A6 Sensitive Data Exposure Lab

The door to your modestly-furnished office bursts open. "Boss, we've got a serious problem!" your Chief Security Officer says. I found a place in our system where our customers' credit cards can be compromised.
"What!?!" you exclaim. "I thought we took cards out of the database for exactly this reason."
"We did," he laments, "but our customers complained that we needed to store their cards to make it easier for them to order. Our VP Sales demanded that we store them. So our VP of Development put them back in – but not in the database. He's storing the cards in a flat file in an area of the website we thought was protected. The problem was that they weren't encrypted when they are stored."
"Unencrypted?!" you say with an eye-roll. "Look, this is so important I'm going to take care of it myself and right away."

Mounting the attack

1. First, open a Windows Explorer window and find our site. Look in the Checkout folder for a file called StoredCreditCards.xml. Open that file. Scared yet? It gets worse.
2. Next, open a browser and navigate to http://localhost:7777/Checkout/StoredCreditCards.xml Can you see the file? What this means is that any user could have seen the contents of that file simply by pointing his browser to that address. Good thing your CSO found it before a hacker did!

Preparing to encrypt the file

Now, let's look at how this file is read from and written to.

3. Find and open Checkout.aspx.cs. Notice that we are reading credit credit cards when the page loads and writing them in the button's click event.
4. Also notice that those methods are part of the CreditCard class. So go to the credit card class. You'll find it in the "Core" project.
5. Find where the XML file is being read from. What is the name of the method? _____
6. And what is the method where it is being written to? _____

These are the methods where we'll add code to encrypt and decrypt the file.

Encrypting the file

To encrypt the file, we simply need to insert a CryptoStream.

7. First, at the top of the file, add this namespace:
   ```
   using System.Security.Cryptography
   ```
8. Next, create an encryptor/decryptor. Immediately inside the class and outside of any methods, add a private variable:
   ```
   private Rijndael _rijndael = Rijndael.Create();
   ```
9. Then, for this to work you'll need to set a key and an initialization vector. Go to the constructor of the CreditCard class and add a key and IV of your own choice. Here are some example lines with sample a sample key, salt and IV:
   ```
   _rijndael.Key = new Rfc2898DeriveBytes("NetWebGoat",
     new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 })
     .GetBytes(_rijndael.KeySize / 8);
   _rijndael.IV = new byte[16]
     { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6 };
   ```
10. Finally, insert the encryption stream into the existing write stream. Go to the method where we are writing to the XML document and change this:
    ```
    using (var writeStream = File.Open(Filename, FileMode.Create))
    {
      var writer = new StreamWriter(writeStream);
      XmlDocument.Save(writer);
      writer.Close();
    ```

```
}
```
To this:
```
using (var writeStream = File.Open(Filename, FileMode.Create))
{
    var cryptoStream = new CryptoStream(writeStream,
        _rijndael.CreateEncryptor(), CryptoStreamMode.Write);
    var writer = new StreamWriter(cryptoStream);
    XmlDocument.Save(writer);
    writer.Close();
}
```

Decrypting the file

Now that we're writing in an encrypted way, we need to read in the same way.

11. Go to the method where we're reading from the credit card file.
12. Insert the decryptor into the stream. Change this:
```
using (FileStream readStream = File.Open(Filename, FileMode.Open))
{
    var xr = new XmlTextReader(readStream);
    document = XDocument.Load(xr);
    xr.Close();
}
```
To this:
```
using (FileStream readStream = File.Open(Filename, FileMode.Open))
{
    var cryptoStream = new CryptoStream(readStream,
        _rijndael.CreateDecryptor(), CryptoStreamMode.Read);
    var xr = new XmlTextReader(cryptoStream);
    document = XDocument.Load(xr);
    xr.Close();
}
```

Replaying the attack

13. Now we can't read an encrypted file like it were a clear text file. And we can't read a clear text file like it were encrypted. So go ahead and delete the XML file so we can start from scratch.
14. Log in as several users, placing orders and saving credit card info along the way.
15. After you've saved some cards, pull up the XML file and see if the contents are encrypted.
16. If they're not encrypted or if you're having any troubles reading or writing, please fix the problems and test again.

But don't relax just yet! There are also several places where sensitive data in motion is being sent insecurely in our site. Let's identify and fix those problems.

Securing cookies

1. Search through the site to see if you can find where we're writing a cookie. (Hint: search for Response.Cookies.Add)
2. Verify that the cookie being written is regular – not secure.
3. Run the site and navigate to the page where that cookie is being written. Examine the cookie using FireCookie or Fiddler or another tool that can show cookies.
4. Now stop the site running and add a secure property to the cookie. Something sort of like this will do:
```
sessionCookie.Secure = true;
```
5. Re-run the site and examine the cookie. Is it secure this time? _____.