Перевыпуск сертификатов vCenter 6.5

Полезные скрипты

```
"C:\Program Files\VMware\vCenter Server\python\python.exe" .checksts.py
```

Код для проверки срока действия сертификатов:

```
$VCInstallHome =
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_CIS_HOME%")
foreach ($STORE in & "$VCInstallHome\vmafdd\vecs-cli" store list){
  Write-host STORE: $STORE
  & "$VCInstallHome\vmafdd\vecs-cli" entry list --store $STORE --text |
findstr /C:"Alias" /C:"Not After"
}
```

Перевыпуск сертификатов SSL Machine и сервисов

```
cd "C:\Program Files\VMware\vCenter Server\vmcad"
Certificate-manager.bat
```

Выбрать пункт 6 *6. Replace Solution user certificates with VMCA certificates*

STS Certificate

Если просрочен STS сертификат:

```
.\fixsts.ps1
```

Скрипты

fixsts.ps1y

fixsts.ps1

```
#generate new certificate and key

$VCInstallHome =
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_CIS_HOME%")
$VCConfigHome =
```

```
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_CFG_DIR%")
$VCJavaHome =
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_JAVA_HOME%")
$VCJavaHome = $VCJavaHome.TrimEnd('\')
$VCLogHome =
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_LOG_DIR%")
$VCDataHome =
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_DATA_DIR%")
$VCRuntimeDataHome =
[System.Environment]::ExpandEnvironmentVariables("%VMWARE_RUNTIME_DATA_
DIR%")
$VMWARE_IDENTITY_SERVICES_HOME = "$VCInstallHome\VMware Identity
Services"
$VMWARE_STS_SERVICES_HOME =
"$VCRuntimeDataHome\VMwareSTSService\webapps\ROOT\WEB-INF\lib"

[int]$VCWinBuild = (Get-ItemProperty -Path "HKLM:\SOFTWARE\VMware,
Inc.\vCenter Server" -name BuildNumber).BuildNumber
$VCWinProduct = (Get-ItemProperty -Path "HKLM:\SOFTWARE\VMware,
Inc.\vCenter Server" -name ProductVersion).ProductVersion
$DeploymentType = (Get-ItemProperty -Path "HKLM:\SOFTWARE\VMware,
Inc.\vCenter Server" -name INSTALL_TYPE).INSTALL_TYPE

$VMCAReplaced = $false
$ScriptPath = $PSScriptRoot

if ($DeploymentType -eq "management")
{
    Write-Host "This vCenter Server is installed as $($DeploymentType)
Node, please execute the script on Platform Service Controller or
Embedded PSC Node, exiting.."
    Exit
}

$PNID = & "$VCInstallHome\vmafdd\vmafd-cli" get-pnid --server-name
localhost

$CRT = New-Object
System.Security.Cryptography.X509Certificates.X509Certificate
$CRT.Import("$VCDataHome\vmca\root.cer")
$expdate = ($CRT.GetExpirationDateString() | get-date)
if ($expdate -lt $(get-date))
{
    Write-Host "VMCA Root Cert is Expired, replacing Root Certificate"
    & "$VCInstallHome\vmcad\certool.exe" --genselfcacert --outprivkey
"C:\Windows\temp\vmcacert.key" --outcert "C:\windows\temp\vmcacert.crt"
--config="$VCInstallHome\vmcad\certool.cfg" --Name="VMCA" --
Hostname=$PNID
    & "$VCInstallHome\vmcad\certool.exe" --rootca --privkey
"C:\Windows\temp\vmcacert.key" --cert "C:\windows\temp\vmcacert.crt"
    $VMCAReplaced = $True
```

```
}

Write-Host "Generating New STS Certificate"
& "$VCInstallHome\vmcad\certool.exe" --genkey --
privkey="C:\Windows\temp\new-ssoserverSign.key" --
pubkey="C:\Windows\temp\new-ssoserverSign.pub"
& "$VCInstallHome\vmcad\certool.exe" --gencert --
priv="C:\Windows\temp\new-ssoserverSign.key" --Name="ssoserverSign" --
cert="C:\Windows\temp\new-ssoserverSign.crt" --
config="$VCInstallHome\vmcad\certool.cfg" --Hostname=$PNID --
server="localhost"


#Load the assemblies
[void]
[System.Reflection.Assembly]::LoadWithPartialName("System.DirectoryServ
ices.Protocols")
[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Net")


#Connects to myopenldap.mikesblog.lan using SSL on a non-standard port
$connection = New-Object
System.DirectoryServices.Protocols.LdapConnection "localhost:389"

#Set session options
$connection.SessionOptions.SecureSocketLayer = $false
$connection.SessionOptions.ProtocolVersion = 3

# Pick Authentication type:
# Anonymous, Basic, Digest, DPA (Distributed Password Authentication),
# External, Kerberos, Msn, Negotiate, Ntlm, Sicily
$connection.AuthType =
[System.DirectoryServices.Protocols.AuthType]::Basic
# Gets username and password.


$sso_domain = &"$Env:VMWARE_CIS_HOME\vmafdd\vmafd-cli.exe" get-domain-
name --server-name localhost
$sso_domain_DN = $sso_domain -replace "\.",",dc="

# Gets username and password.
$credential_prompt = Get-Credential -Message "Enter the Password for
the Single Sign-On Administrator account: " -UserName
"administrator@$sso_domain"
if ($VCWinProduct -like "6.5.*")
{
    $STSReplaceCMD = '& "$VCJavaHome\bin\java.exe" -cp
"$VCInstallHome\VMware Identity Services\*;$VCInstallHome\vmware-
sso\commonlib\*;.;*" "-Dvmware.log.dir=$VCLogHome\sso\" -
XX:ErrorFile=$VCLogHome\sso\hs_err_stsinstaller_pid%p.log -
XX:HeapDumpPath=$VCLogHome\sso\
```

```
com.vmware.identity.installer.STSInstaller --install --root-cert-path
"$VCDataHome\vmca\root.cer" --cert-path "C:\Windows\temp\new-
ssoserverSign.crt" --private-key-path "C:\windows\temp\new-
ssoserverSign.key" --retry-count 10 --retry-interval 30'
}
elseif ($VCWinProduct -like "6.7.*")
{
    get-content "C:\Windows\temp\new-ssoserverSign.crt",
"$VCDataHome\vmca\root.cer", "C:\Windows\temp\new-ssoserverSign.key" |
set-content "C:\Windows\temp\newsts.pem"

if ($VCWinBuild -ge 16046470)
{
    $STSReplaceCMD = '& "$VCInstallHome\VMware Identity Services\sso-
config.bat" -set_signing_cert -t $sso_domain
C:\Windows\temp\newsts.pem'
}
else
{
    if (Test-Path $VMWARE_IDENTITY_SERVICES_HOME\vmware-identity-sso-
config67u3g.jar)
        {
            $STSReplaceCMD = '& "$VCJavaHome\bin\java" -cp
"$VMWARE_IDENTITY_SERVICES_HOME\vmware-identity-sso-
config67u3g.jar;$VMWARE_IDENTITY_SERVICES_HOME\*;$VMWARE_STS_SERVICES_H
OME\*;$VCInstallHome\vmware-sso\commonlib\commons-cli-1.2.jar" "-
Dlog4j.configurationFile=file://$VMWARE_IDENTITY_SERVICES_HOME\ssoconfi
g.log4j2.xml" "-Dvmware.log.dir=$VCLogHome\sso\"
com.vmware.identity.ssoconfig.SsoConfig -set_signing_cert -t
$sso_domain C:\Windows\temp\newsts.pem'
        }
    elseif (Test-Path $ScriptPath\vmware-identity-sso-config67u3g.jar)
        {
            $STSReplaceCMD = '& "$VCJavaHome\bin\java" -cp
"$ScriptPath\vmware-identity-sso-
config67u3g.jar;$VMWARE_IDENTITY_SERVICES_HOME\*;$VMWARE_STS_SERVICES_H
OME\*;$VCInstallHome\vmware-sso\commonlib\commons-cli-1.2.jar" "-
Dlog4j.configurationFile=file://$VMWARE_IDENTITY_SERVICES_HOME\ssoconfi
g.log4j2.xml" "-Dvmware.log.dir=$VCLogHome\sso\"
com.vmware.identity.ssoconfig.SsoConfig -set_signing_cert -t
$sso_domain C:\Windows\temp\newsts.pem'
        }
    else
        {
            Write-Host "JAR file # vmware-identity-sso-config67u3g.jar
# does not exist in the script path or in # $($VCInstallHome)\VMware
Identity Services #, please copy the file attached in the KB to any of
these locations and retry.."
            Exit
        }
}
```

```
}
else
{
    write-host "This Script is not supported on this vCenter Server
Version $VCWinProduct Build $VCWinBuild, exiting.."
    exit
}

if($credential_prompt) {
   $user = $credential_prompt.Username.split('@')[0]
   $username = "cn=$user,cn=users,dc=$sso_domain_DN"
   Write-Host "User DN is: $username"
   #$password = [System.Net.NetworkCredential]::new("",
$credential_prompt.Password).Password
   $password = $credential_prompt.GetNetworkCredential().password
   $credentials = New-Object "System.Net.NetworkCredential" -
ArgumentList $username,$password

   # Bind with the network credentials. Depending on the type of
server,
   # the username will take different forms. Authentication type is
controlled
   # above with the AuthType
   $connection.Bind($credentials)

   $search_DN =
"cn=$sso_domain,cn=Tenants,cn=IdentityManager,cn=Services,dc=$sso_domai
n_DN"
   $search_filter =
"(|(objectclass=vmwSTSTenantCredential)(objectclass=vmwSTSTenantTrusted
CertificateChain))"
   $search_scope =
[System.DirectoryServices.Protocols.SearchScope]::Subtree
   $search_attribute = @('*')

   $search_request = New-Object
System.DirectoryServices.Protocols.SearchRequest -ArgumentList
$search_DN,$search_filter,$search_scope,$search_attribute

   #Actually process the request through the server
   $search_request_result = $connection.SendRequest($search_request)

   if ($search_request_result.ResultCode -ne
[System.directoryServices.Protocols.ResultCode]::Success) {
      Write-Host "Failed!"
      Write-Host ("ResultCode: " + $search_request_result.ResultCode)
      Write-Host ("Message: " + $search_request_result.ErrorMessage)
   } else {
      $delete_success = $true
      $search_results = @{}
```

```
      foreach ($branch in $search_request_result.Entries) {
         $result_DN = $branch.DistinguishedName
         [regex]$regex = 'cn='
         $CN_count = $regex.matches($result_DN).count

         if (! $search_results.ContainsKey($CN_count) ) {
            $search_results[$CN_count] = @()
         }

         $search_results[$CN_count] += $branch.DistinguishedName
      }

      foreach ($CN_counter in $search_results.GetEnumerator()) {
         foreach ($dn in $($CN_counter.Value)) {
            $delete_request = New-Object
System.DirectoryServices.Protocols.DeleteRequest
            $delete_request.DistinguishedName = $dn

            $delete_request_result =
$connection.SendRequest($delete_request)

            if($delete_request_result.ResultCode -ne
[System.directoryServices.Protocols.ResultCode]::Success) {
               Write-Host "Failed!"
               Write-Host ("ResultCode: " +
$delete_request_result.ResultCode)
               Write-Host ("Message: " +
$delete_request_result.ErrorMessage)
               $delete_success = $false
            } else {
               Write-Host "Successfully deleted $dn"
            }
         }
      }
      if($delete_success) {
         Write-Host "vCenter Server Version is $($VCWinProduct) Build
$($VCWinBuild)"
         Write-Host "All STS Tenant branches deleted!"
         Write-Host "`nRe-creating STS tenant"
         $ErrorActionPreference = 'SilentlyContinue'
         $result = Invoke-Expression $STSReplaceCMD
         if ( ($result -like "*Successfully installed*") -or ($result -
eq $null) )
         {
            Write-Host "STS Certificate Replaced Successfully!!, please
restart the services"
         }
         elseif($result)
         {
            Write-Host "STS Certificate Replacement Failed!!, with
error - $result"
```

```
         }
         if($VMCAReplaced)
         {
            Write-Host "VMCA Certificate is replaced by the script,
please follow https://kb.vmware.com/s/article/2097936 to replace
Machine SSL and Solution User Certificates if those are expired"
         }
         Write-host "Since the STS certificate has been replaced, you
may need to re-register external solutions (SRM, NSX, etc.)"
      }
   }
}
```

checklist.py

checklist.py

```
#!/opt/vmware/bin/python


"""
Copyright 2020-2022 VMware, Inc.  All rights reserved. -- VMware
Confidential
Author:  Keenan Matheny (keenanm@vmware.com)

"""
##### BEGIN IMPORTS #####

import os
import sys
import json
import subprocess
import re
import pprint
import ssl
from datetime import datetime, timedelta
import textwrap
from codecs import encode, decode
import subprocess
from time import sleep
try:
    # Python 3 hack.
    import urllib.request as urllib2
    import urllib.parse as urlparse
except ImportError:
    import urllib2
    import urlparse
```

```
sys.path.append(os.environ['VMWARE_PYTHON_PATH'])
from cis.defaults import def_by_os
sys.path.append(os.path.join(os.environ['VMWARE_CIS_HOME'],
                def_by_os('vmware-vmafd/lib64', 'vmafdd')))
import vmafd
from OpenSSL.crypto import (load_certificate, dump_privatekey,
dump_certificate, X509, X509Name, PKey)
from OpenSSL.crypto import (TYPE_DSA, TYPE_RSA, FILETYPE_PEM,
FILETYPE_ASN1 )

today = datetime.now()
today = today.strftime("%d-%m-%Y")

vcsa_kblink = "https://kb.vmware.com/s/article/76719"
win_kblink = "https://kb.vmware.com/s/article/79263"

##### END IMPORTS #####

class parseCert( object ):
    # Certificate parsing

    def format_subject_issuer(self, x509name):
        items = []
        for item in x509name.get_components():
            items.append('%s=%s' %  (decode(item[0],'utf-8'),
decode(item[1],'utf-8')))
        return ", ".join(items)

    def format_asn1_date(self, d):
        return datetime.strptime(decode(d,'utf-8'),
'%Y%m%d%H%M%SZ').strftime("%Y-%m-%d %H:%M:%S GMT")

    def merge_cert(self, extensions, certificate):
        z = certificate.copy()
        z.update(extensions)
        return z

    def __init__(self, certdata):

        built_cert = certdata
        self.x509 = load_certificate(FILETYPE_PEM, built_cert)
        keytype = self.x509.get_pubkey().type()
        keytype_list = {TYPE_RSA:'rsaEncryption',
TYPE_DSA:'dsaEncryption', 408:'id-ecPublicKey'}
        extension_list = ["extendedKeyUsage",
                          "keyUsage",
                          "subjectAltName",
                          "subjectKeyIdentifier",
                          "authorityKeyIdentifier"]
        key_type_str = keytype_list[keytype] if keytype in keytype_list
else 'other'
```

```
        certificate = {}
        extension = {}
        for i in range(self.x509.get_extension_count()):
            critical = 'critical' if
self.x509.get_extension(i).get_critical() else ''

            if
decode(self.x509.get_extension(i).get_short_name(),'utf-8') in
extension_list:
extension[decode(self.x509.get_extension(i).get_short_name(),'utf-8')]
= self.x509.get_extension(i).__str__()

        certificate = {'Thumbprint':
decode(self.x509.digest('sha1'),'utf-8'), 'Version':
self.x509.get_version(),
         'SignatureAlg' :
decode(self.x509.get_signature_algorithm(),'utf-8'), 'Issuer'
:self.format_subject_issuer(self.x509.get_issuer()),
         'Valid From' :
self.format_asn1_date(self.x509.get_notBefore()), 'Valid Until' :
self.format_asn1_date(self.x509.get_notAfter()),
         'Subject' :
self.format_subject_issuer(self.x509.get_subject())}

        combined = self.merge_cert(extension,certificate)
        cert_output = json.dumps(combined)

        self.subjectAltName = combined.get('subjectAltName')
        self.subject = combined.get('Subject')
        self.validfrom = combined.get('Valid From')
        self.validuntil = combined.get('Valid Until')
        self.thumbprint = combined.get('Thumbprint')
        self.subjectkey = combined.get('subjectKeyIdentifier')
        self.authkey = combined.get('authorityKeyIdentifier')
        self.combined = combined

class parseSts( object ):

    def __init__(self):
        self.processed = []
        self.results = {}
        self.results['expired'] = {}
        self.results['expired']['root'] = []
        self.results['expired']['leaf'] = []
        self.results['valid'] = {}
        self.results['valid']['root'] = []
        self.results['valid']['leaf'] = []

    def get_certs(self,force_refresh):
        urllib2.getproxies = lambda: {}
```

```
        vmafd_client = vmafd.client('localhost')
        domain_name = vmafd_client.GetDomainName()

        dc_name = vmafd_client.GetAffinitizedDC(domain_name,
force_refresh)
        if vmafd_client.GetPNID() == dc_name:
            url = (
'http://localhost:7080/idm/tenant/%s/certificates?scope=TENANT'
                % domain_name)
        else:
            url = (
                'https://%s/idm/tenant/%s/certificates?scope=TENANT'
                % (dc_name,domain_name))
        return json.loads(urllib2.urlopen(url).read().decode('utf-8'))

    def check_cert(self,certificate):
        cert = parseCert(certificate)
        certdetail = cert.combined

            #  Attempt to identify what type of certificate it is
        if cert.authkey:
            cert_type = "leaf"
        else:
            cert_type = "root"

        #  Try to only process a cert once
        if cert.thumbprint not in self.processed:
            # Date conversion
            self.processed.append(cert.thumbprint)
            exp = cert.validuntil.split()[0]
            conv_exp = datetime.strptime(exp, '%Y-%m-%d')
            exp = datetime.strftime(conv_exp, '%d-%m-%Y')
            now = datetime.strptime(today, '%d-%m-%Y')
            exp_date = datetime.strptime(exp, '%d-%m-%Y')

            # Get number of days until it expires
            diff = exp_date - now
            certdetail['daysUntil'] = diff.days

            # Sort expired certs into leafs and roots, put the rest in
goodcerts.
            if exp_date <= now:
                self.results['expired'][cert_type].append(certdetail)
            else:
                self.results['valid'][cert_type].append(certdetail)

    def execute(self):

        json = self.get_certs(force_refresh=False)
        for item in json:
            for certificate in item['certificates']:
```

```
                self.check_cert(certificate['encoded'])
        return self.results

def main():

    warning = False
    warningmsg = '''
    WARNING!
    You have expired STS certificates.  Please follow the KB
corresponding to your OS:
    VCSA:  %s
    Windows:  %s
    ''' % (vcsa_kblink, win_kblink)
    parse_sts = parseSts()
    results = parse_sts.execute()
    valid_count = len(results['valid']['leaf']) +
len(results['valid']['root'])
    expired_count = len(results['expired']['leaf']) +
len(results['expired']['root'])


    #### Display Valid ####
    print("\n%s VALID CERTS\n================" % valid_count)
    print("\n\tLEAF CERTS:\n")
    if len(results['valid']['leaf']) > 0:
        for cert in results['valid']['leaf']:
            print("\t[] Certificate %s will expire in %s days (%s
years)." % (cert['Thumbprint'], cert['daysUntil'],
round(cert['daysUntil']/365)))
    else:
        print("\tNone")
    print("\n\tROOT CERTS:\n")
    if len(results['valid']['root']) > 0:
        for cert in results['valid']['root']:
            print("\t[] Certificate %s will expire in %s days (%s
years)." % (cert['Thumbprint'], cert['daysUntil'],
round(cert['daysUntil']/365)))
    else:
        print("\tNone")


    #### Display expired ####
    print("\n%s EXPIRED CERTS\n================" % expired_count)
    print("\n\tLEAF CERTS:\n")
    if len(results['expired']['leaf']) > 0:
        for cert in results['expired']['leaf']:
            print("\t[] Certificate: %s expired on %s!" %
(cert.get('Thumbprint'),cert.get('Valid Until')))
            continue
    else:
        print("\tNone")
```

```
    print("\n\tROOT CERTS:\n")
    if len(results['expired']['root']) > 0:
        for cert in results['expired']['root']:
            print("\t[] Certificate: %s expired on %s!" %
(cert.get('Thumbprint'),cert.get('Valid Until')))
            continue
    else:
        print("\tNone")

    if expired_count > 0:
        print(warningmsg)


if __name__ == '__main__':
    exit(main())
```