MichaelGrafnetter Improved DPAPI credential parsing 4efd132 · 7 months ago History

Files

39ee8a6

Go to file

.github
Documentation
Keys
Scripts
Src
.nuget
Configuration
DSInternals.Common.Test
DSInternals.Common
ADSI
AzureAD
Cryptography
Data
DPAPI
DPAPIBackupKey.cs
DPAPIBackupKeyType.cs
DPAPIObject.cs
KdsRootKey.cs
ProtectionKeyIdentifier.cs
RoamedCredential.cs
RoamedCredentialFlags.cs
RoamedCredentialType.cs
Hello
Principals
Schema

DSInternals / Src / DSInternals.Common / Data / DPAPI / DPAPIBackupKey.cs ↑ Top

Code Blame 300 lines (267 loc) · 12.3 KB Raw

```
1   namespace DSInternals.Common.Data
2   {
3       using DSInternals.Common.Cryptography;
4       using System;
5       using System.IO;
6       using System.Security.Cryptography;
7       using System.Security.Cryptography.X509Certificates;
8       using System.Text.RegularExpressions;
9
10      public class DPAPIBackupKey : DPAPIObject
15          private const int RSACertificateSizeOffset = RSAPrivateKeySizeOffset + sizeof(i
16          private const int RSAPrivateKeyOffset = RSACertificateSizeOffset + sizeof(int);
17          private const string BackupKeyNameFormat = "G$BCKUPKEY_{0}";
18          private const string BackupKeyDNFormat = "CN=BCKUPKEY_{0} Secret,CN=System,{1}"
19          // Examples:
20          // CN=BCKUPKEY_P Secret,CN=System,DC=contoso,DC=com
21          // CN=BCKUPKEY_PREFERRED Secret,CN=System,DC=contoso,DC=com
22          // CN=BCKUPKEY_PREFERRED Secret\0ACNF:26c8edbb-6b48-4f11-9e13-9ddbccedab5a,CN=S
23          // CN=BCKUPKEY_ac9e427c-fa85-4b78-8db1-771d94c03bad Secret,CN=System,DC=contoso
24          private const string BackupKeyDNRegex = "CN=BCKUPKEY_(.+) Secret(\\\\0ACNF:[0-9
25          private const string PreferredLegacyKeyPointerName = "P";
26          private const string PreferredRSAKeyPointerName = "PREFERRED";
27          private const string TemporaryKeyContainerName = "DSInternals";
28          private const string RSAKeyFileNameFormat = "ntds_capi_{0}.pvk";
29          private const string RSACertFileNameFormat = "ntds_capi_{0}.cer";
30          private const string RSAP12FileNameFormat = "ntds_capi_{0}.pfx";
31          private const string LegacyKeyFileNameFormat = "ntds_legacy_{0}.key";
32          private const string UnknownKeyFileNameFormat = "ntds_unknown_{0}_{1}.key";
33          private const string KiwiCommandFormat = "REM Add this parameter to at least th
34          private const int PVKHeaderSize = 6 * sizeof(int);
35          private const uint PVKHeaderMagic = 0xb0b5f11e;
36          private const uint PVKHeaderVersion = 0;
37          private const uint PVKHeaderKeySpec = 1; // = AT_KEYEXCHANGE
38
39          public DPAPIBackupKey(DirectoryObject dsObject, DirectorySecretDecryptor pek)
40          {
41              // Parameter validation
42              Validator.AssertNotNull(dsObject, "dsObject");
43              Validator.AssertNotNull(pek, "pek");
44              // TODO: Test Object type
45
46              // Decrypt the secret value
47              byte[] encryptedSecret;
48              dsObject.ReadAttribute(CommonDirectoryAttributes.CurrentValue, out encrypte
49              byte[] decryptedBlob = pek.DecryptSecret(encryptedSecret);
50
51              // Initialize properties
52              this.Initialize(dsObject.DistinguishedName, decryptedBlob);
53          }
54
55          public DPAPIBackupKey(string distinguishedName, byte[] blob)
56          {
57              // Validate the input
```

DNWithBinary.cs
DSDataRepresentation.cd
DirectoryObject.cs
DistinguishedName.cs
DistinguishedNameCompone...
InstanceType.cs
Exceptions
Extensions
Interop
Properties
DSInternals.Common.csproj
DSInternals.Common.nuspec
Validator.cs
packages.config
DSInternals.DataStore.Test
DSInternals.DataStore
DSInternals.PowerShell

```
            // Validate the input
            Validator.AssertNotNullOrWhiteSpace(distinguishedName, "distinguishedName");
            Validator.AssertNotNull(blob, "blob");

            this.Initialize(distinguishedName, blob);
        }

        public DPAPIBackupKey(Guid keyId, byte[] blob)
        {
            Validator.AssertNotNull(blob, "blob");
            this.KeyId = keyId;
            this.Type = GetKeyType(blob);
            this.Data = blob;
        }

        public override string FilePath
        {
            get
            {
                switch(this.Type)
                {
                    case DPAPIBackupKeyType.RSAKey:
                        // .pvk file
                        return String.Format(RSAKeyFileNameFormat, this.KeyId);
                    case DPAPIBackupKeyType.LegacyKey:
                        // .key file
                        return String.Format(LegacyKeyFileNameFormat, this.KeyId);
                    case DPAPIBackupKeyType.Unknown:
                        // Generate an additional random ID to prevent potential filena
                        int rnd = new Random().Next();
                        return String.Format(UnknownKeyFileNameFormat, this.KeyId, rnd)
                    default:
                        // Saving pointers or other domain key types to files is not su
                        return null;
                }
            }
        }

        public override string KiwiCommand
        {
            get
            {
                return this.Type == DPAPIBackupKeyType.RSAKey ? String.Format(KiwiComma
            }
        }

        public DPAPIBackupKeyType Type
        {
            get;
            private set;
        }
        public string DistinguishedName
        {
            get;
            private set;
        }

        public Guid KeyId
        {
            get;
            private set;
        }

        public override void Save(string directoryPath)
        {
            // The target directory must exist
            Validator.AssertDirectoryExists(directoryPath);

            string fullFilePath;

            switch (this.Type)
            {
                case DPAPIBackupKeyType.RSAKey:
                    // Parse the public and private keys
                    int privateKeySize = BitConverter.ToInt32(this.Data, RSAPrivateKeyS
```

```
132                 int certificateSize = BitConverter.ToInt32(this.Data, RSACertificat
133
134                 byte[] privateKey = this.Data.Cut(RSAPrivateKeyOffset, privateKeySi
135                 byte[] certificate = this.Data.Cut(RSAPrivateKeyOffset + privateKey
136
137                 // Create PVK file
138                 fullFilePath = Path.Combine(directoryPath, this.FilePath);
139                 byte[] pvk = EncapsulatePvk(privateKey);
140                 File.WriteAllBytes(fullFilePath, pvk);
141
142                 // Create PFX file
143                 byte[] pkcs12 = CreatePfx(certificate, privateKey);
144                 var pfxFile = String.Format(RSAP12FileNameFormat, this.KeyId);
145                 fullFilePath = Path.Combine(directoryPath, pfxFile);
```

```
227             public static string GetKeyName(Guid keyId)
228             {
229                 return String.Format(BackupKeyNameFormat, keyId);
230             }
231
232             public static string GetPreferredRSAKeyPointerDN(string domainDN)
233             {
234                 return String.Format(BackupKeyDNFormat, PreferredRSAKeyPointerName, domainD
235             }
236
237             public static string GetPreferredLegacyKeyPointerDN(string domainDN)
238             {
239                 return String.Format(BackupKeyDNFormat, PreferredLegacyKeyPointerName, doma
240             }
241
242             private static string GetSecretNameFromDN(string distinguishedName)
243             {
244                 var match = Regex.Match(distinguishedName, BackupKeyDNRegex);
245                 bool success = match.Success && (match.Groups.Count >= 2);
246                 return success ? match.Groups[1].Value : null;
247             }
248
249             private static byte[] CreatePfx(byte[] certificate, byte[] privateKey)
250             {
251                 // The PFX export only works if the key is stored in a named container
252                 var cspParameters = new CspParameters();
253                 cspParameters.KeyContainerName = TemporaryKeyContainerName;
254                 using (var keyContainer = new RSACryptoServiceProvider(cspParameters))
255                 {
256                     // Make the key temporary
257                     keyContainer.PersistKeyInCsp = false;
258                     keyContainer.ImportCspBlob(privateKey);
259                     // Combine the private and public keys
260                     var combinedCertificate = new X509Certificate2(certificate);
261                     combinedCertificate.PrivateKey = keyContainer;
262                     // Convert to binary PFX
263                     return combinedCertificate.Export(X509ContentType.Pfx);
264                 }
265             }
266
267             private static byte[] EncapsulatePvk(byte[] privateKey)
268             {
269                 // We do a quick and dirty encapsulation of the private key into the PVK fo
270                 // See: http://www.drh-consultancy.demon.co.uk/pvk.html
271                 // TODO: Extract PVK code to a distinct class.
272                 int pvkSize = PVKHeaderSize + privateKey.Length;
273                 byte[] pvk = new byte[pvkSize];
274
275                 using (var stream = new MemoryStream(pvk, true))
276                 {
277                     using (var writer = new BinaryWriter(stream))
278                     {
279                         // Write PVK header
280                         writer.Write(PVKHeaderMagic);
```

```
281                         writer.Write(PVKHeaderVersion);
282                         writer.Write(PVKHeaderKeySpec);
283                         writer.Write((int)PrivateKeyEncryptionType.None);
284                         writer.Write((int)0); // Size of salt
285                         writer.Write(privateKey.Length);
286
287                         // Write the actual data
288                         writer.Write(privateKey);
289                     }
290                 }
291
292                 return pvk;
293             }
294
295             private static DPAPIBackupKeyType GetKeyType(byte[] blob)
296             {
297                 return (DPAPIBackupKeyType)BitConverter.ToInt32(blob, KeyVersionOffset);
298             }
299         }
300     }
```