **server** – The FQDN to a server running the Certification Authority Web Enrollment role (must be listening on https).
- **username** – The username for authentication.
- **pasword** – The password for authentication.
- **encoding** – The desired encoding for the returned certificates. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).
- **auth_method** – The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (ssl client certificate).
- **cafile** – A PEM file containing the CA certificates that should be trusted.

Returns The CA chain from the server, in PKCS#7 format.

Note: This method is deprecated.

`certsrv.`**get_existing_cert**(*server*, *req_id*, *username*, *password*, *encoding='b64'*, ***kwargs*)

Gets a certificate that has already been created from a Microsoft AD Certificate Services web page.

Parameters

- **server** – The FQDN to a server running the Certification Authority Web Enrollment role (must be listening on https).
- **req_id** – The request ID to retrieve.
- **username** – The username for authentication.
- **pasword** – The password for authentication.
- **encoding** – The desired encoding for the returned certificate. Possible values are 'bin' for binary and 'b64' for Base64 (PEM).
- **auth_method** – The chosen authentication method. Either 'basic' (the default), 'ntlm' or 'cert' (ssl client certificate).
- **cafile** – A PEM file containing the CA certificates that should be trusted.

Returns The issued certificate.

Raises *CouldNotRetrieveCertificateException* – If something went wrong while fetching the cert.

Note: This method is deprecated.

4.2 Examples

Generate a CSR with Cryptography and get a cert from an ADCS server:

```
from certsrv import Certsrv

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import hashes

# Generate a key
key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
)

# Generate a CSR
csr = x509.CertificateSigningRequestBuilder().subject_name(x509.Name([
    x509.NameAttribute(NameOID.COMMON_NAME, u"myserver.example.com"),
])).add_extension(
    x509.SubjectAlternativeName([
        x509.DNSName(u"myserver.example.com"),
    ]),
    critical=False,
).sign(key, hashes.SHA256(), default_backend())

# Get the cert from the ADCS server
pem_req = csr.public_bytes(serialization.Encoding.PEM)
```

(continues on next page)

(continued from previous page)

```
ca_server = Certsrv("my-adcs-server.example.net", "myUser", "myPassword")
pem_cert = ca_server.get_cert(pem_req, "WebServer")

# Print the key and the cert
pem_key = key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption(),
)

print("Cert:\n{}".format(pem_cert.decode()))
print("Key:\n{}".format(pem_key.decode()))
```

Generate a CSR with pyOpenSSL and get a cert from an ADCS server:

```
import OpenSSL
from certsrv import Certsrv

# Generate a key
key = OpenSSL.crypto.PKey()
key.generate_key(OpenSSL.crypto.TYPE_RSA, 2048)

# Generate a CSR
req = OpenSSL.crypto.X509Req()
req.get_subject().CN="myserver.example.com"
san = b"DNS: myserver.example.com"
san_extension = OpenSSL.crypto.X509Extension(b"subjectAltName", False, san)
req.add_extensions([san_extension])

req.set_pubkey(key)
req.sign(key, "sha256")

# Get the cert from the ADCS server
pem_req = OpenSSL.crypto.dump_certificate_request(OpenSSL.crypto.FILETYPE_PEM, req)

ca_server = Certsrv("my-adcs-server.example.net", "myUser", "myPassword")
pem_cert = ca_server.get_cert(pem_req, "WebServer")

# Print the key and the cert
pem_key = OpenSSL.crypto.dump_privatekey(OpenSSL.crypto.FILETYPE_PEM, key)

print("Cert:\n{}".format(pem_cert.decode()))
print("Key:\n{}".format(pem_key.decode()))
```

Generate a CSR with pyOpenSSL and get a cert from an ADCS server with a template that requires admin approval:

```
import time
import OpenSSL
import certsrv

# Generate a key
key = OpenSSL.crypto.PKey()
key.generate_key(OpenSSL.crypto.TYPE_RSA, 2048)

# Generate a CSR
```

(continues on next page)

(continued from previous page)

```
req = OpenSSL.crypto.X509Req()
req.get_subject().CN="myserver.example.com"
san = b"DNS: myserver.example.com"
san_extension = OpenSSL.crypto.X509Extension(b"subjectAltName", False, san)
req.add_extensions([san_extension])

req.set_pubkey(key)
req.sign(key, "sha256")

# Get the cert from the ADCS server
ca_server = certsrv.Certsrv("my-adcs-server.example.net", "myUser", "myPassword")
pem_req = OpenSSL.crypto.dump_certificate_request(OpenSSL.crypto.FILETYPE_PEM, req)

try:
    pem_cert = ca_server.get_cert(pem_req, "WebServerManual")
except certsrv.CertificatePendingException as error:
    print("The request needs to be approved by the CA admin."
          "The Request Id is {}. She has a minute to approve it...".format(error.req_
↪id))
    time.sleep(60)
    pem_cert = ca_server.get_existing_cert(error.req_id)

# Print the key and the cert
pem_key = OpenSSL.crypto.dump_privatekey(OpenSSL.crypto.FILETYPE_PEM, key)

print("Cert:\n{}".format(pem_cert.decode()))
print("Key:\n{}".format(pem_key.decode()))
```

Python Module Index

Index