



Sign in

MichaelGrafnetter / DSInternals Public

Notifications

Fork 254

Star 1.6k

Code Issues 33 Pull requests 1 Actions Security Insights

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



300 lines (267 loc) · 12.3 KB

Code Blame

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
11    {
12        private const int KeyVersionOffset = 0;
13        private const int KeyVersionSize = sizeof(int);
14        private const int RSAPrivateKeySizeOffset = KeyVersionOffset + KeyVersionSize;
15        private const int RSACertificateSizeOffset = RSAPrivateKeySizeOffset + sizeof(int);
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-6b48-4f11-9e13-9ddbcedab5a,CN=System,DC=cor
23        // CN=BCKUPKEY_ac9e427c-fa85-4b78-8db1-771d94c03bad Secret,CN=System,DC=contoso,DC=com
24        private const string BackupKeyDNRegex = "CN=BCKUPKEY_(.) Secret(\\\\\\\\0ACNF:[0-9a-f]{8})-[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 the first dpapi";
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 encryptedSecret);
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
58         Validator.AssertNotNullOrWhiteSpace(distinguishedName, "distinguishedName");
59         Validator.AssertNotNull(blob, "blob");
60
61         this.Initialize(distinguishedName, blob);
62     }
63
64     public DPAPIBackupKey(Guid keyId, byte[] blob)
65     {
66         Validator.AssertNotNull(blob, "blob");
67         this.KeyId = keyId;
68         this.Type = GetKeyType(blob);
69         this.Data = blob;
70     }
71
72     public override string FilePath
```

```
73     {
74         get
75         {
76             switch(this.Type)
77             {
78                 case DPAPIBackupKeyType.RSAKey:
79                     // .pvk file
80                     return String.Format(RSAKeyFileNameFormat, this.KeyId);
81                 case DPAPIBackupKeyType.LegacyKey:
82                     // .key file
83                     return String.Format(LegacyKeyFileNameFormat, this.KeyId);
84                 case DPAPIBackupKeyType.Unknown:
85                     // Generate an additional random ID to prevent potential filename conflicts
86                     int rnd = new Random().Next();
87                     return String.Format(UnknownKeyFileNameFormat, this.KeyId, rnd);
88                 default:
89                     // Saving pointers or other domain key types to files is not supported.
90                     return null;
91             }
92         }
93     }
94
95     public override string KiwiCommand
96     {
97         get
98         {
99             return this.Type == DPAPIBackupKeyType.RSAKey ? String.Format(KiwiCommandFormat, th
100         }
101     }
102
103     public DPAPIBackupKeyType Type
104     {
105         get;
106         private set;
107     }
108
109     public string DistinguishedName
110     {
111         get;
112         private set;
113     }
114
115     public Guid KeyId
116     {
117         get;
118         private set;
119     }
```

110 J



```
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, domainDN);
235     }
236
237     public static string GetPreferredLegacyKeyPointerDN(string domainDN)
238     {
239         return String.Format(BackupKeyDNFormat, PreferredLegacyKeyPointerName, domainDN);
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 format.
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 }
```

