MyWebUniversity
python
Red Hat Enterprise Linux
Manual Pages
FPDF Library
PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'sss-certmap.5' command

$ man sss-certmap.5

SSS-CERTMAP(5)   File Formats and Conventions   SSS-CERTMAP(5)

NAME

sss-certmap - SSSD Certificate Matching and Mapping Rules

DESCRIPTION

The manual page describes the rules which can be used by SSSD and other components to match X.509 certificates and map them to accounts.

Each rule has four components, a ?priority?, a ?matching rule?, a ?mapping rule? and a ?domain list?. All components are optional. A missing ?priority? will add the rule with the lowest priority. The default ?matching rule? will match certificates with the digitalSignature key usage and clientAuth extended key usage. If the ?mapping rule? is empty the certificates will be searched in the userCertificate attribute as DER encoded binary. If no domains are given only the local domain will be searched.

To allow extensions or completely different style of rule the ?mapping? and ?matching rules? can contain a prefix separated with a ':' from the main part of the rule. The prefix may only contain upper-case ASCII letters and numbers. If the prefix is omitted the default type will be used which is 'KRB5' for the matching rules and 'LDAP' for the mapping rules.

The 'sssctl' utility provides the 'cert-eval-rule' command to check if a given certificate matches a matching rules and how the output of a mapping rule would look like.

RULE COMPONENTS

PRIORITY

The rules are processed by priority while the number '0' (zero) indicates the highest priority. The higher the number the lower is the priority. A missing value indicates the lowest priority. The rules processing is stopped when a matched rule is found and no further rules are checked.

Internally the priority is treated as unsigned 32bit integer, using a priority value larger than 4294967295 will cause an error.

If multiple rules have the same priority and only one of the related matching rules applies, this rule will be chosen. If there are multiple rules with the same priority which matches, one is chosen but which one is undefined. To avoid this undefined behavior either use different priorities or make the matching rules more specific e.g. by using distinct <ISSUER> patterns.

MATCHING RULE

The matching rule is used to select a certificate to which the mapping rule should be applied. It uses a system similar to the one used by ?pkinit_cert_match? option of MIT Kerberos. It consists of a keyword enclosed by '<' and '>' which identified a certain part of the certificate and a pattern which should be found for the rule to match. Multiple keyword pattern pairs can be either joined with '&&' (and) or '||' (or).

Given the similarity to MIT Kerberos the type prefix for this rule is 'KRB5'. But 'KRB5' will also be the default for ?matching rules? so that "<SUBJECT>.*,DC=MY,DC=DOMAIN" and "KRB5:<SUBJECT>.*,DC=MY,DC=DOMAIN" are equivalent.

The available options are:

<SUBJECT>regular-expression

With this a part or the whole subject name of the certificate can be matched. For the matching POSIX Extended Regular Expression syntax is used, see regex(7) for details.

For the matching the subject name stored in the certificate in DER encoded ASN.1 is converted into a string according to RFC 4514. This means the most specific name component comes first. Please note that not all possible attribute names are covered by RFC 4514. The names included are 'CN', 'L', 'ST', 'O', 'OU', 'C', 'STREET', 'DC' and 'UID'. Other attribute names might be shown differently on different platform and by different tools. To avoid confusion those attribute names are best not used or covered by a suitable regular-expression.

Example: <SUBJECT>.*,DC=MY,DC=DOMAIN

Please note that the characters "^.[$()|*+?{\" have a special meaning in regular expressions and must be escaped with the help of the '\' character so that they are matched as ordinary characters.

Example: <SUBJECT>^CN=.* \(Admin\),DC=MY,DC=DOMAIN$

<ISSUER>regular-expression

With this a part or the whole issuer name of the certificate can be matched. All comments for <SUBJECT> apply her as well.

Example: <ISSUER>^CN=My-CA,DC=MY,DC=DOMAIN$

<KU>key-usage

This option can be used to specify which key usage values the certificate should have. The following values can be used in a comma separated list:

? digitalSignature

? nonRepudiation

? keyEncipherment

? dataEncipherment

? keyAgreement

? keyCertSign

? cRLSign

? encipherOnly

? decipherOnly

A numerical value in the range of a 32bit unsigned integer can be used as well to cover special use cases.

Example: <KU>digitalSignature,keyEncipherment

<EKU>extended-key-usage

This option can be used to specify which extended key usage the certificate should have. The following value can be used in a comma separated list:

? serverAuth

? clientAuth

? codeSigning

? emailProtection

? timeStamping

? OCSPSigning

? KPClientAuth

? pkinit

? msScLogin

Extended key usages which are not listed above can be specified with their OID in dotted-decimal notation.

Example: <EKU>clientAuth,1.3.6.1.5.2.3.4

<SAN>regular-expression

To be compatible with the usage of MIT Kerberos this option will match the Kerberos principals in the PKINIT or AD NT Principal SAN as <SAN:Principal> does.

Example: <SAN>.*@MY\.REALM

<SAN:Principal>regular-expression

Match the Kerberos principals in the PKINIT or AD NT Principal SAN.

Example: <SAN:Principal>.*@MY\.REALM

<SAN:ntPrincipalName>regular-expression

Match the Kerberos principals from the AD NT Principal SAN.

Example: <SAN:ntPrincipalName>.*@MY.AD.REALM

<SAN:pkinit>regular-expression

Match the Kerberos principals from the PKINIT SAN.

Example: <SAN:ntPrincipalName>.*@MY\.PKINIT\.REALM

<SAN:dotted-decimal-oid>regular-expression

Take the value of the otherName SAN component given by the OID in dotted-decimal notation, interpret it as string and try to match it against the regular expression.

Example: <SAN:1.2.3.4>test

<SAN:otherName>base64-string

Do a binary match with the base64 encoded blob against all otherName SAN components. With this option it is possible to match against custom otherName components with special encodings which could not be treated as strings.

Example: <SAN:otherName>MTIz

<SAN:rfc822Name>regular-expression

Match the value of the rfc822Name SAN.

Example: <SAN:rfc822Name>.*@email\.domain

<SAN:dNSName>regular-expression

Match the value of the dNSName SAN.

Example: <SAN:dNSName>.*\.my\.dns\.domain

<SAN:x400Address>base64-string

Binary match the value of the x400Address SAN.

Example: <SAN:x400Address>MTIz

<SAN:directoryName>regular-expression

Match the value of the directoryName SAN. The same comments as given for <ISSUER> and <SUBJECT> apply here as well.

Example: <SAN:directoryName>.*,DC=com

<SAN:ediPartyName>base64-string

Binary match the value of the ediPartyName SAN.

Example: <SAN:ediPartyName>MTIz

<SAN:uniformResourceIdentifier>regular-expression

Match the value of the uniformResourceIdentifier SAN.

Example: <SAN:uniformResourceIdentifier>URN:.*

<SAN:iPAddress>regular-expression

Match the value of the iPAddress SAN.

Example: <SAN:iPAddress>192\.168\..*

<SAN:registeredID>regular-expression

Match the value of the registeredID SAN as dotted-decimal string.

Example: <SAN:registeredID>1\.2\.3\..*

MAPPING RULE

The mapping rule is used to associate a certificate with one or more accounts. A Smartcard with the certificate and the matching private key can then be used to authenticate as one of those accounts.

Currently SSSD basically only supports LDAP to lookup user information (the exception is the proxy provider which is not of relevance here). Because of this the mapping rule is based on LDAP search filter syntax with templates to add certificate content to the filter. It is expected that the filter will only contain the specific data needed for the mapping and that the caller will embed it in another filter to do the actual search. Because of this the filter string should start and stop with '(' and ')' respectively.

In general it is recommended to use attributes from the certificate and add them to special attributes to the LDAP user object. E.g. the 'altSecurityIdentities' attribute in AD or the 'ipaCertMapData' attribute for IPA can be used.

This should be preferred to read user specific data from the certificate like e.g. an email address and search for it in the LDAP server. The reason is that the user specific data in LDAP might change for various reasons would break the mapping. On the other hand it would be hard to break the mapping on purpose for a specific user.

The default ?mapping rule? type is 'LDAP' which can be added as a prefix to a rule like e.g. 'LDAP:(userCertificate;binary={cert!bin})'. There is an extension called 'LDAPU1' which offer more templates for more flexibility. To allow older versions of this library to ignore the extension the prefix 'LDAPU1' must be used when using the new templates in a ?mapping rule? otherwise the old version of this library will fail with a parsing error. The new templates are described in section the section called ?LDAPU1 extension?.

The templates to add certificate data to the search filter are based on Python-style formatting strings. They consist of a keyword in curly braces with an optional sub-component specifier separated by a '.' or an optional conversion/formatting option separated by a '!'. Allowed values are:

{issuer_dn[!((ad|ad_x500)|ad_ldap|nss_x500|(nss|nss_ldap))]}

This template will add the full issuer DN converted to a string according to RFC 4514. If X.500 ordering (most specific RDN comes last) an option with the '_x500' prefix should be used.

The conversion options starting with 'ad_' will use attribute names as used by AD, e.g. 'S' instead of 'ST'.

The conversion options starting with 'nss_' will use attribute names as used by NSS.

The default conversion option is 'nss', i.e. attribute names according to NSS and LDAP/RFC 4514 ordering.

Example: (ipacertmapdata=X509:<I>{issuer_dn!ad}<S>{subject_dn!ad})

{subject_dn[!((ad|ad_x500)|ad_ldap|nss_x500|(nss|nss_ldap))]}

This template will add the full subject DN converted to string according to RFC 4514. If X.500 ordering (most specific RDN comes last) an option with the '_x500' prefix should be used.

The conversion options starting with 'ad_' will use attribute names as used by AD, e.g. 'S' instead of 'ST'.

The conversion options starting with 'nss_' will use attribute names as used by NSS.

The default conversion option is 'nss', i.e. attribute names according to NSS and LDAP/RFC 4514 ordering.

Example:

(ipacertmapdata=X509:<I>{issuer_dn!nss_x500}<S>{subject_dn!nss_x500})

{cert[!(bin|base64)]}

This template will add the whole DER encoded certificate as a string to the search filter. Depending on the conversion option the binary certificate is either converted to an escaped hex sequence '\xx' or base64. The escaped hex sequence is the default and can e.g. be used with the LDAP attribute 'userCertificate;binary'.

Example: (userCertificate;binary={cert!bin})

{subject_principal[.short_name]}

This template will add the Kerberos principal which is taken either from the SAN used by pkinit or the one used by AD. The 'short_name' component represents the first part of the principal before the '@' sign.

Example:

(|(userPrincipal={subject_principal})(samAccountName={subject_principal.short_name}))

{subject_pkinit_principal[.short_name]}

This template will add the Kerberos principal which is given by the SAN used by pkinit. The 'short_name' component represents the first part of the principal before the '@' sign.

Example:

(|(userPrincipal={subject_pkinit_principal})(uid={subject_pkinit_principal.short_name}))

{subject_nt_principal[.short_name]}

This template will add the Kerberos principal which is given by the SAN used by AD. The 'short_name' component represent the first part of the principal before the '@' sign.

Example:

(|(userPrincipalName={subject_nt_principal})(samAccountName={subject_nt_principal.short_name}))

{subject_rfc822_name[.short_name]}

This template will add the string which is stored in the rfc822Name component of the SAN, typically an email address. The 'short_name' component represents the first part of the address before the '@' sign.

Example:

(|(mail={subject_rfc822_name})(uid={subject_rfc822_name.short_name}))

{subject_dns_name[.short_name]}

This template will add the string which is stored in the dNSName component of the SAN, typically a fully-qualified host name. The 'short_name' component represents the first part of the name before the first '.' sign.

Example:

(|(fqdn={subject_dns_name})(host={subject_dns_name.short_name}))

{subject_uri}

This template will add the string which is stored in the uniformResourceIdentifier component of the SAN.

Example: (uri={subject_uri})

{subject_ip_address}

This template will add the string which is stored in the iPAddress component of the SAN.

Example: (ip={subject_ip_address})

{subject_x400_address}

This template will add the value which is stored in the x400Address component of the SAN as escaped hex sequence.

Example: (attr:binary={subject_x400_address})

{subject_directory_name[!((ad|ad_x500)|ad_ldap|nss_x500|(nss|nss_ldap))]}

This template will add the DN string of the value which is stored in the directoryName component of the SAN.

Example: (orig_dn={subject_directory_name})

{subject_ediparty_name}

This template will add the value which is stored in the ediPartyName component of the SAN as escaped hex sequence.

Example: (attr:binary={subject_ediparty_name})

{subject_registered_id}

This template will add the OID which is stored in the registeredID component of the SAN as a dotted-decimal string.

Example: (oid={subject_registered_id})

LDAPU1 extension

The following template are available when using the 'LDAPU1' extension:

{serial_number[!(dec|hex[_ucr])]}

This template will add the serial number of the certificate. By default it will be printed as a hexadecimal number with lower-case letters.

With the formatting option '!dec' the number will be printed as decimal string. The hexadecimal output can be printed with upper-case letters ('!hex_u'), with a colon separating the hexadecimal bytes ('!hex_c') or with the hexadecimal bytes in reverse order ('!hex_r'). The postfix letters can be combined so that e.g. '!hex_uc' will produce a colon-separated hexadecimal string with upper-case letters.

Example: LDAPU1:(serial={serial_number})

{subject_key_id[!hex[_ucr]]}

This template will add the subject key id of the certificate.

By default it will be printed as a hexadecimal number with lower-case letters.

The hexadecimal output can be printed with upper-case letters ('!hex_u'), with a colon separating the hexadecimal bytes ('!hex_c') or with the hexadecimal bytes in reverse order ('!hex_r'). The postfix letters can be combined so that e.g. '!hex_uc' will produce a colon-separated hexadecimal string with upper-case letters.

Example: LDAPU1:(ski={subject_key_id})

{cert[!DIGEST[_ucr]]}

This template will add the hexadecimal digest/hash of the certificate where DIGEST must be replaced with the name of a digest/hash function supported by OpenSSL, e.g. 'sha512'.

The hexadecimal output can be printed with upper-case letters ('!sha512_u'), with a colon separating the hexadecimal bytes ('!sha512_c') or with the hexadecimal bytes in reverse order ('!sha512_r'). The postfix letters can be combined so that e.g. '!sha512_uc' will produce a colon-separated hexadecimal string with upper-case letters.

Example: LDAPU1:(dgst={cert!sha256})

{subject_dn_component[(.attr_name|[number]]}

This template will add an attribute value of a component of the subject DN, by default the value of the most specific component.

A different component can it either selected by attribute name, e.g. {subject_dn_component.uid} or by position, e.g. {subject_dn_component.[2]} where positive numbers start counting from the most specific component and negative numbers start counting from the least specific component. Attribute name and the position can be combined as e.g. {subject_dn_component.uid[2]} which means that the name of the second component must be 'uid'.

Example: LDAPU1:(uid={subject_dn_component.uid})

{issuer_dn_component[(.attr_name|[number]]}

This template will add an attribute value of a component of the issuer DN, by default the value of the most specific component. See 'subject_dn_component' for details about the attribute name and position specifiers.

Example:

LDAPU1:(domain={issuer_dn_component.[-2]}.{issuer_dn_component.dc[-1]})

{sid[.rid]}

This template will add the SID if the corresponding extension introduced by Microsoft with the OID 1.3.6.1.4.1.311.25.2 is available. With the '.rid' selector only the last component, i.e. the RID, will be added.

Example: LDAPU1:(objectsid={sid})

DOMAIN LIST

If the domain list is not empty users mapped to a given certificate are not only searched in the local domain but in the listed domains as well as long as they are know by SSSD. Domains not know to SSSD will be ignored.

AUTHORS

The SSSD upstream - https://github.com/SSSD/sssd/

SSSD 07/10/2023 SSS-CERTMAP(5)