عنوان: **تهیه XML امضاء شده جهت تولید مجوز استفاده از برنامه**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۰/۲۵ ۸:۵
آدرس: www.dotnettips.info
برچسب‌ها: Security, RSA, Signed Xml

اگر به فایل مجوز استفاده از برنامه‌ای مانند EF Profiler دقت کنید، یک فایل XML به ظاهر ساده بیشتر نیست:

```
<?xml version="1.0" encoding="utf-8"?>
<license id="17d46246-a6cb-4196-98a0-ff6fc08cb67f" expiration="2012-06-12T00:00:00.0000000"
type="Trial" prof="EFProf">
  <name>MyName</name>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
        <DigestValue>b8N0bDE4gTakfdGKtzDflmmyyXI=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>IPELgc9BbkD8smXSe0sGqp5vS57CtZo9ME2ZfXSq/thVu...=</SignatureValue>
  </Signature>
</license>
```

در این فایل ساده متنی، نوع مجوز استفاده از برنامه، Trial ذکر شده است. شاید عنوان کنید که خوب ... نوع مجوز را به Standard تغییر می‌دهیم و فایل را ذخیره کرده و نهایتا استفاده می‌کنیم. اما ... نتیجه حاصل کار نخواهد کرد! حتی اگر یک نقطه به اطلاعات این فایل اضافه شود، دیگر قابل استفاده نخواهد بود. علت آن هم به قسمت Signature فایل XML فوق بر می‌گردد.
در ادامه به نحوه تولید و استفاده از یک چنین مجوزهای امضاء شده‌ای در برنامه‌های دات نتی خواهیم پرداخت.

تولید کلیدهای RSA

برای تهیه امضای دیجیتال یک فایل XML نیاز است از الگوریتم RSA استفاده شود.
برای تولید فایل XML امضاء شده، از کلید خصوصی استفاده خواهد شد. برای خواندن اطلاعات مجوز (فایل XML امضاء شده)، از کلیدهای عمومی که در برنامه قرار می‌گیرند کمک خواهیم گرفت (برای نمونه برنامه EF Prof این کلیدها را در قسمت Resourceهای خود قرار داده است).
استفاده کننده تنها زمانی می‌تواند مجوز معتبری را تولید کند که دسترسی به کلیدهای خصوصی تولید شده را داشته باشد.

```
public static string CreateRSAKeyPair(int dwKeySize = 1024)
        {
            using (var provider = new RSACryptoServiceProvider(dwKeySize))
            {
                return provider.ToXmlString(includePrivateParameters: true);
            }
        }
```

امکان تولید کلیدهای اتفاقی مورد استفاده در الگوریتم RSA، در دات نت پیش بینی شده است. خروجی متد فوق یک فایل XML است که به همین نحو در صورت نیاز توسط متد provider.FromXmlString مورد استفاده قرار خواهد گرفت.

تهیه ساختار مجوز

در ادامه یک enum که بیانگر انواع مجوزهای برنامه ما است را مشاهده می‌کنید:

```
namespace SignedXmlSample
{
    public enum LicenseType
    {
```

```
        None,
        Trial,
        Standard,
        Personal
    }
}
```

به همراه کلاسی که اطلاعات مجوز تولیدی را دربر خواهد گرفت:

```
using System;
using System.Xml.Serialization;

namespace SignedXmlSample
{
    public class License
    {
        [XmlAttribute]
        public Guid Id { set; get; }

        public string Domain { set; get; }

        [XmlAttribute]
        public string IssuedTo { set; get; }

        [XmlAttribute]
        public DateTime Expiration { set; get; }

        [XmlAttribute]
        public LicenseType Type { set; get; }
    }
}
```

خواص این کلاس یا عناصر enum یاد شده کاملا دلخواه هستند و نقشی را در ادامه بحث نخواهند داشت؛ از این جهت که از مباحث XmlSerializer برای تبدیل وهله‌ای از شیء مجوز به معادل XML آن استفاده می‌شود. بنابراین المان‌های آن‌را مطابق نیاز خود می‌توانید تغییر دهید. همچنین ذکر ویژگی XmlAttribute نیز اختیاری است. در اینجا صرفا جهت شبیه سازی معادل مثالی که در ابتدای بحث مطرح شد، از آن استفاده شده است. این ویژگی راهنمایی است برای کلاس XmlSerializer تا خواص مزین شده با آن‌را به شکل یک Attribute در فایل نهایی ثبت کند.

تولید و خواندن مجوز دارای امضای دیجیتال

کدهای کامل کلاس تولید و خواندن یک مجوز دارای امضای دیجیتال را در اینجا مشاهده می‌کنید:

```
using System;
using System.IO;
using System.Security.Cryptography; // needs a ref. to `System.Security.dll` asm.
using System.Security.Cryptography.Xml;
using System.Text;
using System.Xml;
using System.Xml.Serialization;

namespace SignedXmlSample
{
    public static class LicenseGenerator
    {
        public static string CreateLicense(string licensePrivateKey, License licenseData)
        {
            using (var provider = new RSACryptoServiceProvider())
            {
                provider.FromXmlString(licensePrivateKey);
                var xmlDocument = createXmlDocument(licenseData);
                var xmlDigitalSignature = getXmlDigitalSignature(xmlDocument, provider);
                appendDigitalSignature(xmlDocument, xmlDigitalSignature);
                return xmlDocumentToString(xmlDocument);
            }
        }

        public static string CreateRSAKeyPair(int dwKeySize = 1024)
        {
            using (var provider = new RSACryptoServiceProvider(dwKeySize))
```

```
            {
                return provider.ToXmlString(includePrivateParameters: true);
            }
        }

        public static License ReadLicense(string licensePublicKey, string xmlFileContent)
        {
            var doc = new XmlDocument();
            doc.LoadXml(xmlFileContent);

            using (var provider = new RSACryptoServiceProvider())
            {
                provider.FromXmlString(licensePublicKey);

                var nsmgr = new XmlNamespaceManager(doc.NameTable);
                nsmgr.AddNamespace("sig", "http://www.w3.org/2000/09/xmldsig#");

                var xml = new SignedXml(doc);
                var signatureNode = (XmlElement)doc.SelectSingleNode("//sig:Signature", nsmgr);
                if (signatureNode == null)
                    throw new InvalidOperationException("This license file is not signed.");

                xml.LoadXml(signatureNode);
                if (!xml.CheckSignature(provider))
                    throw new InvalidOperationException("This license file is not valid.");

                var ourXml = xml.GetXml();
                if (ourXml.OwnerDocument == null || ourXml.OwnerDocument.DocumentElement == null)
                    throw new InvalidOperationException("This license file is coruppted.");

                using (var reader = new XmlNodeReader(ourXml.OwnerDocument.DocumentElement))
                {
                    var xmlSerializer = new XmlSerializer(typeof(License));
                    return (License)xmlSerializer.Deserialize(reader);
                }
            }
        }

        private static void appendDigitalSignature(XmlDocument xmlDocument, XmlNode
xmlDigitalSignature)
        {
            xmlDocument.DocumentElement.AppendChild(xmlDocument.ImportNode(xmlDigitalSignature, true));
        }

        private static XmlDocument createXmlDocument(License licenseData)
        {
            var serializer = new XmlSerializer(licenseData.GetType());
            var sb = new StringBuilder();
            using (var writer = new StringWriter(sb))
            {
                var ns = new XmlSerializerNamespaces(); ns.Add("", "");
                serializer.Serialize(writer, licenseData, ns);
                var doc = new XmlDocument();
                doc.LoadXml(sb.ToString());
                return doc;
            }
        }

        private static XmlElement getXmlDigitalSignature(XmlDocument xmlDocument, AsymmetricAlgorithm
key)
        {
            var xml = new SignedXml(xmlDocument) { SigningKey = key };
            var reference = new Reference { Uri = "" };
            reference.AddTransform(new XmlDsigEnvelopedSignatureTransform());
            xml.AddReference(reference);
            xml.ComputeSignature();
            return xml.GetXml();
        }

        private static string xmlDocumentToString(XmlDocument xmlDocument)
        {
            using (var ms = new MemoryStream())
            {
                var settings = new XmlWriterSettings { Indent = true, Encoding = Encoding.UTF8 };
                var xmlWriter = XmlWriter.Create(ms, settings);
                xmlDocument.Save(xmlWriter);
                ms.Position = 0;
                return new StreamReader(ms).ReadToEnd();
            }
        }
    }
```

```
}
```

توضیحات:
در حین کار با متد CreateLicense، پارامتر licensePrivateKey همان اطلاعاتی است که به کمک متد CreateRSAKeyPair قابل تولید است. توسط پارامتر licenseData، اطلاعات مجوز در حال تولید اخذ می‌شود. در این متد به کمک provider.FromXmlString، اطلاعات کلیدهای RSA دریافت خواهند شد. سپس توسط متد createXmlDocument، محتوای licenseData دریافتی به یک فایل XML نگاشت می‌گردد (بنابراین اهمیتی ندارد که حتما از ساختار کلاس مجوز یاد شده استفاده کنید). در ادامه متد getXmlDigitalSignature با در اختیار داشتن معادل XML شیء مجوز و کلیدهای لازم، امضای دیجیتال متناظری را تولید می‌کند. با استفاده از متد appendDigitalSignature، این امضاء را به فایل XML اولیه اضافه می‌کنیم. از این امضاء جهت بررسی اعتبار مجوز برنامه در متد ReadLicense استفاده خواهد شد.
برای خواندن یک فایل مجوز امضاء شده در برنامه خود می‌توان از متد ReadLicense استفاده کرد. توسط آرگومان licensePublicKey، اطلاعات کلید عمومی دریافت می‌شود. این کلید دربرنامه، ذخیره و توزیع می‌گردد. پارامتر xmlFileContent معادل محتوای فایل XML مجوزی است که قرار است مورد ارزیابی قرار گیرد.

مثالی در مورد نحوه استفاده از کلاس تولید مجوز

در ادامه نحوه استفاده از متدهای CreateLicense و ReadLicense را ملاحظه می‌کنید؛ به همراه آشنایی با نمونه کلیدهایی که باید به همراه برنامه منتشر شوند:

```
using System;
using System.IO;

namespace SignedXmlSample
{
    class Program
    {
        static void Main(string[] args)
        {
            //Console.WriteLine(LicenseGenerator.CreateRSAKeyPair());
            writeLicense();
            readLicense();

            Console.WriteLine("Press a key...");
            Console.ReadKey();
        }

        private static void readLicense()
        {
            var xml = File.ReadAllText("License.xml");
            const string publicKey = @"<RSAKeyValue>
                                    <Modulus>

mBNKFIc/LkMfaXvLlB/+6EujPkx3vBOvLu8jdESDSQLisT8K96RaDMD1ORmdw2XNdMw/6ZBuJjLhoY13qCU9t7biuL3SIxr858oJ1RL
M4PKhA/wVDcYnJXmAUuOyxP/vfvb798o6zAC1R2QWuzG+yJQR7bFmbKH0tXF/NOcSgbc=
                                    </Modulus>
                                    <Exponent>
                                        AQAB
                                    </Exponent>
                                </RSAKeyValue>";

            var result = LicenseGenerator.ReadLicense(publicKey, xml);

            Console.WriteLine(result.Domain);
            Console.WriteLine(result.IssuedTo);
        }

        private static void writeLicense()
        {
            const string rsaData = @"<RSAKeyValue>
                    <Modulus>

mBNKFIc/LkMfaXvLlB/+6EujPkx3vBOvLu8jdESDSQLisT8K96RaDMD1ORmdw2XNdMw/6ZBuJjLhoY13qCU9t7biuL3SIxr858oJ1RL
M4PKhA/wVDcYnJXmAUuOyxP/vfvb798o6zAC1R2QWuzG+yJQR7bFmbKH0tXF/NOcSgbc=
                    </Modulus>
                    <Exponent>
                        AQAB
                    </Exponent>
```

```
                    <P>

xwPKN77EcolMTD2O2Csv6k9Y4aen8UBVYjeQ4PtrNGz0Zx6I1MxLEFzRpiKC/Ney3xKg0Icwj0ebAQ04d5+HAQ==
                    </P>
                    <Q>

w568t0Xe6OBUfCyAuo7tTv4eLgczHntVLpjjcxdUksdVw7NJtlnOLApJVJ+U6/85Z7Ji+eVhuN91yn04pQkAtw==
                    </Q>
                    <DP>

svkEjRdA4WP5uoKNiHdmMshCvUQh8wKRBq/D2aAgq9fj/yxlj0FdrAxc+ZQFyk5MbPH6ry00jVWu3sY95s4PAQ==
                    </DP>
                    <DQ>

WcRsIUYk5oSbAGiDohiYeZlPTBvtr101V669IUFhhAGJL8cEWnOXksodoIGimzGBrD5GARrr3yRcL1GLPuCEvQ==
                    </DQ>
                    <InverseQ>

wIbuKBZSCioG6MHdT1jxlv6U1+Y3TX9sHED9PqGzWWpVGA+xFJmQUxoFf/SvHzwbBlXnG0DLqUvxEv+BkEid2w==
                    </InverseQ>
                    <D>
Yk21yWdT1BfXqlw30NyN7qNWNuM/Uvh2eaRkCrhvFTckSucxs7st6qig2/RPIwwfr6yIc/bE/TRO3huQicTpC2W3aXsBI9822OOX4Bd
WCec2txXpSkbZW24moXu+OSHfAdYoOEN6ocR7tAGykIqENshRO7HvONJsOE5+1kF2GAE=
                    </D>
                </RSAKeyValue>";

            string data = LicenseGenerator.CreateLicense(
                                                rsaData,
                                                new License
                                                {
                                                     Id = Guid.NewGuid(),
                                                     Domain = "dotnettips.info",
                                                     Expiration = DateTime.Now.AddYears(2),
                                                     IssuedTo = "VahidN",
                                                     Type = LicenseType.Standard
                                                });
            File.WriteAllText("License.xml", data);
        }
    }
}
```

ابتدا توسط متد CreateRSAKeyPair کلیدهای لازم را تهیه و ذخیره کنید. این کار یکبار باید صورت گیرد.
همانطور که مشاهده می‌کنید، اطلاعات کامل یک نمونه از آن، در متد writeLicense مورد نیاز است. اما در متد readLicense تنها به قسمت عمومی آن یعنی Modulus و Exponent نیاز خواهد بود (موارد قابل انتشار به همراه برنامه).

سؤال: امنیت این روش تا چه اندازه است؟
پاسخ: تا زمانیکه کاربر نهایی به کلیدهای خصوصی شما دسترسی پیدا نکند، امکان تولید معادل آن‌ها تقریبا در حد صفر است و به طول عمر او قد نخواهد داد!
اما ... مهاجم می‌تواند کلیدهای عمومی و خصوصی خودش را تولید کند. مجوز دلخواهی را بر این اساس تهیه کرده و سپس کلید عمومی جدید را در برنامه، بجای کلیدهای عمومی شما درج (patch) کند! بنابراین روش بررسی اینکه آیا برنامه جاری patch شده است یا خیر را فراموش نکنید. یا عموما مطابق معمول قسمتی از برنامه که در آن مقایسه‌ای بین اطلاعات دریافتی و اطلاعات مورد انتظار وجود دارد، وصله می‌شوند؛ این مورد عمومی است و منحصر به روش جاری نمی‌باشد.

دریافت نسخه جنریک این مثال:
SignedXmlSample.zip

نظرات خوانندگان

نویسنده: آراز پاشازاده
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۹:۴۳

مقاله کاملا مفید و جامع بود من نمونه مثال را دانلود و اجرا کردم ولی متاسفانه چیزی در مورد طریقه تست برنامه نگفتین؟

چطوری تست کنم تا متوجه بشم مجوز کار میکنه یا نه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۱۲/۲۳ ۱۹:۵۳

نحوه بررسی فایل تولیدی، در متد readLicense انتهای مقاله ذکر شده است. writeLicense یک فایل مجوز تولید می‌کند؛ سپس readLicense محتوای آن‌را خوانده و نمایش می‌دهد (چیزی که نهایتا در برنامه شما استفاده خواهد شد).