Protect Your Flash Files From Decompilers by Using Encryption

By **Nikita Leshenko**, 11 Feb 2011

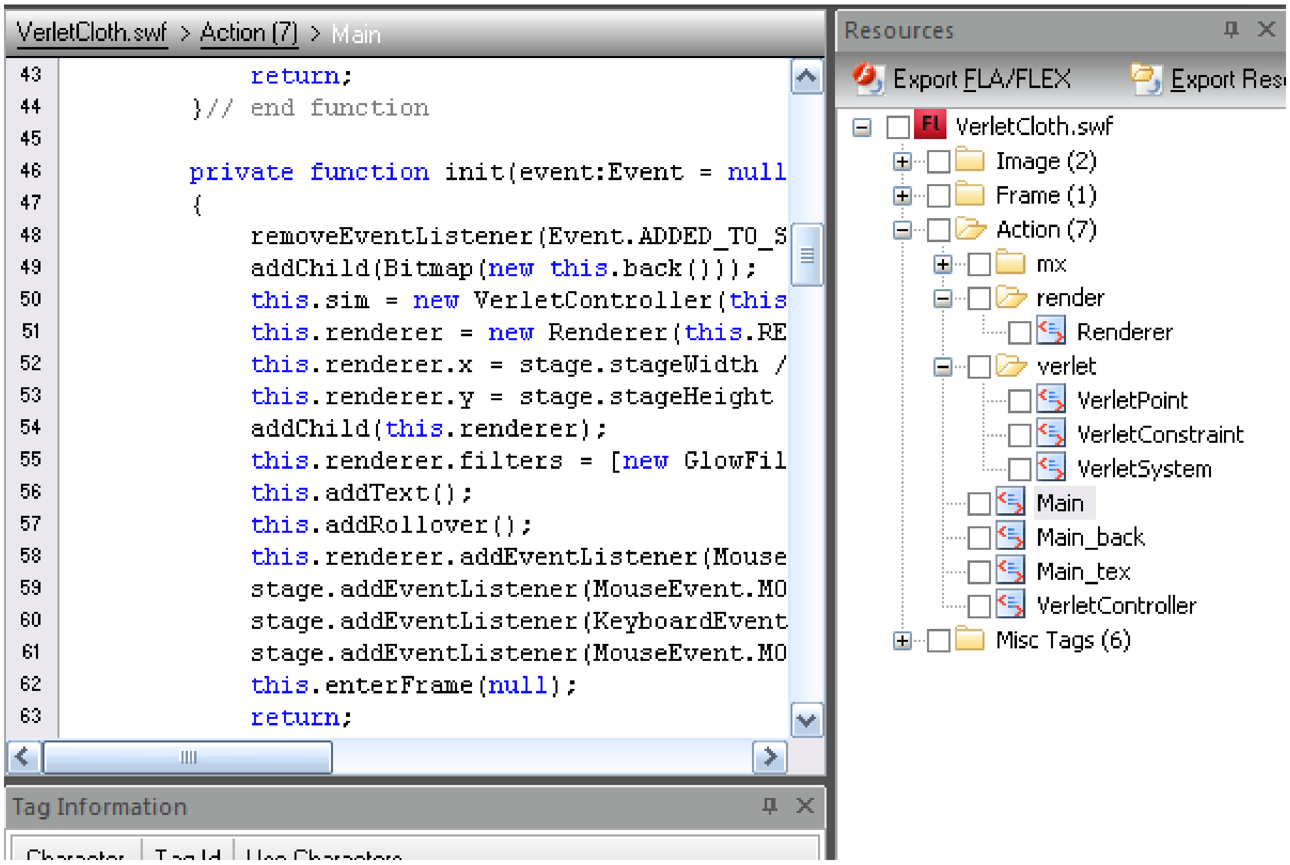
Twice a month, we revisit some of our readers' favorite posts from throughout the history of Activetuts+. This tutorial was first published in February, 2010.

In this tutorial I will demonstrate a technique I use to protect code and assets from theft.

Decompilers are a real worry for people who create Flash content. You can put a lot of effort into creating the best game out there, then someone can steal it, replace the logo and put it on their site without asking you. How? Using a Flash Decompiler. Unless you put some protection over your SWF it can be decompiled with a push of a button and the decompiler will output readable source code.

Before We Begin

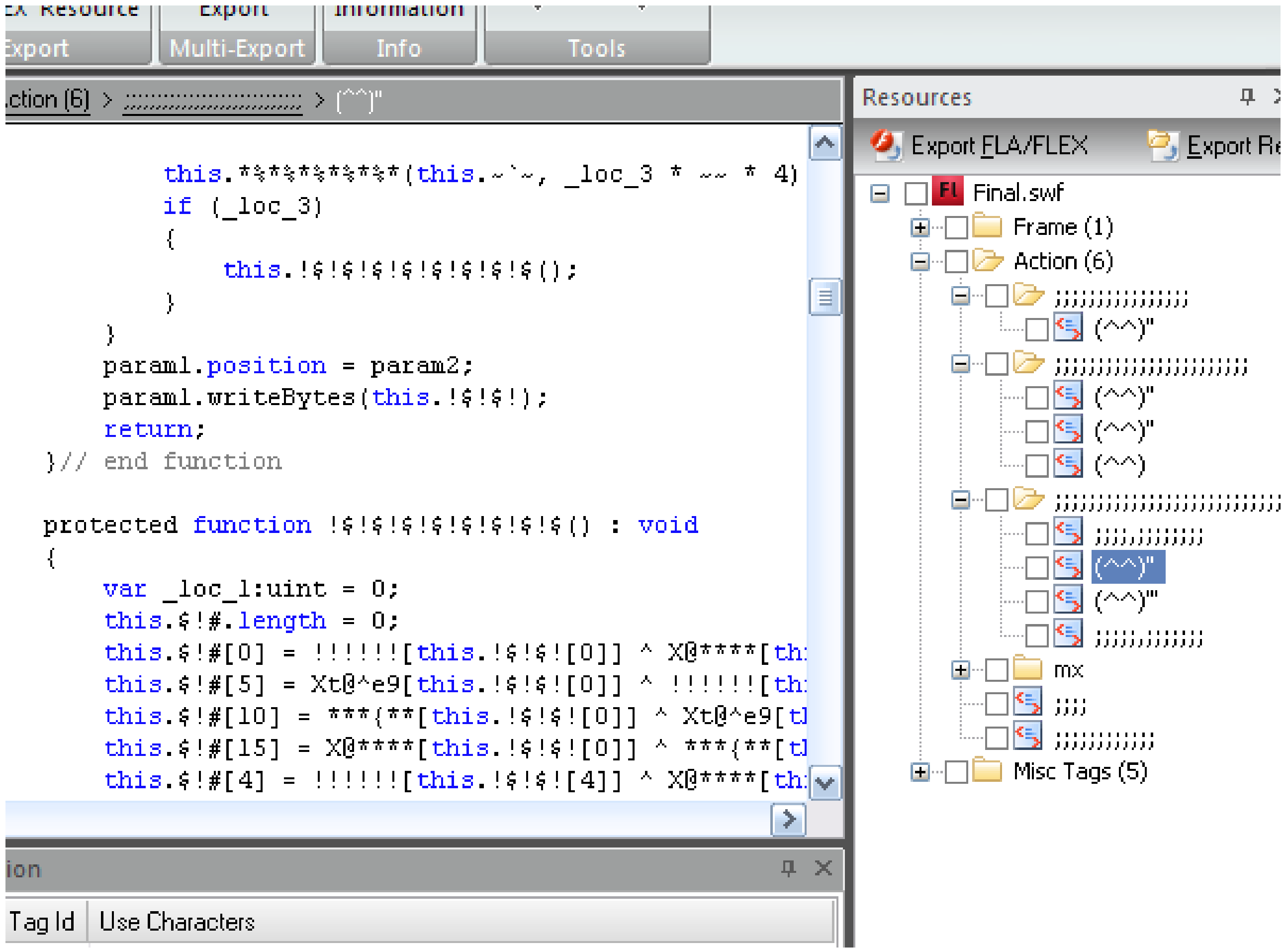
I used a small project of mine to demonstrate how vulnerable SWFs are to decompilation. You can download it and test yourself via the source link above. I used Sothink SWF Decompiler 5 to decompile the SWF and look under its hood. The code is quite readable and you can understand and reuse it fairly easily.



```
            return;
        }// end function

        private function init(event:Event = null
        {
            removeEventListener(Event.ADDED_TO_S
            addChild(Bitmap(new this.back()));
            this.sim = new VerletController(this
            this.renderer = new Renderer(this.RE
            this.renderer.x = stage.stageWidth /
            this.renderer.y = stage.stageHeight
            addChild(this.renderer);
            this.renderer.filters = [new GlowFil
            this.addText();
            this.addRollover();
            this.renderer.addEventListener(Mouse
            stage.addEventListener(MouseEvent.MO
            stage.addEventListener(KeyboardEvent
            stage.addEventListener(MouseEvent.MO
            this.enterFrame(null);
            return;
```

What Can We do About it?

I came up with a technique for protecting SWFs from decompilers and I'm going to demonstrate it in this tutorial. We should be able to produce this:



```
        this.*§*§*§*§*§*(this.~`~, _loc_3 * ~~ * 4)
        if (_loc_3)
        {
            this.!$!$!$!$!$!$!$!$();
        }
    }
    param1.position = param2;
    param1.writeBytes(this.!$!$!);
    return;
}// end function

protected function !$!$!$!$!$!$!$!$() : void
{
    var _loc_1:uint = 0;
    this.$!#.length = 0;
    this.$!#[0] = !!!!!![this.!$!$![0]] ^ X@****[th
    this.$!#[5] = Xt@^e9[this.!$!$![0]] ^ !!!!!![th
    this.$!#[10] = ***{**[this.!$!$![0]] ^ Xt@^e9[t
    this.$!#[15] = X@****[this.!$!$![0]] ^ ***{**[t
    this.$!#[4] = !!!!!![this.!$!$![4]] ^ X@****[th
```

The code that is decompiled is actually the code for decrypting the content and has nothing to do with your main code. Additionally, the names are illegal so it won't compile back. Try to decompile it yourself.

Before we get going, I want to point out that this tutorial is not suitable for beginners and you should have solid knowledge of AS3 if you want to follow along. This tutorial is also about low level programming that involves bytes, ByteArrays and manipulating SWF files with a hex editor.

Here's what we need:

- **A SWF to protect.** Feel free to download the SWF I'll be working on.
- **Flex SDK.** We will be using it to embed content using the Embed tag. You can download it from opensource.adobe.com.
- **A hex editor.** I'll be using a free editor called Hex-Ed. You can download it from nielshorn.net or you can use an editor of your choice.
- **A decompiler.** Whilst not necessary, it would be nice to check if our protection actually works. You can grab a trial of Sothink SWF Decompiler from sothink.com

Step 1: Load SWF at Runtime

Open a new ActionScript 3.0 project, and set it to compile with Flex SDK (I use FlashDevelop to write code). Choose a SWF you want to protect and embed it as binary data using the Embed tag:

```
[Embed (source = "VerletCloth.swf", mimeType = "application/octet-stream")]
// source = path to the swf you want to protect
private var content:Class;
```

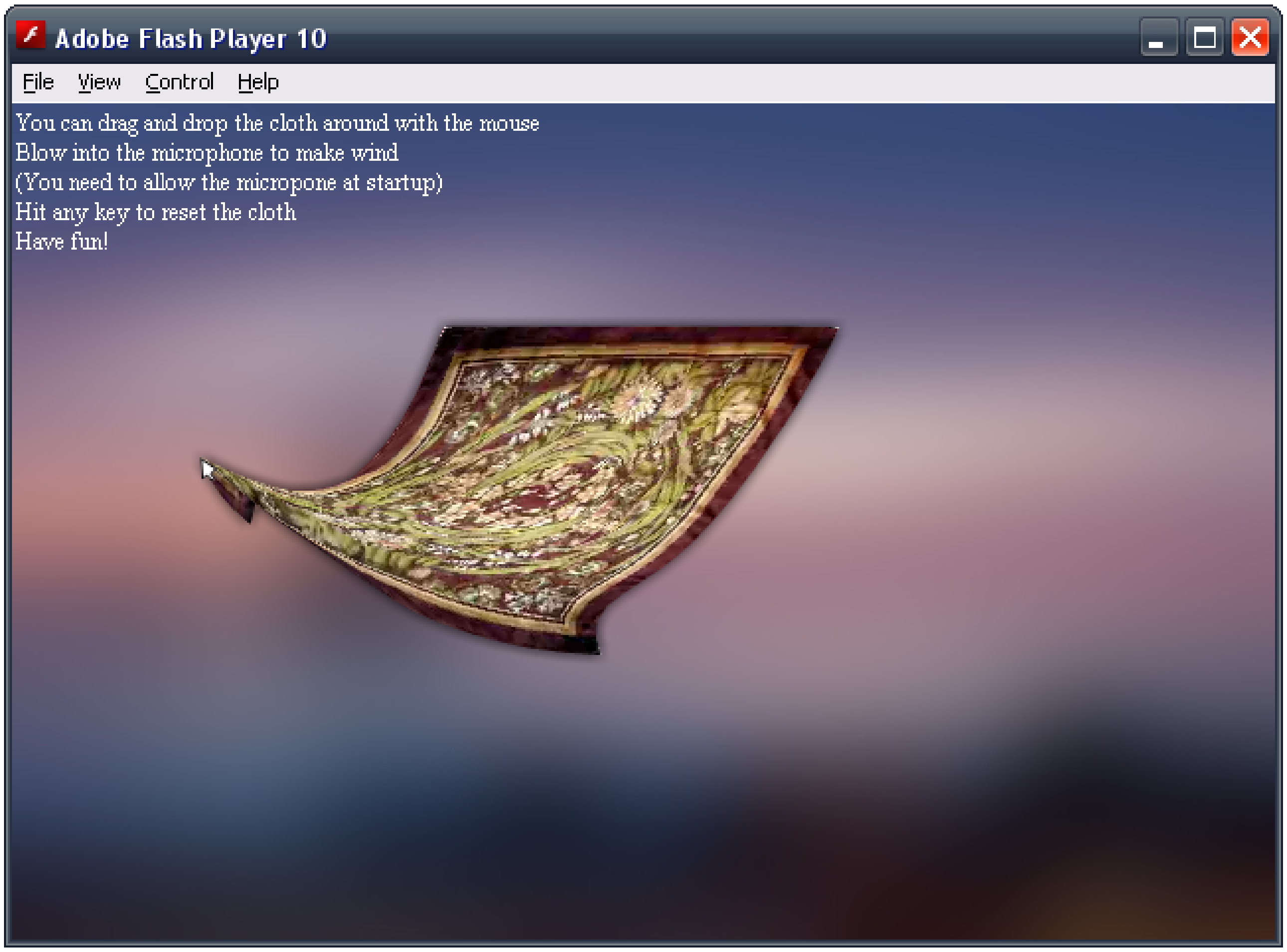
Now the SWF is embedded as a *ByteArray* into the loader SWF and it can be loaded through *Loader.loadBytes()*.

```
var loader:Loader = new Loader();
addChild(loader);
loader.loadBytes(new content(), new LoaderContext(false, new ApplicationDomain()));
```

In the end we should have this code:

```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    [SWF (width = 640, height = 423)] //the dimensions should be same as the l
    public class Main extends Sprite
    {
        [Embed (source = "VerletCloth.swf", mimeType = "application/octet-stre
        // source = path to the swf you want to protect
        private var content:Class;

        public function Main():void
        {
            var loader:Loader = new Loader();
            addChild(loader);
            loader.loadBytes(new content(), new LoaderContext(false, new Appli
        }
    }

}
```

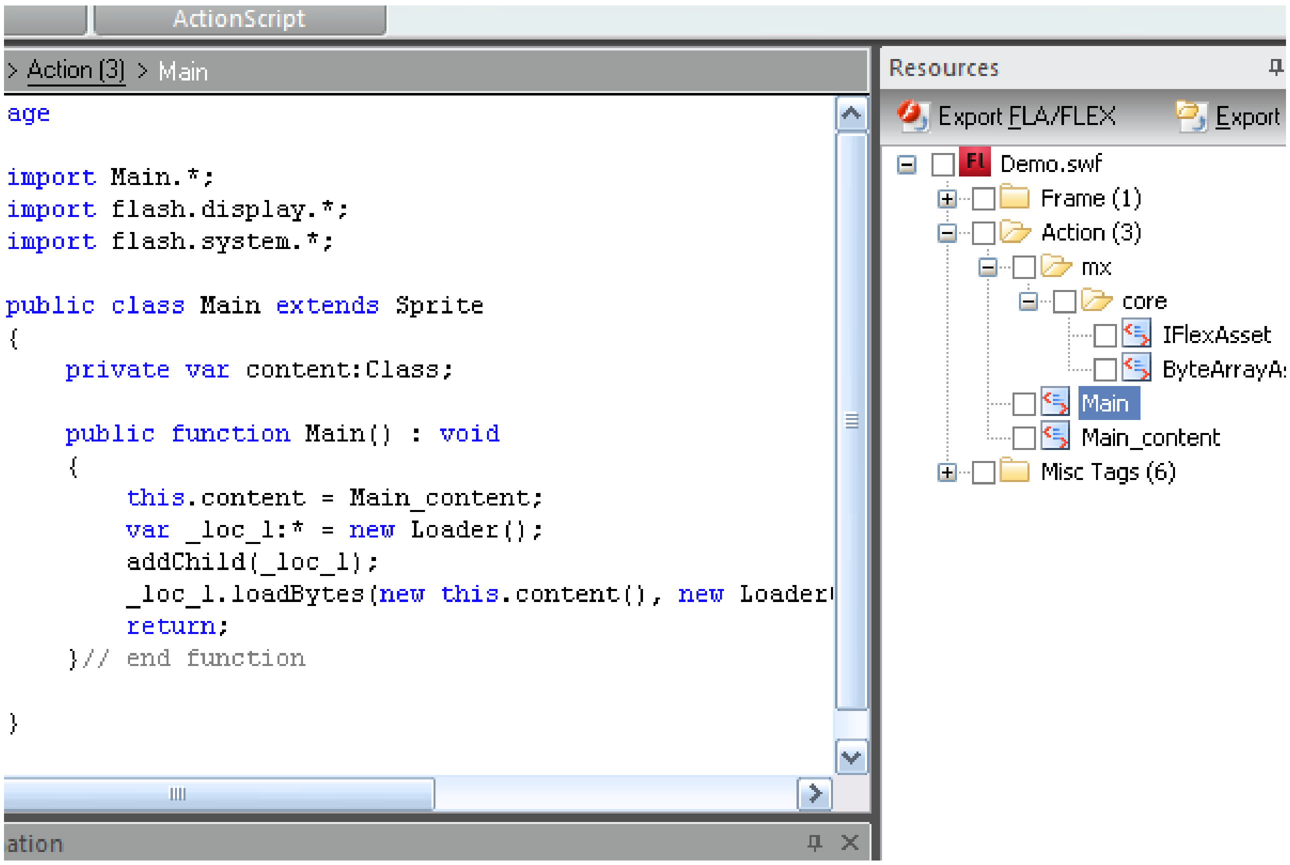
Compile and see if it works (it should). From now on I will call the embedded SWF the "protected SWF", and the SWF we just compiled the "loading SWF".



You can drag and drop the cloth around with the mouse
Blow into the microphone to make wind
(You need to allow the micropone at startup)
Hit any key to reset the cloth
Have fun!

Step 2: Analyze the Result

Let's try to decompile and see if it works.

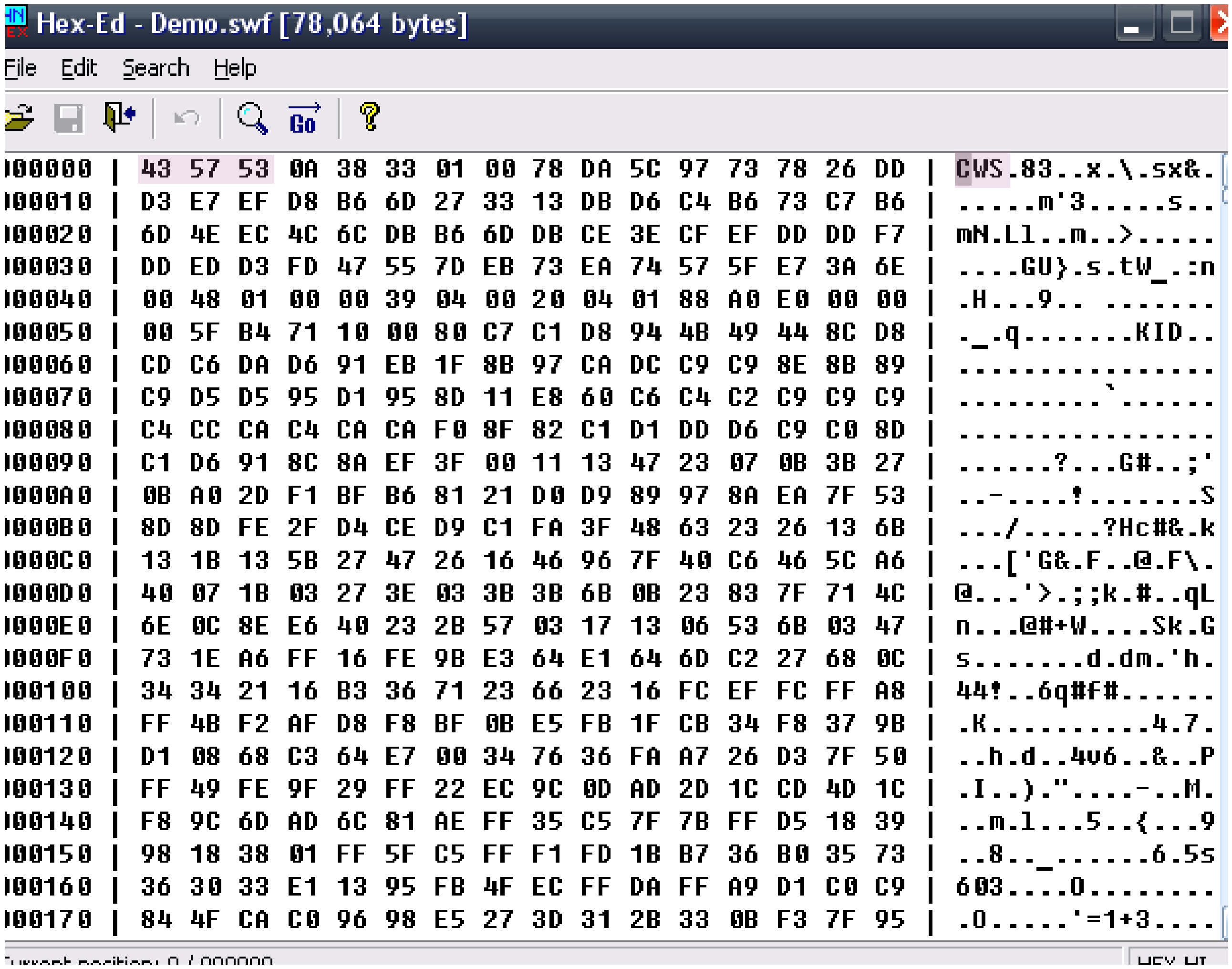


```
age

import Main.*;
import flash.display.*;
import flash.system.*;

public class Main extends Sprite
{
    private var content:Class;

    public function Main() : void
    {
        this.content = Main_content;
        var _loc_1:* = new Loader();
        addChild(_loc_1);
        _loc_1.loadBytes(new this.content(), new Loader
        return;
    }// end function

}
```

Yey! The assets and the original code are gone! What's shown now is the code that loads the protected SWF and not its content. This would probably stop most of the first-time attackers who are not too familiar with Flash but it's still not good enough to protect your work from skilled attackers because the protected SWF is waiting for them untouched inside the loading SWF.

Step 3: Decompressing the SWF

Let's open the loading SWF with a hex editor:



```
000000 | 43 57 53 0A 38 33 01 00 78 DA 5C 97 73 78 26 DD | CWS.83..x.\.sx&.
000010 | D3 E7 EF D8 B6 6D 27 33 13 DB D6 C4 B6 73 C7 B6 | .....m'3.....s..
000020 | 6D 4E EC 4C 6C DB B6 6D DB CE 3E CF EF DD DD F7 | mN.Ll..m..>.....
000030 | DD ED D3 FD 47 55 7D EB 73 EA 74 57 5F E7 3A 6E | ....GU}.s.tW_.:n
000040 | 00 48 01 00 00 39 04 00 20 04 01 88 A0 E0 00 00 | .H...9.. .......
000050 | 00 5F B4 71 10 00 80 C7 C1 D8 94 4B 49 44 8C D8 | ._.q.......KID..
000060 | CD C6 DA D6 91 EB 1F 8B 97 CA DC C9 C9 8E 8B 89 | ................
000070 | C9 D5 D5 95 D1 95 8D 11 E8 60 C6 C4 C2 C9 C9 C9 | .........`......
000080 | C4 CC CA C4 CA CA F0 8F 82 C1 D1 DD D6 C9 C0 8D | ................
000090 | C1 D6 91 8C 8A EF 3F 00 11 13 47 23 07 0B 3B 27 | ......?...G#..;'
0000A0 | 0B A0 2D F1 BF B6 81 21 D0 D9 89 97 8A EA 7F 53 | ..-....!.......S
0000B0 | 8D 8D FE 2F D4 CE D9 C1 FA 3F 48 63 23 26 13 6B | .../.....?Hc#&.k
0000C0 | 13 1B 13 5B 27 47 26 16 46 96 7F 40 C6 46 5C A6 | ...['G&.F..@.F\.
0000D0 | 40 07 1B 03 27 3E 03 3B 3B 6B 0B 23 83 7F 71 4C | @...'>.;;k.#..qL
0000E0 | 6E 0C 8E E6 40 23 2B 57 03 17 13 06 53 6B 03 47 | n...@#+W....Sk.G
0000F0 | 73 1E A6 FF 16 FE 9B E3 64 E1 64 6D C2 27 68 0C | s.......d.dm.'h.
000100 | 34 34 21 16 B3 36 71 23 66 23 16 FC EF FC FF A8 | 44!..6q#f#......
000110 | FF 4B F2 AF D8 F8 BF 0B E5 FB 1F CB 34 F8 37 9B | .K..........4.7.
000120 | D1 08 68 C3 64 E7 00 34 76 36 FA A7 26 D3 7F 50 | ..h.d..4v6..&..P
000130 | FF 49 FE 9F 29 FF 22 EC 9C 0D AD 2D 1C CD 4D 1C | .I..).".....-..M.
000140 | F8 9C 6D AD 6C 81 AE FF 35 C5 7F 7B FF D5 18 39 | ..m.l...5..{...9
000150 | 98 18 38 01 FF 5F C5 FF F1 FD 1B B7 36 B0 35 73 | ..8.._......6.5s
000160 | 36 30 33 E1 13 95 FB 4F EC FF DA FF A9 D1 C0 C9 | 603....O........
000170 | 84 4F CA C0 96 98 E5 27 3D 31 2B 33 0B F3 7F 95 | .O.....'=1+3....
```

It should look like random binary data because it's compressed and it should begin with ASCII "CWS". We need to decompress it! (If your SWF begins with "FWS" and you see meaningful strings in the SWF it's likely that it didn't get compressed. You have to enable compression to follow along).

At first it might sound difficult but it's not. The SWF format is an open format and there is a document that describes it. Download it from adobe.com and scroll down to page 25 in the document. There is a description of the header and how the SWF is compressed, so we can uncompress it easily.

What is written there is that the first 3 bytes are a signature (CWS or FWS), the next byte is the Flash version, the next 4 bytes are the size of the SWF. The remaining is compressed if the signature is CWS or uncompressed if the signature is FWS. Let's write a simple function to decompress a SWF:

```
private function decompress(data:ByteArray):ByteArray
{
    var header:ByteArray = new ByteArray();
    var compressed:ByteArray = new ByteArray();
    var decompressed:ByteArray = new ByteArray();

    header.writeBytes(data, 3, 5); //read the uncompressed header, excluding t
    compressed.writeBytes(data, 8); //read the rest, compressed

    compressed.uncompress();

    decompressed.writeMultiByte("FWS", "us-ascii"); //mark as uncompressed
    decompressed.writeBytes(header); //write the header back
    decompressed.writeBytes(compressed); //write the now uncompressed content

    return decompressed;
}
```

The function does a few things:

1. It reads the uncompressed header (the first 8 bytes) without the signature and remembers it.
2. It reads the rest of the data and uncompresses it.
3. It writes back the header (with the "FWS" signature) and the uncompressed data, creating a new, uncompressed SWF.

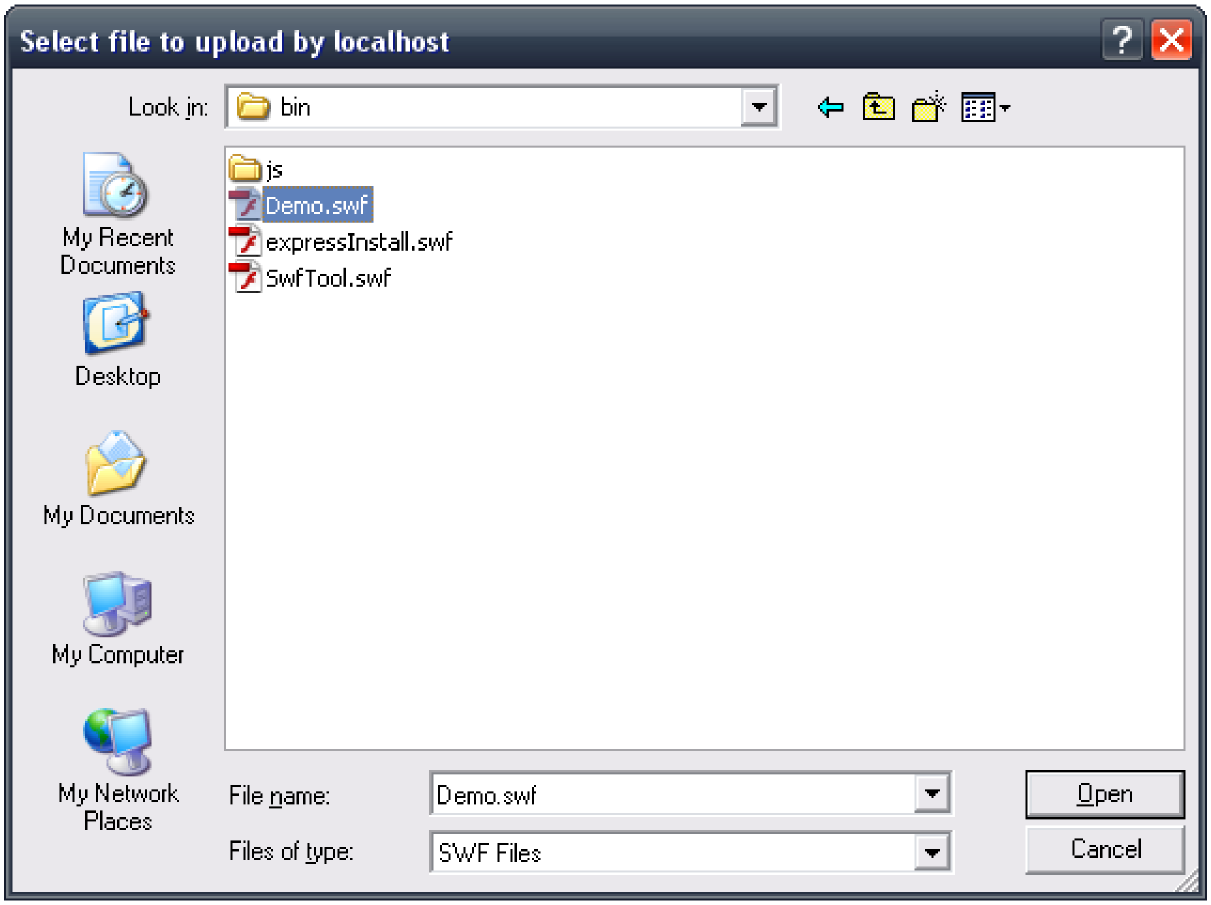
Step 4: Creating a Utility

Next we'll create a handy utility in Flash for compressing and decompressing SWF files. In a new AS3 project, compile the following class as a document class:

```
package
{
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.utils.ByteArray;

    public class Compressor extends Sprite
    {
        private var ref:FileReference;

        public function Compressor()
        {

            ref = new FileReference();
            ref.addEventListener(Event.SELECT, load);
            ref.browse([new FileFilter("SWF Files", "*.swf")]);
        }

        private function load(e:Event):void
        {
            ref.addEventListener(Event.COMPLETE, processSWF);
            ref.load();
        }

        private function processSWF(e:Event):void
        {
            var swf:ByteArray;
            switch(ref.data.readMultiByte(3, "us-ascii"))
            {
                case "CWS":
                    swf = decompress(ref.data);
                    break;
                case "FWS":
                    swf = compress(ref.data);
                    break;
                default:
                    throw Error("Not SWF...");
                    break;
            }

            new FileReference().save(swf);
        }

        private function compress(data:ByteArray):ByteArray
       {
            var header:ByteArray = new ByteArray();
            var decompressed:ByteArray = new ByteArray();
            var compressed:ByteArray = new ByteArray();

            header.writeBytes(data, 3, 5); //read the header, excluding the si
            decompressed.writeBytes(data, 8); //read the rest

            decompressed.compress();

            compressed.writeMultiByte("CWS", "us-ascii"); //mark as compressec
            compressed.writeBytes(header);
            compressed.writeBytes(decompressed);

            return compressed;
        }

        private function decompress(data:ByteArray):ByteArray
        {
            var header:ByteArray = new ByteArray();
            var compressed:ByteArray = new ByteArray();
            var decompressed:ByteArray = new ByteArray();

            header.writeBytes(data, 3, 5); //read the uncompressed header, exc
            compressed.writeBytes(data, 8); //read the rest, compressed

            compressed.uncompress();

            decompressed.writeMultiByte("FWS", "us-ascii"); //mark as uncompre
            decompressed.writeBytes(header); //write the header back
            decompressed.writeBytes(compressed); //write the now uncompressed

            return decompressed;
        }

    }
}
```

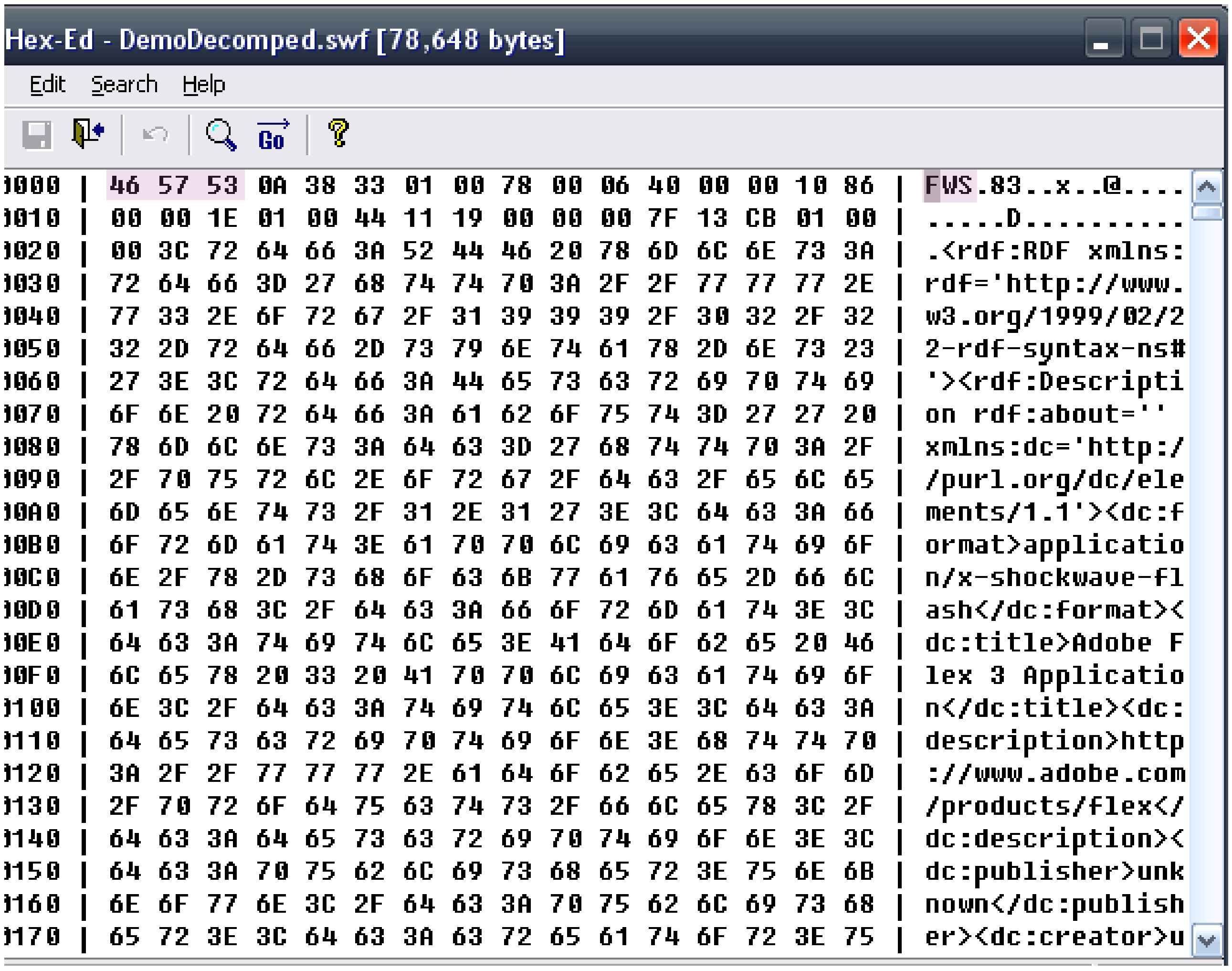
As you probably noticed I've added 2 things: File loading and the compress function.

The compress function is identical to the decompress function, but in reverse. The file loading is done using FileReference (FP10 required) and the loaded file is either compressed or uncompressed. Note that you have to run the SWF locally from a standalone player, as *FileReference.browse()* must be invoked by user interaction (but the local standalone player allows to run it without).



Step 5: Uncompressing the Loading SWF

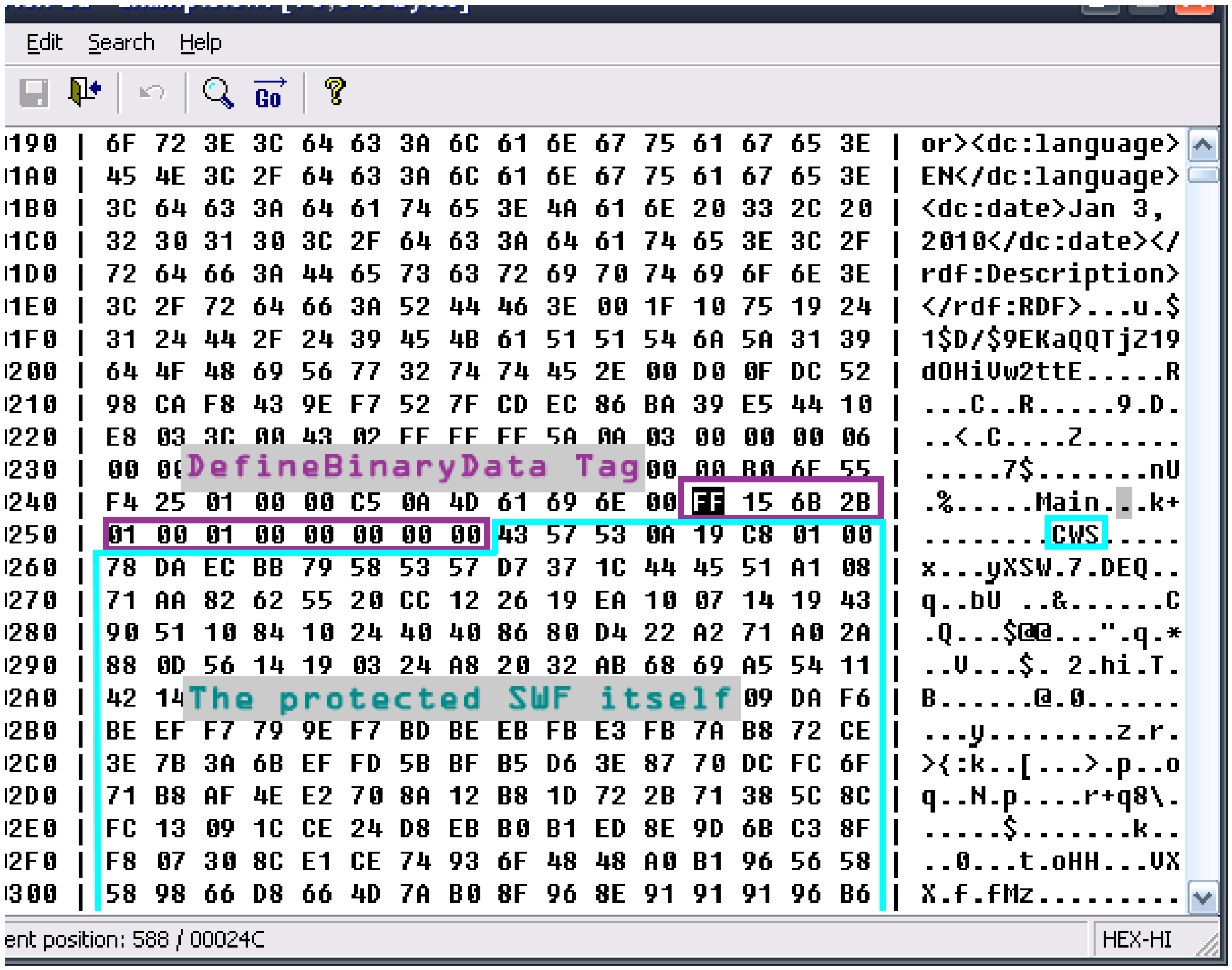
To test the tool, fire it up, select the loading SWF and choose where to save it. Then open it up with a hex editor and scrub through. You should see ascii strings inside like this:



```
)000 | 46 57 53 0A 38 33 01 00 78 00 06 40 00 00 10 86 | FWS.83..x..@....
)010 | 00 00 1E 01 00 44 11 19 00 00 00 7F 13 CB 01 00 | .....D..........
)020 | 00 3C 72 64 66 3A 52 44 46 20 78 6D 6C 6E 73 3A | .<rdf:RDF xmlns:
)030 | 72 64 66 3D 27 68 74 74 70 3A 2F 2F 77 77 77 2E | rdf='http://www.
)040 | 77 33 2E 6F 72 67 2F 31 39 39 39 2F 30 32 2F 32 | w3.org/1999/02/2
)050 | 32 2D 72 64 66 2D 73 79 6E 74 61 78 2D 6E 73 23 | 2-rdf-syntax-ns#
)060 | 27 3E 3C 72 64 66 3A 44 65 73 63 72 69 70 74 69 | '><rdf:Descripti
)070 | 6F 6E 20 72 64 66 3A 61 62 6F 75 74 3D 27 27 20 | on rdf:about=''
)080 | 78 6D 6C 6E 73 3A 64 63 3D 27 68 74 74 70 3A 2F | xmlns:dc='http:/
)090 | 2F 70 75 72 6C 2E 6F 72 67 2F 64 63 2F 65 6C 65 | /purl.org/dc/ele
)0A0 | 6D 65 6E 74 73 2F 31 2E 31 27 3E 3C 64 63 3A 66 | ments/1.1'><dc:f
)0B0 | 6F 72 6D 61 74 3E 61 70 70 6C 69 63 61 74 69 6F | ormat>applicatio
)0C0 | 6E 2F 78 2D 73 68 6F 63 6B 77 61 76 65 2D 66 6C | n/x-shockwave-fl
)0D0 | 61 73 68 3C 2F 64 63 3A 66 6F 72 6D 61 74 3E 3C | ash</dc:format><
)0E0 | 64 63 3A 74 69 74 6C 65 3E 41 64 6F 62 65 20 46 | dc:title>Adobe F
)0F0 | 6C 65 78 20 33 20 41 70 70 6C 69 63 61 74 69 6F | lex 3 Applicatio
)100 | 6E 3C 2F 64 63 3A 74 69 74 6C 65 3E 3C 64 63 3A | n</dc:title><dc:
)110 | 64 65 73 63 72 69 70 74 69 6F 6E 3E 68 74 74 70 | description>http
)120 | 3A 2F 2F 77 77 77 2E 61 64 6F 62 65 2E 63 6F 6D | ://www.adobe.com
)130 | 2F 70 72 6F 64 75 63 74 73 2F 66 6C 65 78 3C 2F | /products/flex</
)140 | 64 63 3A 64 65 73 63 72 69 70 74 69 6F 6E 3E 3C | dc:description><
)150 | 64 63 3A 70 75 62 6C 69 73 68 65 72 3E 75 6E 6B | dc:publisher>unk
)160 | 6E 6F 77 6E 3C 2F 64 63 3A 70 75 62 6C 69 73 68 | nown</dc:publish
)170 | 65 72 3E 3C 64 63 3A 63 72 65 61 74 6F 72 3E 75 | er><dc:creator>u
```

Step 6: Analyze Again

Let's return back to step 2. While the decompiler didn't show any useful info about the protected SWF, it's quite easy to get the SWF from the now uncompressed loader; just search for the signature "CWS" (if the protected SWF is uncompressed search for "FWS") and see the results:



```
190 | 6F 72 3E 3C 64 63 3A 6C 61 6E 67 75 61 67 65 3E | or><dc:language>
1A0 | 45 4E 3C 2F 64 63 3A 6C 61 6E 67 75 61 67 65 3E | EN</dc:language>
1B0 | 3C 64 63 3A 64 61 74 65 3E 4A 61 6E 20 33 2C 20 | <dc:date>Jan 3,
1C0 | 32 30 31 30 3C 2F 64 63 3A 64 61 74 65 3E 3C 2F | 2010</dc:date></
1D0 | 72 64 66 3A 44 65 73 63 72 69 70 74 69 6F 6E 3E | rdf:Description>
1E0 | 3C 2F 72 64 66 3A 52 44 46 3E 00 1F 10 75 19 24 | </rdf:RDF>...u.$
1F0 | 31 24 44 2F 24 39 45 4B 61 51 51 54 6A 5A 31 39 | 1$D/$9EKaQQTjZ19
200 | 64 4F 48 69 56 77 32 74 74 45 2E 00 D0 0F DC 52 | dOHiVw2ttE.....R
210 | 98 CA F8 43 9E F7 52 7F CD EC 86 BA 39 E5 44 10 | ...C..R.....9.D.
220 | E8 03 3C 00 43 02 FF FF FF 5A 00 03 00 00 00 06 | ..<.C....Z......
230 | 00 0                                00 00 BA 6F 55 | .....7$.......nU
240 | F4 25 01 00 00 C5 0A 4D 61 69 6E 00 FF 15 6B 2B | .%.....Main...k+
250 | 01 00 01 00 00 00 00 00 43 57 53 0A 19 C8 01 00 | ........CWS.....
260 | 78 DA EC BB 79 58 53 57 D7 37 1C 44 45 51 A1 08 | x...yXSW.7.DEQ..
270 | 71 AA 82 62 55 20 CC 12 26 19 EA 10 07 14 19 43 | q..bU ..&......C
280 | 90 51 10 84 10 24 40 40 86 80 D4 22 A2 71 A0 2A | .Q...$@@...".q.*
290 | 88 0D 56 14 19 03 24 A8 20 32 AB 68 69 A5 54 11 | ..V...$. 2.hi.T.
2A0 | 42 14                                  09 DA F6 | B......@.0......
2B0 | BE EF F7 79 9E F7 BD BE EB FB E3 FB 7A B8 72 CE | ...y........z.r.
2C0 | 3E 7B 3A 6B EF FD 5B BF B5 D6 3E 87 70 DC FC 6F | >{:k..[...>.p..o
2D0 | 71 B8 AF 4E E2 70 8A 12 B8 1D 72 2B 71 38 5C 8C | q..N.p....r+q8\.
2E0 | FC 13 09 1C CE 24 D8 EB B0 B1 ED 8E 9D 6B C3 8F | .....$.......k..
2F0 | F8 07 30 8C E1 CE 74 93 6F 48 48 A0 B1 96 56 58 | ..0...t.oHH...VX
300 | 58 98 66 D8 66 4D 7A B0 8F 96 8E 91 91 91 96 B6 | X.f.fMz.........
ent position: 588 / 00024C                         HEX-HI
```

What we found is a DefineBinaryData tag that contains the protected SWF, and extracting it from there is a piece of cake. We are about to add another layer of protection over the loading SWF : Encryption.

Step 7: Encryption

To make the protected SWF less "accessible" we will add some kind of encryption. I chose to use as3crypto and you can download it from code.google.com. You can use any library you want instead (or your own implementation, even better), the only requirement is that it should be able to encrypt and decrypt binary data using a key.

Step 8: Encrypting Data

The first thing we want to do is write a utility to encrypt the protected SWF before we embed it. It requires very basic knowledge of the as3crypto library and it's pretty straightforward. Add the library into your library path and let's begin by writing the following:

```
var aes:AESKey = new AESKey(binKey);
var bytesToEncrypt:int = (data.length & ~15); //make sure that it can be devide
for (var i:int = 0; i < bytesToEncrypt; i += 16)
        aes.encrypt(data, i);
```

What's going on here? We use a class from as3crypto called AESKey to encrypt the content. The class encrypts 16 bytes in a time (128-bit), and we have to for-loop over the data to encrypt it all. Note the second line : data.length & ~15. It makes sure that the number of bytes encrypted can be divided by 16 and we don't run out of data when calling *aes.encrypt()*.

Note: It's important to understand the point of encryption in this case. It's not really encryption, but rather obfuscation since we include the key inside the SWF. The purpose is to turn the data into binary rubbish, and the code above does it's job, although it can leave up to 15 unencrypted bytes (which doesn't matter in our case). I'm not a cryptographer, and I'm quite sure that the above code could look lame and weak from a cryptographer's perspective, but as I said it's quite irrelevant as we include the key inside the SWF.

Step 9: Encryption Utility

Time to create another utility that will help us encrypt SWF files. It's almost the same as the compressor we created earlier, so I won't talk much about it. Compile it in a new project as a document class:

```
package
{
    import com.hurlant.crypto.symmetric.AESKey;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.net.FileReference;
    import flash.utils.ByteArray;

    public class Encryptor extends Sprite
    {
        private var key:String = "activetuts"; //I hardcoded the key

        private var ref:FileReference;

        public function Encryptor()
        {
            ref = new FileReference();
            ref.addEventListener(Event.SELECT, load);
            ref.browse();
        }

        private function load(e:Event):void
        {
            ref.addEventListener(Event.COMPLETE, encrypt);
            ref.load();
        }

        private function encrypt(e:Event):void
        {
            var data:ByteArray = ref.data;

            var binKey:ByteArray = new ByteArray();
            binKey.writeUTF(key); //AESKey requires binary key

            var aes:AESKey = new AESKey(binKey);
            var bytesToEncrypt:int = (data.length & ~15); //make sure that it
            for (var i:int = 0; i < bytesToEncrypt; i += 16)
                aes.encrypt(data, i);

            new FileReference().save(data);
        }

    }

}
```

Now run it, and make an encrypted copy of the protected SWF by selecting it first and then saving it under a different name.

Step 10: Modifying the Loader

Return back to the loading SWF project. Because the content is now encrypted we need to modify the loading SWF and add decryption code into it. Don't forget to change the src in the Embed tag to point to the encrypted SWF.

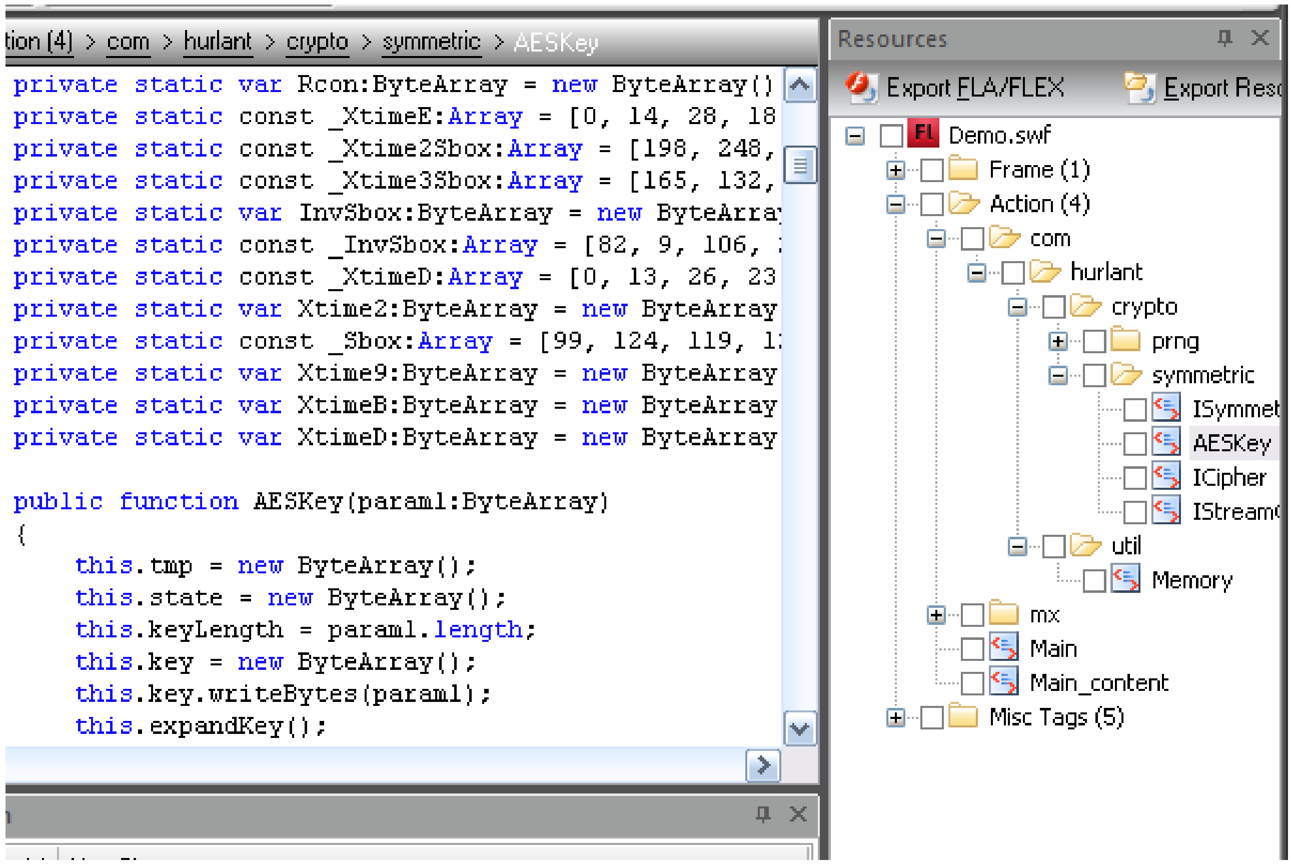
```
package
```

```
{
    import com.hurlant.crypto.symmetric.AESKey;
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;
    import flash.utils.ByteArray;

    [SWF (width = 640, height = 423)] //the dimensions should be same as the l
    public class Main extends Sprite
    {
        [Embed (source = "VerletClothEn.swf", mimeType = "application/octet-st
        // source = path to the swf you want to protect
        private var content:Class;

        private var key:String = "activetuts";

        public function Main():void
        {
            var data:ByteArray = new content();

            var binKey:ByteArray = new ByteArray();
            binKey.writeUTF(key); //AESKey requires binary key

            var aes:AESKey = new AESKey(binKey);
            var bytesToDecrypt:int = (data.length & ~15); //make sure that it
            for (var i:int = 0; i < bytesToDecrypt; i += 16)
                aes.decrypt(data, i);

            var loader:Loader = new Loader();
            addChild(loader);
            loader.loadBytes(data, new LoaderContext(false, new ApplicationDom
        }
    }

}
```

This is the same as before except with the decryption code stuck in the middle. Now compile the loading SWF and test if it works. If you followed carefully up to now, the protected SWF should load and display without errors.

Step 11: Look Inside Using a Decompiler

Open the new loading SWF with a decompiler and have a look.


```
private static var Rcon:ByteArray = new ByteArray()
private static const _XtimeE:Array = [0, 14, 28, 18
private static const _Xtime2Sbox:Array = [198, 248,
private static const _Xtime3Sbox:Array = [165, 132,
private static var InvSbox:ByteArray = new ByteArray
private static const _InvSbox:Array = [82, 9, 106,
private static const _XtimeD:Array = [0, 13, 26, 23
private static var Xtime2:ByteArray = new ByteArray
private static const _Sbox:Array = [99, 124, 119, 1
private static var Xtime9:ByteArray = new ByteArray
private static var XtimeB:ByteArray = new ByteArray
private static var XtimeD:ByteArray = new ByteArray

public function AESKey(param1:ByteArray)
{
    this.tmp = new ByteArray();
    this.state = new ByteArray();
    this.keyLength = param1.length;
    this.key = new ByteArray();
    this.key.writeBytes(param1);
    this.expandKey();
```

It contains over a thousand lines of tough looking encryption code, and it's probably harder to get the protected SWF out of it. We've added a few more steps the attacker must undertake:

1. He (or she) has to find the DefineBinaryData that holds the encrypted content and extract it.
2. He must create a utility to decrypt it.

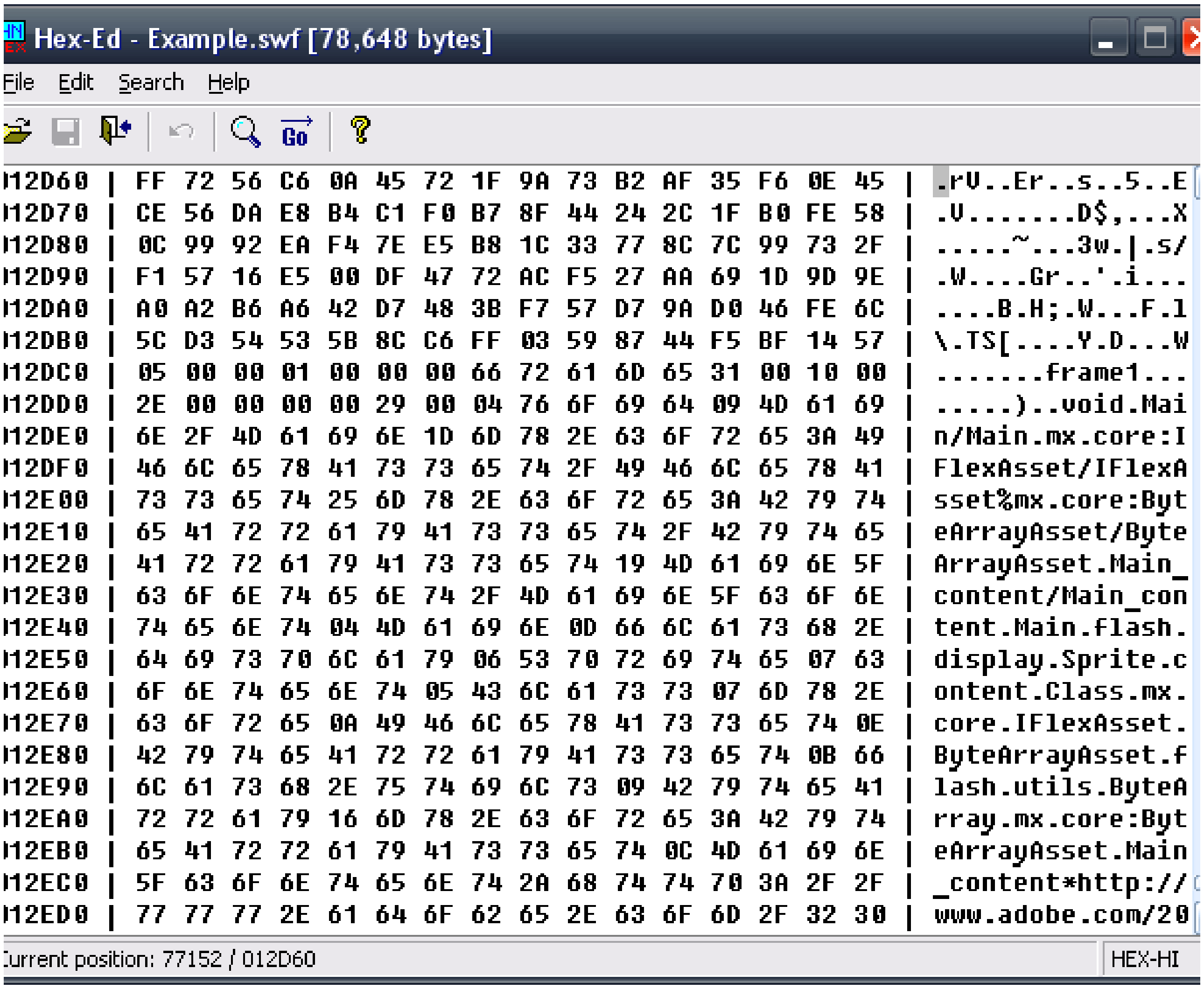
The problem is that creating a utility is as simple as copy-pasting from the decompiler into the code editor and tweaking the code a little bit. I tried to break my protection myself, and it was quite easy - I managed to do it in about 5 minutes. So we're going to have to take some measurements against it.

Step 12: **String Obfuscation**

First we'd put the protected SWF into the loading SWF, then encrypted it, and now we'll put the final touches to the loading SWF. We'll rename classes, functions and variables to illegal names.

By saying *illegal names* I mean names such as ,;!@@,^#^ and (^_^). The cool thing is that this matters to the compiler but not to the Flash Player. When the compiler encounters illegal characters inside identifiers, it fails to parse them and thus the project fails to compile. On the other hand, the Player doesn't have any problems with those illegal names. We can compile the SWF with legal identifiers, decompress it and rename them to a bunch of meaningless illegal symbols. The decompiler will output illegal code and the attacker will have to go over the hundreds of lines of code manually, removing illegal identifiers before he can compile it. He deserves it!

This is how it looks before any string obfuscation:



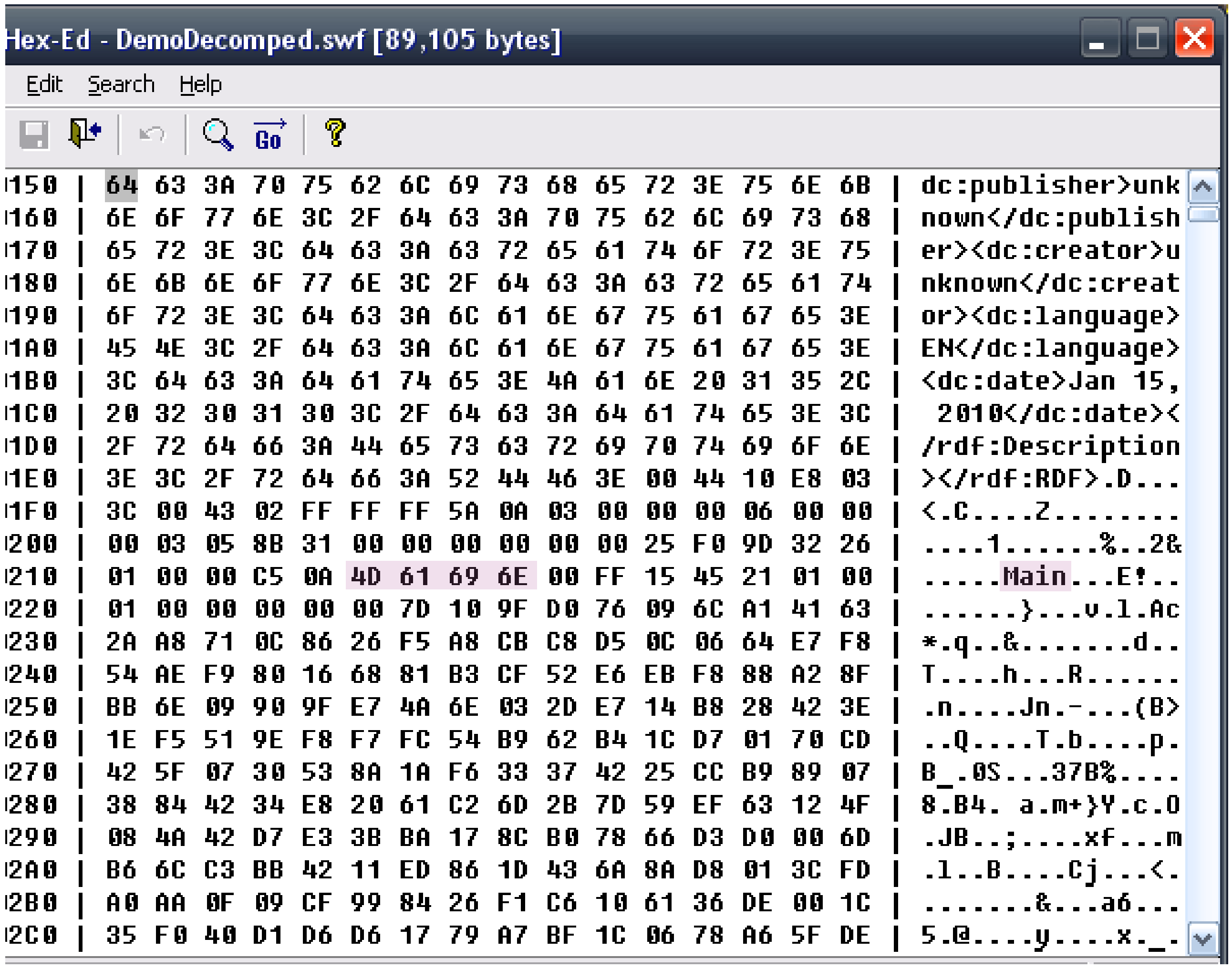
```
12D60 | FF 72 56 C6 0A 45 72 1F 9A 73 B2 AF 35 F6 0E 45 | .rV..Er..s..5..E
12D70 | CE 56 DA E8 B4 C1 F0 B7 8F 44 24 2C 1F B0 FE 58 | .V.......D$,...X
12D80 | 0C 99 92 EA F4 7E E5 B8 1C 33 77 8C 7C 99 73 2F | .....~...3w.|.s/
12D90 | F1 57 16 E5 00 DF 47 72 AC F5 27 AA 69 1D 9D 9E | .W....Gr..'.i...
12DA0 | A0 A2 B6 A6 42 D7 48 3B F7 57 D7 9A D0 46 FE 6C | ....B.H;.W...F.l
12DB0 | 5C D3 54 53 5B 8C C6 FF 03 59 87 44 F5 BF 14 57 | \.TS[....Y.D...W
12DC0 | 05 00 00 01 00 00 00 66 72 61 6D 65 31 00 10 00 | .......frame1...
12DD0 | 2E 00 00 00 00 29 00 04 76 6F 69 64 09 4D 61 69 | .....)..void.Mai
12DE0 | 6E 2F 4D 61 69 6E 1D 6D 78 2E 63 6F 72 65 3A 49 | n/Main.mx.core:I
12DF0 | 46 6C 65 78 41 73 73 65 74 2F 49 46 6C 65 78 41 | FlexAsset/IFlexA
12E00 | 73 73 65 74 25 6D 78 2E 63 6F 72 65 3A 42 79 74 | sset%mx.core:Byt
12E10 | 65 41 72 72 61 79 41 73 73 65 74 2F 42 79 74 65 | eArrayAsset/Byte
12E20 | 41 72 72 61 79 41 73 73 65 74 19 4D 61 69 6E 5F | ArrayAsset.Main_
12E30 | 63 6F 6E 74 65 6E 74 2F 4D 61 69 6E 5F 63 6F 6E | content/Main_con
12E40 | 74 65 6E 74 04 4D 61 69 6E 0D 66 6C 61 73 68 2E | tent.Main.flash.
12E50 | 64 69 73 70 6C 61 79 06 53 70 72 69 74 65 07 63 | display.Sprite.c
12E60 | 6F 6E 74 65 6E 74 05 43 6C 61 73 73 07 6D 78 2E | ontent.Class.mx.
12E70 | 63 6F 72 65 0A 49 46 6C 65 78 41 73 73 65 74 0E | core.IFlexAsset.
12E80 | 42 79 74 65 41 72 72 61 79 41 73 73 65 74 0B 66 | ByteArrayAsset.f
12E90 | 6C 61 73 68 2E 75 74 69 6C 73 09 42 79 74 65 41 | lash.utils.ByteA
12EA0 | 72 72 61 79 16 6D 78 2E 63 6F 72 65 3A 42 79 74 | rray.mx.core:Byt
12EB0 | 65 41 72 72 61 79 41 73 73 65 74 0C 4D 61 69 6E | eArrayAsset.Main
12EC0 | 5F 63 6F 6E 74 65 6E 74 2A 68 74 74 70 3A 2F 2F | _content*http://
12ED0 | 77 77 77 2E 61 64 6F 62 65 2E 63 6F 6D 2F 32 30 | www.adobe.com/20
```

urrent position: 77152 / 012D60

Let's start! Decompress the loading SWF using the utility we created before and fire up a hex editor.

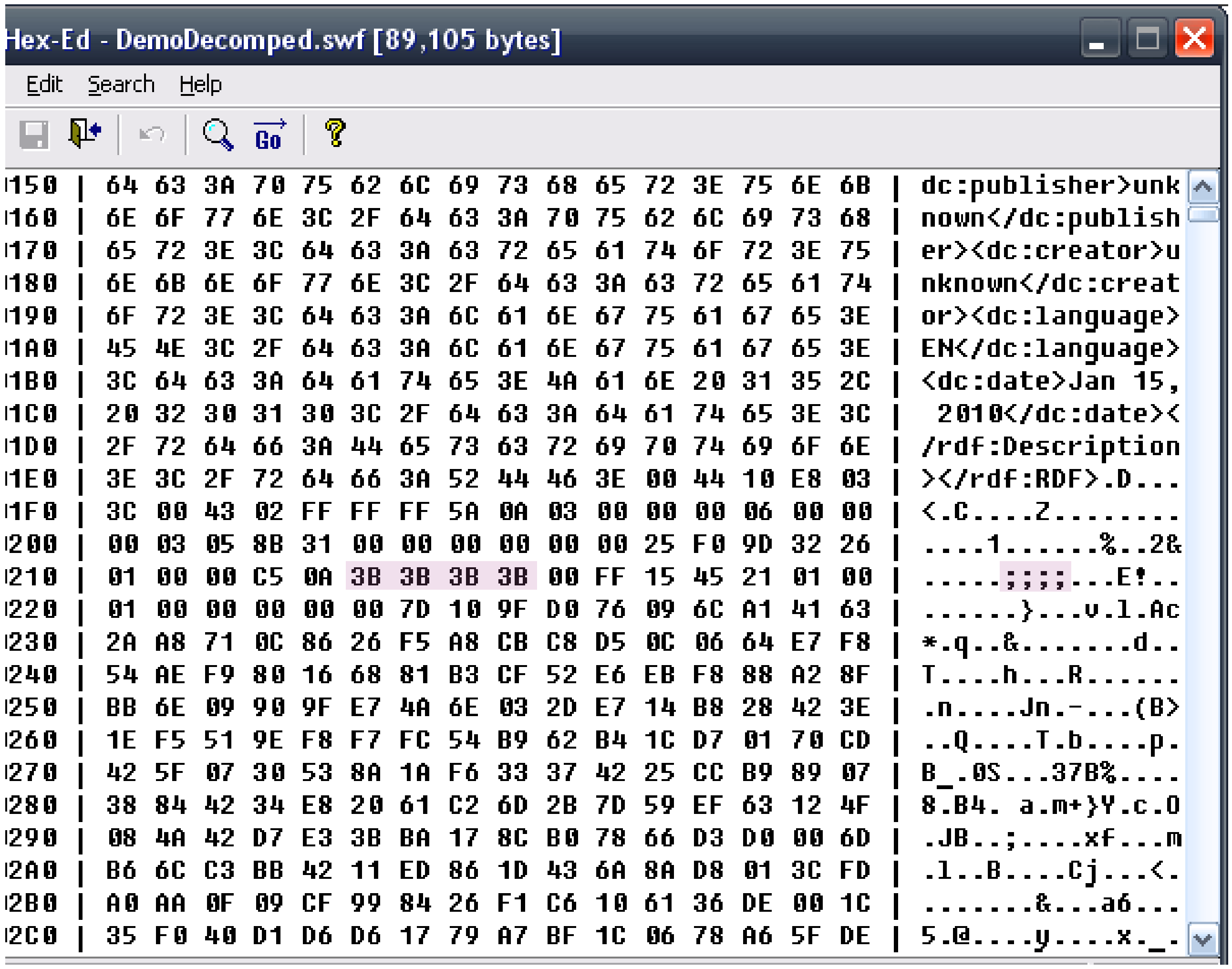
Step 13: Your First Obfuscation

Let's try to rename the document class. Assuming you've left the original name (Main), let's search for it in the uncompressed loader SWF with a hex editor:



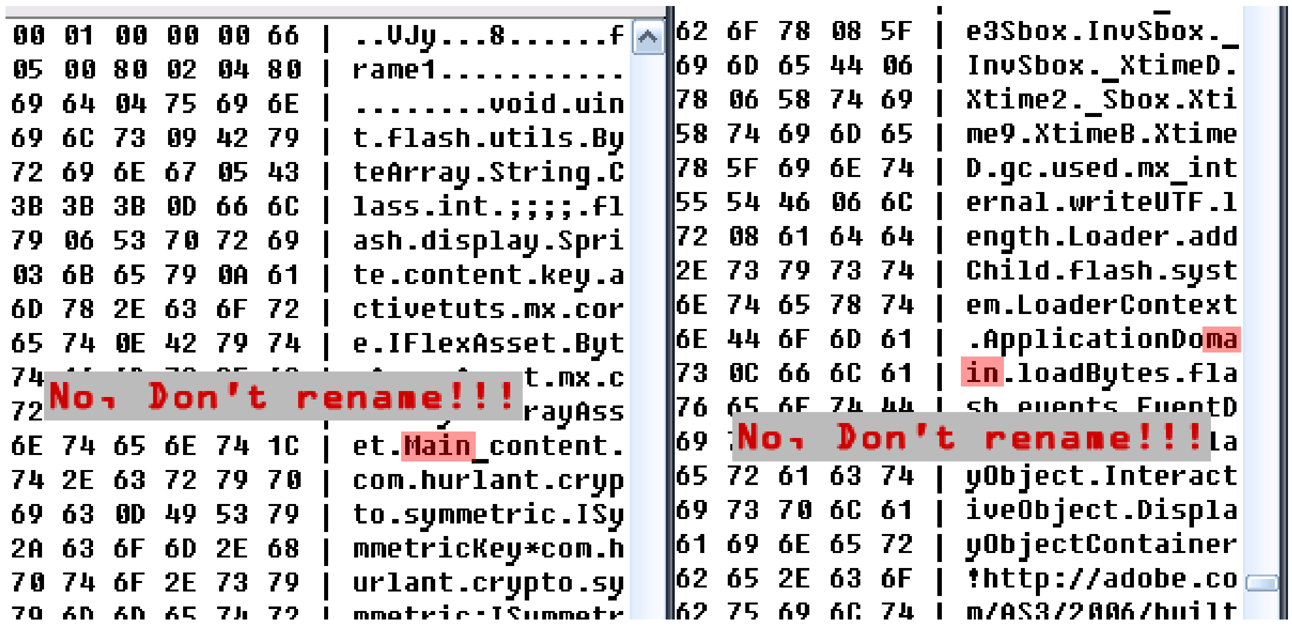
```
150 | 64 63 3A 70 75 62 6C 69 73 68 65 72 3E 75 6E 6B | dc:publisher>unk
160 | 6E 6F 77 6E 3C 2F 64 63 3A 70 75 62 6C 69 73 68 | nown</dc:publish
170 | 65 72 3E 3C 64 63 3A 63 72 65 61 74 6F 72 3E 75 | er><dc:creator>u
180 | 6E 6B 6E 6F 77 6E 3C 2F 64 63 3A 63 72 65 61 74 | nknown</dc:creat
190 | 6F 72 3E 3C 64 63 3A 6C 61 6E 67 75 61 67 65 3E | or><dc:language>
1A0 | 45 4E 3C 2F 64 63 3A 6C 61 6E 67 75 61 67 65 3E | EN</dc:language>
1B0 | 3C 64 63 3A 64 61 74 65 3E 4A 61 6E 20 31 35 2C | <dc:date>Jan 15,
1C0 | 20 32 30 31 30 3C 2F 64 63 3A 64 61 74 65 3E 3C |  2010</dc:date><
1D0 | 2F 72 64 66 3A 44 65 73 63 72 69 70 74 69 6F 6E | /rdf:Description
1E0 | 3E 3C 2F 72 64 66 3A 52 44 46 3E 00 44 10 E8 03 | ></rdf:RDF>.D...
1F0 | 3C 00 43 02 FF FF FF 5A 0A 03 00 00 00 06 00 00 | <.C....Z........
200 | 00 03 05 8B 31 00 00 00 00 00 00 25 F0 9D 32 26 | ....1......%..2&
210 | 01 00 00 C5 0A 4D 61 69 6E 00 FF 15 45 21 01 00 | .....Main...E!..
220 | 01 00 00 00 00 00 7D 10 9F D0 76 09 6C A1 41 63 | ......}...v.l.Ac
230 | 2A A8 71 0C 86 26 F5 A8 CB C8 D5 0C 06 64 E7 F8 | *.q..&.......d..
240 | 54 AE F9 80 16 68 81 B3 CF 52 E6 EB F8 88 A2 8F | T....h...R......
250 | BB 6E 09 90 9F E7 4A 6E 03 2D E7 14 B8 28 42 3E | .n....Jn.-...(B>
260 | 1E F5 51 9E F8 F7 FC 54 B9 62 B4 1C D7 01 70 CD | ..Q....T.b....p.
270 | 42 5F 07 30 53 8A 1A F6 33 37 42 25 CC B9 89 07 | B_.0S...37B%....
280 | 38 84 42 34 E8 20 61 C2 6D 2B 7D 59 EF 63 12 4F | 8.B4. a.m+}Y.c.O
290 | 08 4A 42 D7 E3 3B BA 17 8C B0 78 66 D3 D0 00 6D | .JB..;....xf...m
2A0 | B6 6C C3 BB 42 11 ED 86 1D 43 6A 8A D8 01 3C FD | .l..B....Cj...<.
2B0 | A0 AA 0F 09 CF 99 84 26 F1 C6 10 61 36 DE 00 1C | .......&...a6...
2C0 | 35 F0 40 D1 D6 D6 17 79 A7 BF 1C 06 78 A6 5F DE | 5.@....y....x._.
```

Rename "*Main*" to *;;;;*. Now search for other "Main"s and rename them to *;;;;* too.



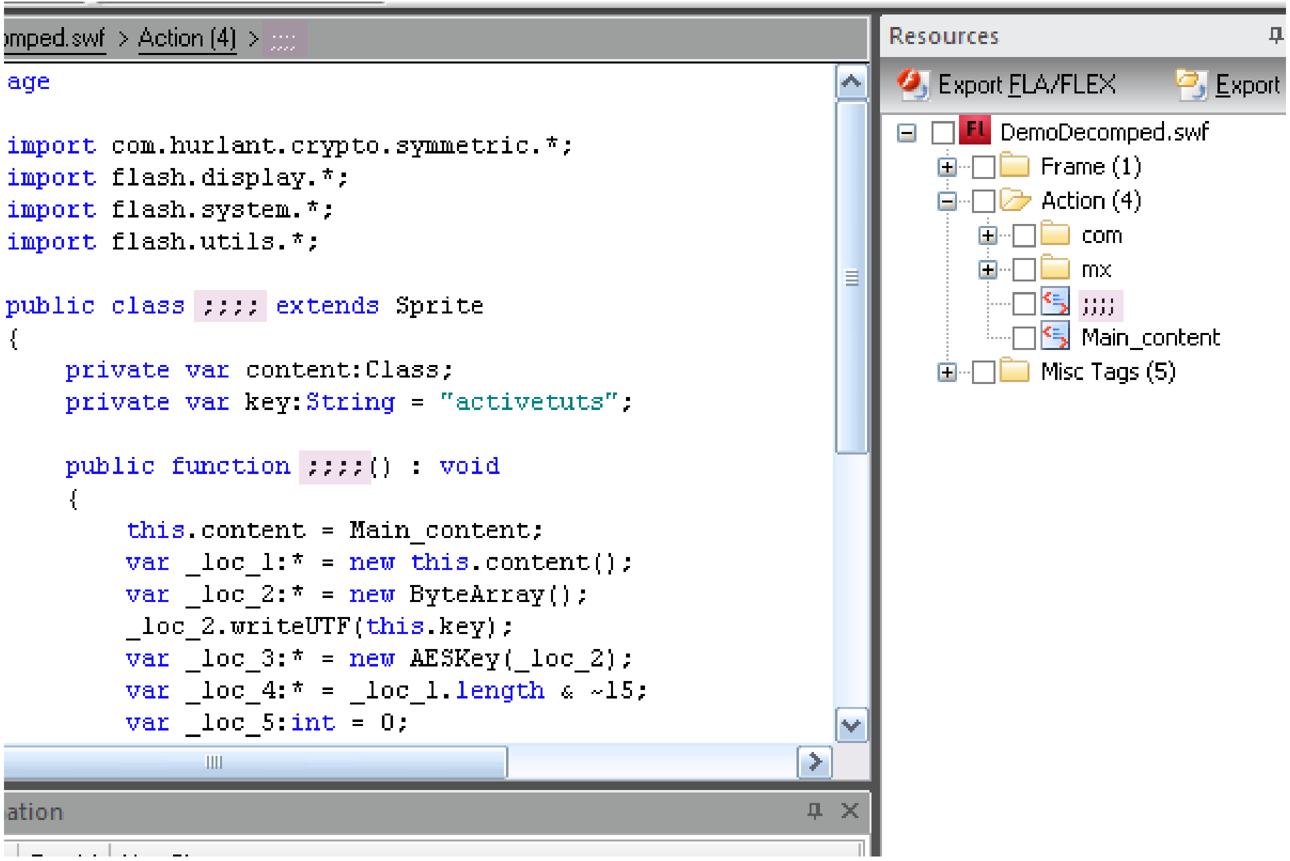
```
150 | 64 63 3A 70 75 62 6C 69 73 68 65 72 3E 75 6E 6B | dc:publisher>unk
160 | 6E 6F 77 6E 3C 2F 64 63 3A 70 75 62 6C 69 73 68 | nown</dc:publish
170 | 65 72 3E 3C 64 63 3A 63 72 65 61 74 6F 72 3E 75 | er><dc:creator>u
180 | 6E 6B 6E 6F 77 6E 3C 2F 64 63 3A 63 72 65 61 74 | nknown</dc:creat
190 | 6F 72 3E 3C 64 63 3A 6C 61 6E 67 75 61 67 65 3E | or><dc:language>
1A0 | 45 4E 3C 2F 64 63 3A 6C 61 6E 67 75 61 67 65 3E | EN</dc:language>
1B0 | 3C 64 63 3A 64 61 74 65 3E 4A 61 6E 20 31 35 2C | <dc:date>Jan 15,
1C0 | 20 32 30 31 30 3C 2F 64 63 3A 64 61 74 65 3E 3C |  2010</dc:date><
1D0 | 2F 72 64 66 3A 44 65 73 63 72 69 70 74 69 6F 6E | /rdf:Description
1E0 | 3E 3C 2F 72 64 66 3A 52 44 46 3E 00 44 10 E8 03 | ></rdf:RDF>.D...
1F0 | 3C 00 43 02 FF FF FF 5A 0A 03 00 00 00 06 00 00 | <.C....Z........
200 | 00 03 05 8B 31 00 00 00 00 00 00 25 F0 9D 32 26 | ....1......%..2&
210 | 01 00 00 C5 0A 3B 3B 3B 3B 00 FF 15 45 21 01 00 | .....;;;;...E!..
220 | 01 00 00 00 00 00 7D 10 9F D0 76 09 6C A1 41 63 | ......}...v.l.Ac
230 | 2A A8 71 0C 86 26 F5 A8 CB C8 D5 0C 06 64 E7 F8 | *.q..&.......d..
240 | 54 AE F9 80 16 68 81 B3 CF 52 E6 EB F8 88 A2 8F | T....h...R......
250 | BB 6E 09 90 9F E7 4A 6E 03 2D E7 14 B8 28 42 3E | .n....Jn.-...(B>
260 | 1E F5 51 9E F8 F7 FC 54 B9 62 B4 1C D7 01 70 CD | ..Q....T.b....p.
270 | 42 5F 07 30 53 8A 1A F6 33 37 42 25 CC B9 89 07 | B_.0S...37B%....
280 | 38 84 42 34 E8 20 61 C2 6D 2B 7D 59 EF 63 12 4F | 8.B4. a.m+}Y.c.O
290 | 08 4A 42 D7 E3 3B BA 17 8C B0 78 66 D3 D0 00 6D | .JB..;....xf...m
2A0 | B6 6C C3 BB 42 11 ED 86 1D 43 6A 8A D8 01 3C FD | .l..B....Cj...<.
2B0 | A0 AA 0F 09 CF 99 84 26 F1 C6 10 61 36 DE 00 1C | .......&...a6...
2C0 | 35 F0 40 D1 D6 D6 17 79 A7 BF 1C 06 78 A6 5F DE | 5.@....y....x._.
```

When renaming make sure that you don't rename unnecessary strings or the SWF will not run.



```
00 01 00 00 00 66 | ..VJy...8......f    62 6F 78 08 5F | e3Sbox.InvSbox._
05 00 80 02 04 80 | rame1...........    69 6D 65 44 06 | InvSbox._XtimeD.
69 64 04 75 69 6E | ........void.uin    78 06 58 74 69 | Xtime2._Sbox.Xti
69 6C 73 09 42 79 | t.flash.utils.By    58 74 69 6D 65 | me9.XtimeB.Xtime
72 69 6E 67 05 43 | teArray.String.C    78 5F 69 6E 74 | D.gc.used.mx_int
3B 3B 3B 0D 66 6C | lass.int.;;;;.fl    55 54 46 06 6C | ernal.writeUTF.l
79 06 53 70 72 69 | ash.display.Spri    72 08 61 64 64 | ength.Loader.add
03 6B 65 79 0A 61 | te.content.key.a    2E 73 79 73 74 | Child.flash.syst
6D 78 2E 63 6F 72 | ctivetuts.mx.cor    6E 74 65 78 74 | em.LoaderContext
65 74 0E 42 79 74 | e.IFlexAsset.Byt    6E 44 6F 6D 61 | .ApplicationDoma
74                        t.mx.c    73 0C 66 6C 61 | in.loadBytes.fla
72                        rayAss    76 65 6F 74 44 | sh.events.EventD
6E 74 65 6E 74 1C | et.Main_content.    69                       la
74 2E 63 72 79 70 | com.hurlant.cryp    65 72 61 63 74 | yObject.Interact
69 63 0D 49 53 79 | to.symmetric.ISy    69 73 70 6C 61 | iveObject.Displa
2A 63 6F 6D 2E 68 | mmetricKey*com.h    61 69 6E 65 72 | yObjectContainer
70 74 6F 2E 73 79 | urlant.crypto.sy    62 65 2E 63 6F | !http://adobe.co
79 6D 6D 65 74 72 | mmetric:ISymmetr    62 75 69 6C 74 | m/AS3/2006/built
```

Save and run the SWF. It works! And look what the decompiler says:

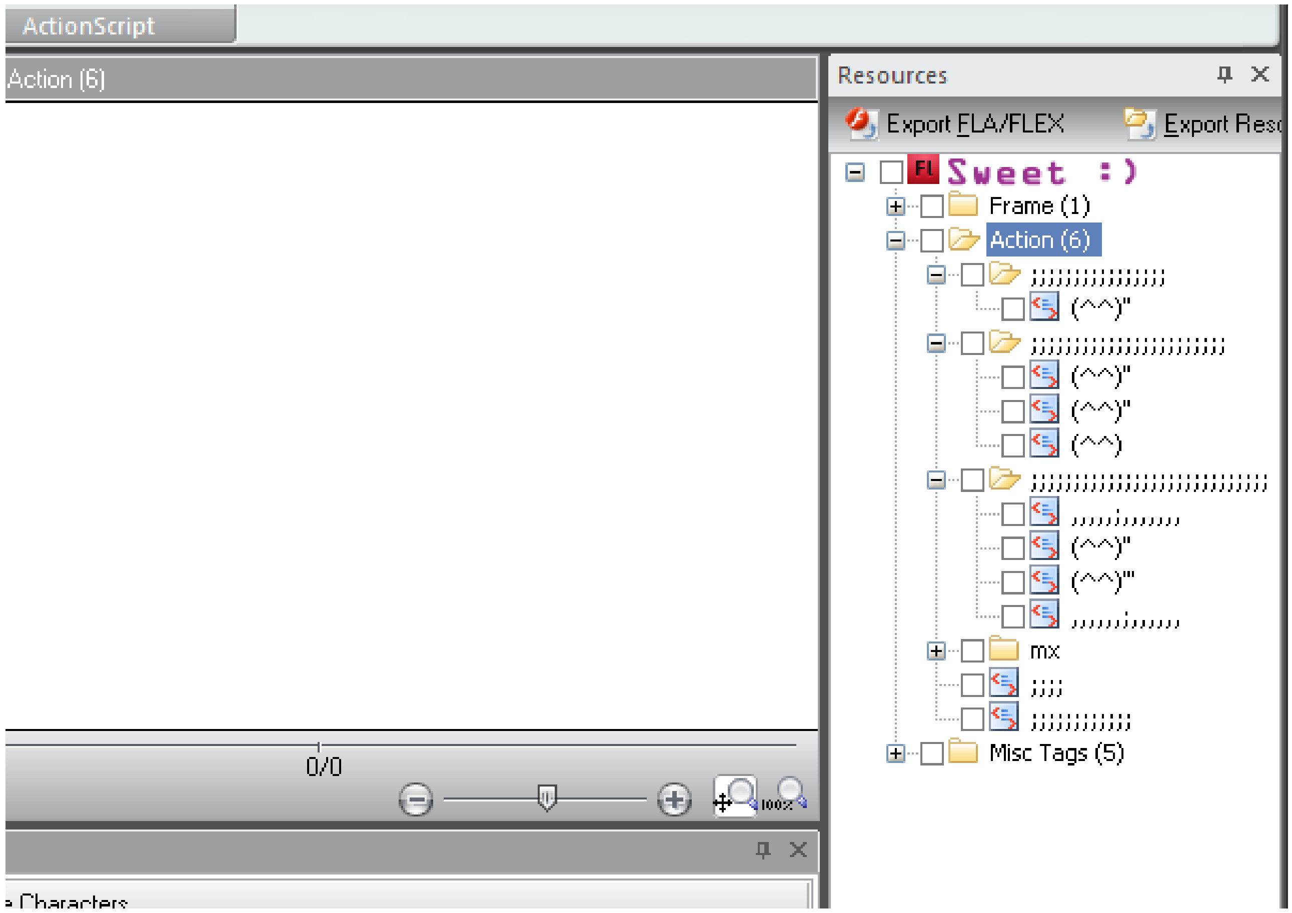


```
age

import com.hurlant.crypto.symmetric.*;
import flash.display.*;
import flash.system.*;
import flash.utils.*;

public class ;;;; extends Sprite
{
    private var content:Class;
    private var key:String = "activetuts";

    public function ;;;;() : void
    {
        this.content = Main_content;
        var _loc_1:* = new this.content();
        var _loc_2:* = new ByteArray();
        _loc_2.writeUTF(this.key);
        var _loc_3:* = new AESKey(_loc_2);
        var _loc_4:* = _loc_1.length & ~15;
        var _loc_5:int = 0;
```

Victory!! :)

Step 14: Renaming the Rest of the Classes

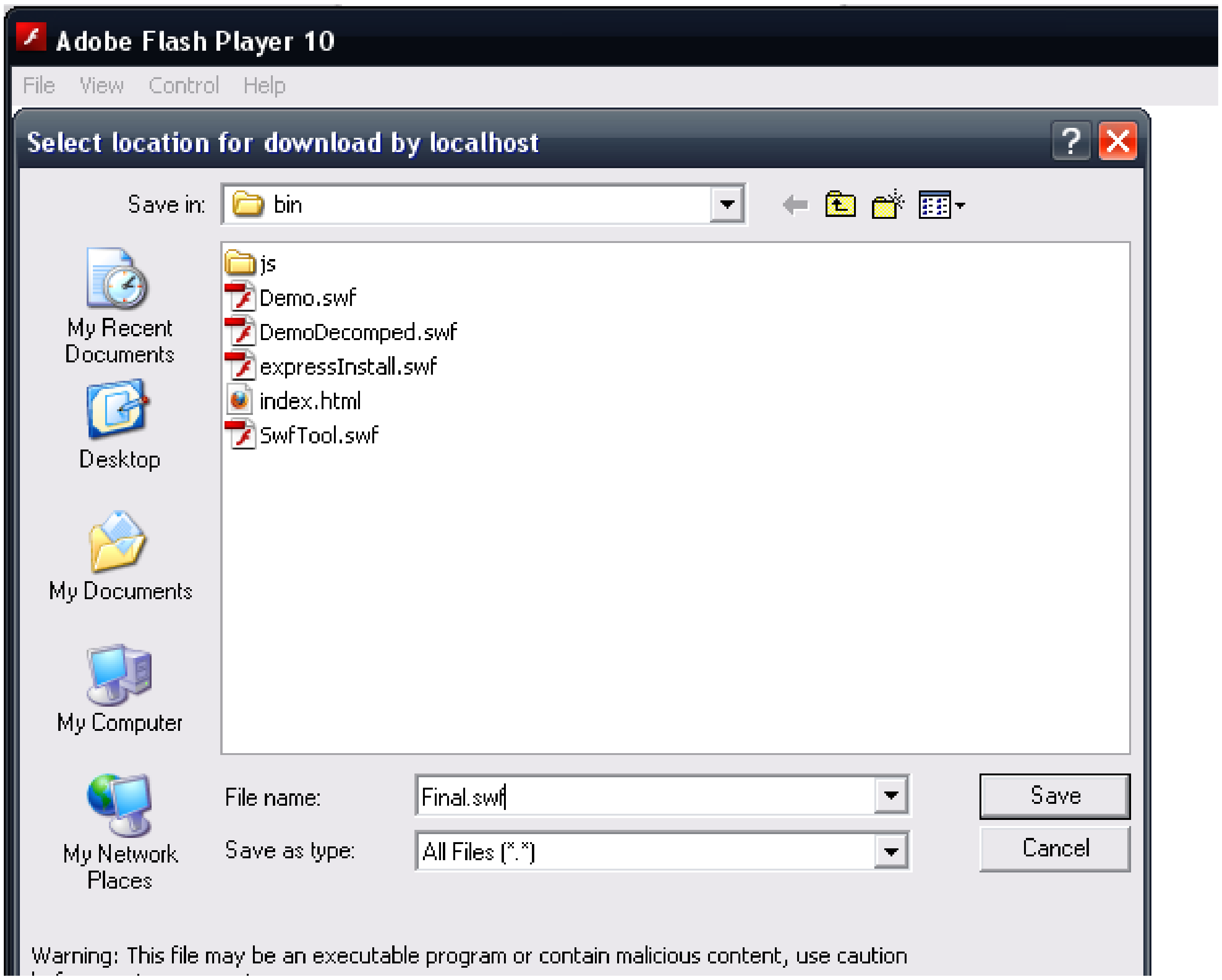
Keep renaming the rest of your classes. Choose a class name and search for it, replacing it with illegal symbols until you reach the end of the file. As I said, the most important thing here is to use your common sense, make sure you don't mess your SWF up. After renaming the classes you can start renaming the packages. Note that when renaming a package, you can erase the periods too and make it one long illegal package name. Look what I made:



After you finish renaming the classes and the packages, you can start renaming functions and variables. They are even easier to rename as they usually appear only once, in one large cloud. Again, make sure you rename only "your" methods and not the built-in Flash methods. Make sure you don't wipe out the key ("activetuts" in our case).

Step 15: Compress the SWF

After you finish renaming you would probably want to compress the SWF so it will be smaller in size. No problem, we can use the compressing utility we created before and it will do the job. Run the utility, select the SWF and save it under another name.



FUEL YOUR SUCCESS

TIMESJOBS.COM

KNOW MORE

Advertisement

Conclusion: Have a Final Look

Open it one last time and have a look. The classes, the variables and the method names are obfuscated and the protected SWF is somewhere inside, encrypted. This technique could be slow to apply at first, but after a few times it takes only a few minutes.

A while ago I created an automatic utility to inject the protected SWF for me into the loading SWF, and it worked fine. The only problem is that if it can be injected using an automatic utility, it can be decrypted using another utility, so if the attacker makes a utility for that he will get all your SWF easily. Because of this I prefer to protect the SWFs manually each time, adding a slight modification so it would be harder to automate.

Another nice application of the technique is *Domain locking*. Instead of decrypting the SWF with a constant string you can decrypt it with the domain the SWF is currently running on. So instead of having an if statement to check the domain, you can introduce a more powerful way to protect the SWF from placement on other sites.

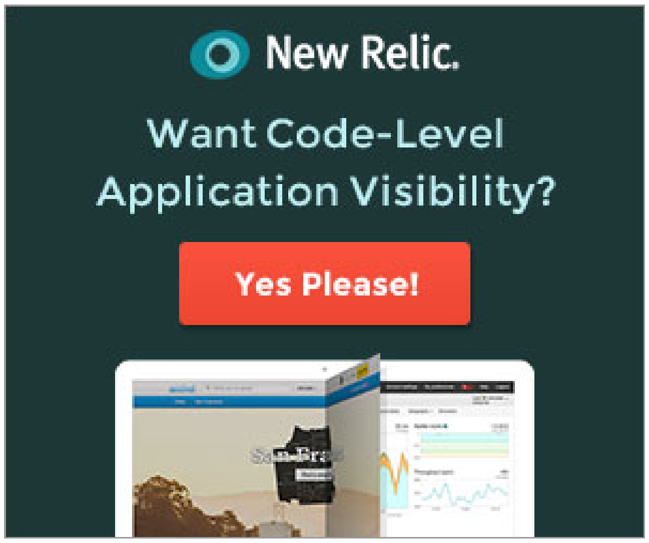
Last thing, you may want to replace the encryption code with your own implementation. Why? We invested efforts in making the crypto code illegal, but the code we use is from a popular open source library and the attacker could recognize it as such. He will download a clean copy, and all the obfuscation work is rendered unnecessary. On the other hand, using your own implementation will require him to fix all the illegal names before he can continue.

Other Protection Methods

Because SWF theft is a big problem in the Flash world, there are other options for protecting SWFs. There are numerous programs out there to obfuscate AS on the bytecode level (like Kindisoft's secureSWF). They mess up the compiled bytecode and when the decompiler attempts to output code it will fail, and even crash sometimes. Of course this protection is better in terms of security but it costs $$$, so before choosing how to protect your SWF consider the amount of security needed. If it's about protecting a proprietary algorithm your 50-employee Flash studio has been developing for the past two years, you may consider something better then renaming the variables. On the other hand if you want to prevent the kiddies from submitting false high scores you may consider using this technique.

What I like about this technique is the fact that your protected SWF is left untouched when run. AS obfuscation tampers with the byte code and it could possibly damage the SWF and cause bugs (although I haven't encountered any myself).

That's all for today, hope you enjoyed the tutorial and learned something new! If you have any questions feel free to drop a comment.



Advertisement

Tutorial Details
Difficulty: Intermediate
Tags: Workflow, Flash

Download Source Files

View Online Demo

About the Author

Nikita is a Flash developer from Israel who has been involved in Flash since 2006. He's especially interested in
more…