Steganographic Techniques for Hiding Data in SWF Files

Mark-Anthony Fouche, Martin Olivier

Chapter 19

STEGANOGRAPHIC TECHNIQUES FOR HIDING DATA IN SWF FILES

Mark-Anthony Fouche and Martin Olivier

Abstract Small Web Format (SWF) or Flash files are widely used on the Internet to provide Rich Internet Applications (RIAs). This makes SWF files an excellent candidate for disseminating hidden data. However, digital forensic investigators are unable to detect and extract the hidden data because limited information is available about the techniques used to hide data in SWF files. This paper investigates several data insertion techniques for hiding data in SWF files. The techniques include appending data to an SWF file, adding an extra Metadata tag, creating a custom Definition tag, and replacing fill bits with hidden data. Experimental results obtained with a simple SWF (version 10) file are used to evaluate the effectiveness of the data hiding techniques and identify the artifacts that remain.

Keywords: Steganography, Small Web Format, Flash files

1. Introduction

Digital steganography is the art of inconspicuously hiding data within data [3]. Steganography is used to enforce copyrights through the creation of watermarks [3, 12] and to conduct covert communications [3, 7, 8]. Several programs are available for embedding data in a variety of file formats, including images [3, 7], audio [3, 7] and video [9].

The Small Web Format (SWF) or Flash is an interactive multimedia file format used in entertainment, education, business and communication applications. The widespread use of SWF files make them an excellent candidate for steganography – a 2008 survey [11] reported that more than 25% of all websites contained SWF files. Zaharis, *et al.* [8] have demonstrated how hidden information is easily spread on a social

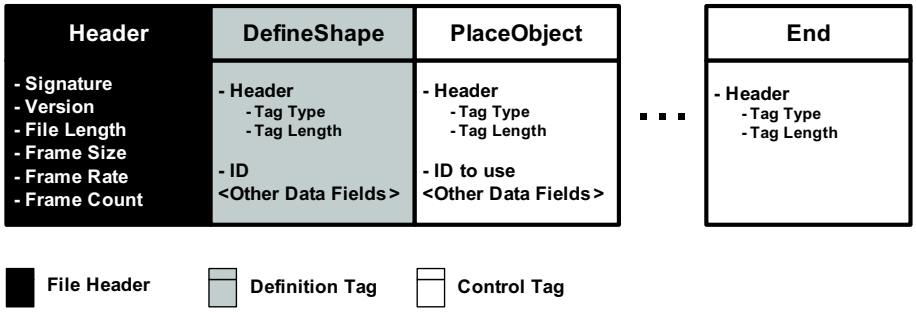

Figure 1. Example SWF file content.

network site using SWF files. This underscores the importance of developing techniques for detecting SWF steganography.

The detection of steganography involves finding artifacts left behind by embedding programs and detecting inconsistencies between the actual file content and typical file content [4]. By analyzing SWF steganography methods, a forensic investigator is better able to identify the inconsistencies and artifacts in an SWF file that contains hidden data.

This paper analyzes the SWF specification [1] and describes four embedding techniques: appending data to an SWF file, adding an extra Metadata tag, creating a custom Definition tag and replacing fill bits with hidden data. Tests of the four techniques, using a simple SWF (version 10) file that was run on a Flash player under Firefox, demonstrate that the hidden data did not influence playback. The tests also show that the techniques are effective in that the hidden data can be retrieved.

2. Small Web Format

Figure 1 shows sample content in an SWF file. The first component of the file is the Header, which contains information about the file. The Header includes a file signature, which is either FWS or CWS. FWS indicates that the file is uncompressed while CWS indicates that the portion of the file following the Header is compressed using the ZLIB algorithm. Following the signature is the file version and the uncompressed file length in bytes. The remainder of the Header contains the frame size, frame rate and frame count.

The Header is followed by a series of tags. Each tag has its own header, which contains the tag type and tag length in bytes. There are two types of tags, Definition tags and Control tags. Definition tags define objects, also known as characters, which include images, sounds

and videos used by an SWF player. Control tags instruct an SWF player what to do with the objects. An example is the PlaceObject tag, which displays the specified object on the screen. Each object has a unique character ID, which is stored in the Dictionary. Control tags that use an object access the object via its character ID in the Dictionary. The last tag is the End tag, which marks the end of the file.

3. Data Hiding in SWF Files

This section describes techniques for hiding data in SWF files. Existing techniques are identified along with additional techniques developed through an analysis of the SWF specification [1].

3.1 Existing Techniques

Tadiparthi and Sueyoshi [10] have described a method for hiding data in the frames of an animation. While their technique focuses on GIF animations, it is also applicable to animations involving SWF files. Zaharis, *et al.* [8] describe two methods for hiding data in SWF files. The first adds data to unused key frames of an SWF file. The second adds an MP3 file containing hidden data to an SWF file.

An FLV file encodes audio and video the same way as an SWF file [1, 2]. For this reason, the technique developed by Mozo, *et al.* [9], which appends data at the end of the Metadata, Audio or Video tags, will work on SWF files. To account for the larger tag and file size, the method adjusts the tag length in the header of the tag as well as the file length in the header of the file. Zhang and Zhang [12] have used User-Defined tags to add hidden data to an SWF file. User-Defined tags are tags with tag types that are not listed in the SWF specification. When an SWF player encounters such a tag type, it skips the tag and processes the next tag.

3.2 Additional Techniques

Analyzing the SWF specification [1] for data hiding opportunities is a systematic approach for developing steganographic techniques. In particular, statements in the SWF specification are examined and techniques are crafted that conform to or contradict the statements.

For example, the SWF specification states that User-Defined tags may be created, but that they are ignored by a Flash player. Thus, creating a User-Defined tag to hide data is in conformance with the SWF specification. On the other hand, if the specification stated that an SWF file should not contain User-Defined tags, then creating a User-Defined tag to hide data contradicts the specification. When a data hiding technique

contradicts the specification, it is important to test if it corrupts the file. Whether a data hiding technique conforms to or contradicts the specification, it will leave an artifact or trace evidence. Note that it is not possible to determine all the possible data hiding techniques purely by examining the SWF specification because all the information pertaining to SWF files is not necessarily stated in the specification. However, the SWF specification is a good starting point for systematically identifying data hiding opportunities.

The first proposed data hiding technique relies on the observation that SWF uses the End tag to signify the end of a file. The hypothesis is that a Flash player will ignore everything after the End tag. Thus, the hiding technique simply appends data at the end of the file. This differs from the technique of Mozo, *et al.* [9] in that the appended data is not a part of a tag. It is also productive to consider a variation of this technique where a compressed SWF file is uncompressed, data is added to the end of the file, and the modified file is then compressed.

The second proposed data hiding technique uses the Metadata tag. The SWF specification states that a Flash player ignores the Metadata tag and that an SWF file should have no more than one such tag. The second technique thus adds an additional custom Metadata tag with hidden data to an SWF file.

The third proposed data hiding technique uses Definition tags. Each of these tags has a character ID that is referenced by a Control tag or by another Definition tag. The SWF specification states that each Definition tag must have a unique character ID. The third technique thus inserts a custom Definition tag with an existing character ID or an unused character ID.

The fourth proposed data hiding technique uses fill bits in an SWF file. When a data item does not end at an eight-bit boundary, the compiler fills the remaining bits with zeros. This ensures that two tags do not share the same byte and that references are in bytes instead of bits. The fourth data hiding technique thus replaces the fill bits in an SWF file with hidden data.

4. Data Hiding Experiments

Experiments involving the data hiding techniques used a simple SWF (version 10) file. The file contained a button with text, Actionscript 3.0 code, an animation of a bouncing ball and sound that played with the animation. The animation and sound played when the button was pressed. Figure 2 shows the display before and after the button was pressed. Note that the arrow in the figure indicates the animation flow.

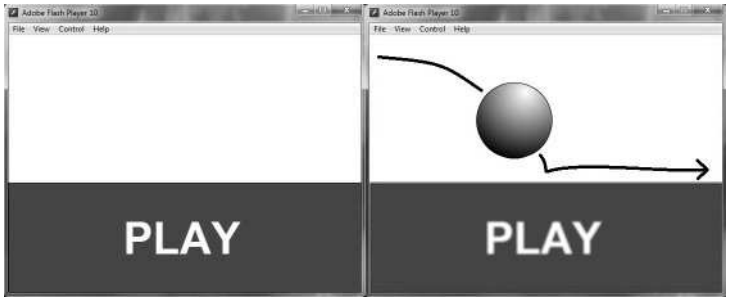

Figure 2. SWF file before and after the button is pressed.

Table 1. Hidden data test files.

File Name	File Type	File Size (Bytes)
`data.txt`	Plain Text	6
`test.swf`	Shockwave Flash/Simple Web Format	7,326
`music.mp3`	MPEG-1 Audio Layer 3	3,502,112

Each data hiding technique was evaluated using a separate copy of the test SWF file. A Java program was executed to hide and to extract three data files: a plain text file (`data.txt`), a copy of the SWF file itself (`test.swf`) and an MP3 file (`music.mp3`). Table 1 lists the three files along with their sizes.

The procedure for hiding and extracting each file involved the following steps:

- Create a copy of the simple SWF file.
- Insert data into the SWF file using the data hiding technique.
- Test whether or not the SWF file plays on a Flash player (version 10.1 r53) add-on for Firefox (version 3.6.6).
- Extract the hidden data.

A data hiding technique is deemed to be successful if: (i) the SWF file with the hidden data can be played on a Flash player with no noticeable differences compared with the original SWF file; and (ii) the hidden data can be extracted from the SWF file.

5. Experimental Results

This section discusses the results obtained for the four data hiding techniques.

Figure 3. Appending data to an SWF file.

5.1 Appending Data

The first experiment tested two variations of the hiding technique that adds data at the end of an SWF file.

The first technique simply appends data to an SWF file without modifying the file (Figure 3).

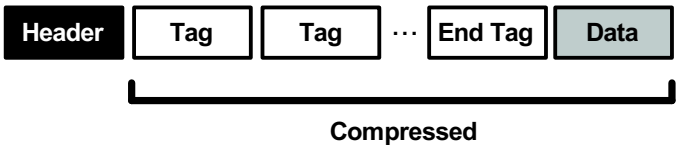

Figure 4. Uncompressing, appending data and compressing an SWF file.

The second technique uncompresses the tags of an SWF file, appends data after the End tag and recompresses the tags with the appended data (Figure 4).

Extracting the hidden data from an SWF file involves the following steps:

- Make a copy of the SWF file.
- Uncompress all the data following the Header.
- Locate the End tag.
- If the hidden data was appended to the compressed SWF file:
  - Compress the data between the Header and the end of the End tag.
  - Record the size of the compressed data plus the size of the Header.
  - Extract the hidden data, which is located after the recorded length.
- If the hidden data was appended after uncompressing the SWF file, extract the hidden data, which is located after the End tag in the uncompressed file.

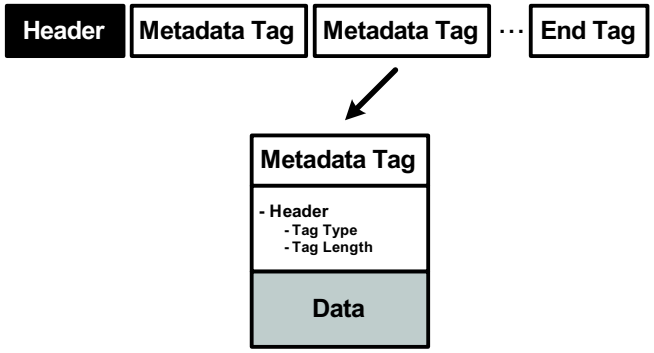

Figure 5. Adding an extra Metadata tag to an SWF file.

The first technique that simply adds data at the end of an SWF file was successful for all three test files. However, the second technique, which uncompresses the file, adds data and then compresses the file, only worked for the plain text file. The SWF file did not run on the Flash player when the audio and SWF files were appended. File `test.swf` worked when the Header was altered to include the size of the hidden data, but `music.mp3` did not.

To detect if an SWF file has appended data, a forensic investigator must search the SWF file to see if data appears after the End tag of the file. Note that attempting to detect appended data by comparing the physical SWF file length to the length recorded in the Header of the file may not work because the file length in the Header can be changed quite easily.

5.2 Adding an Extra Metadata Tag

As described earlier, this data hiding technique exploits the SWF specification that states that an SWF file should have at most one Metadata tag. The data hiding technique thus adds an extra Metadata tag after the first Metadata tag in the SWF file. Unless the extra Metadata tag is explicitly searched for, it will not be detected.

The experimental test of this technique used a Metadata tag of type 77 whose length was set to the length of the tag plus the length of the hidden data. Figure 5 shows an example of adding an extra Metadata tag to an SWF file. To extract the data, it is necessary to search the SWF file for the second Metadata tag of type 77. All the data in the tag after the tag length corresponds to the hidden data.

This data hiding technique was successful for all three test files. The Flash player ignored the second Metadata tag in the SWF file. The data

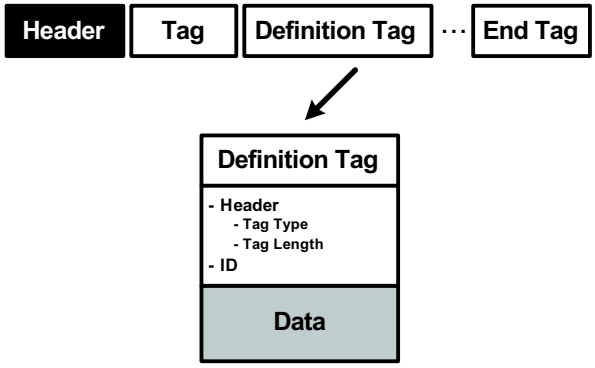

Figure 6. Adding a custom Definition tag to an SWF file.

hiding technique can be detected by counting the number of Metadata tags (i.e., there are multiple tags). Note, however, that an SWF file with only one Metadata tag can also contain hidden data in the tag.

5.3 Adding a Custom Definition Tag

This data hiding technique creates and inserts a custom Definition tag with hidden data into an SWF file. The technique is similar to adding a custom Metadata tag, except that the tag has an extra character ID field.

The experiments employed a DefineShape tag of type 2. Figure 6 shows an example of adding a custom Definition tag to an SWF file.

Three variations of the data hiding technique were evaluated. The first variation used a unique character ID. Extraction of the hidden data requires a search for the character ID. This variation was successful for all three test files. The Flash player loaded the test files with the hidden data; however, since the custom Definition tag with the hidden data was not used by another tag, the player did not reach a state where it failed.

The second variation used a non-unique character ID in a custom Definition tag placed before the correct Definition tag. The resulting SWF file contained two Definition tags with the same character ID. The hypothesis was that the Flash player would overwrite the data in memory that came from the custom Definition tag when the data from the second (correct) Definition tag was read. However, the player ignored the second correct Definition tag and used the data in the first custom Definition tag; thus, the graphic was not displayed properly.

The third variation is similar to the second, except that it inserts the custom Definition tag after the correct Definition tag. This variation

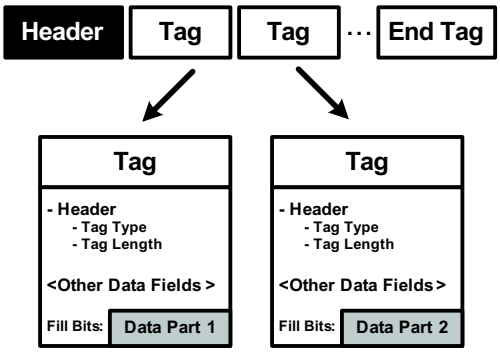

Figure 7. Replacing fill bits in an SWF file.

was successful because the Flash player appeared to ignore a Definition tag for which it already had a character ID.

The first variation of the hiding technique can be detected by checking all the Definition tags for unused character IDs. When an SWF file is created, any unused data is excluded from the file. A Definition tag with an unused character ID is thus an irregular occurrence and would likely indicate that the first variation of the data hiding technique has been used. The other two variations can be detected by searching all the tags for repeated character IDs.

5.4 Replacing Fill Bits

The SWF specification lists all the locations where fill bits are required. Figure 7 shows how the data hiding technique stores data in the fill bit locations.

In the experiments, fill bits were used in the following locations:

- Fields in tags with the RECT type (e.g., ShapeBounds field in the DefineShape tag).
- Fields in tags with the CXFORM type (e.g., ColorTransform field in the PlaceObject tag).
- Fields in tags with the MATRIX type (e.g., Matrix field in the PlaceObject tag).
- Fields in tags with the SHAPE type (e.g., Glyph-ShapeTable field in the DefineFont tag).
- Fields in tags with the TEXTRECORD type (e.g., TextRecords field in the DefineText tag).

Table 2. Results obtained for the data hiding techniques.

Technique	Variation	Result
Append data	Append data at end of file	Successful
	Uncompress, append data and compress file	Successful (small files)
Add Metadata tag		Successful
Add Definition tag	Use unique character ID	Successful
	Add tag with same character ID before original Definition tag	Not successful
	Add tag with same character ID after original Definition tag	Successful
Replace fill bits		Successful (very small files)

Extracting the data requires the fill bits of the file to be concatenated in the order they were written.

This technique was successful for the three test files. However, unlike the other data hiding techniques, there is a limit to the amount of data that can be stored. The SWF file used to hide data had 7,326 bytes, but only 167 fill bits. Thus, the plain text file was the only file that fit in the SWF file; the other test files were too large.

Detecting the use of this data hiding technique is not as simple as checking for non-zero fill bits. An unmodified SWF file only has zeros for fill bits. However, if a custom Definition tag was used to hide data, then the hidden data could have overwritten the fill bits of the Definition tag. Therefore, it could be the case that only the tags with non-zero fill bits contain hidden data, instead of all the fill bits.

6. Conclusions

SWF files are excellent candidates for hiding data. The four new data hiding techniques presented involved appending data to an SWF file, adding an extra Metadata tag, adding a custom Definition tag, and replacing fill bits. Experiments indicate that the data hiding techniques were generally successful with the test files (Table 2). Nevertheless, all four data hiding techniques leave artifacts in SWF files that facilitate detection. Information about the artifacts can also be used by sanitizing tools for browsers and websites to remove potentially malicious hidden data before displaying the associated SWF files.

Additional research is necessary to identify new data hiding techniques and their artifacts. It is also important to investigate the extent to which SWF steganography is being used on the Internet.

References

[1] Adobe Systems, SWF File Format Specification (Version 10), San Jose, California (www.adobe.com/devnet/swf), 2008.

[2] Adobe Systems, Adobe Flash Video File Format Specification (Version 10.1), San Jose, California (www.adobe.com/devnet/f4v), 2010.

[3] D. Artz, Digital steganography: Hiding data within data, *IEEE Internet Computing*, vol. 5(3), pp. 75–80, 2001.

[4] F. Cohen, *Digital Forensic Evidence Examination*, Fred Cohen and Associates, Livermore, California, 2010.

[5] E. Dallaway, Steganography is key ingredient to anti-forensics, *Infosecurity Magazine*, vol. 5(8), p. 11, 2008.

[6] J. Davis, J. MacLean and D. Dampier, Methods of information hiding and detection in file systems, *Proceedings of the Fifth International Workshop on Systematic Approaches to Digital Forensic Engineering*, pp. 66–69, 2010.

[7] G. Kessler, An overview of steganography for the computer forensics examiner, *Forensic Science Communications*, vol. 6(3), 2004.

[8] A. Martini, A. Zaharis and C. Ilioudis, Data hiding in the SWF format and spreading through social network services, *Proceedings of the Fourth International Workshop on Digital Forensics and Incident Analysis*, pp. 105–115, 2009.

[9] A. Mozo, M. Obien, C. Rigor, D. Rayel, K. Chua and G. Tangonan, Video steganography using flash video, *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pp. 822–827, 2009.

[10] G. Tadiparthi and T. Sueyoshi, A novel steganographic algorithm using animations as cover, *Decision Support Systems*, vol. 45(4), pp. 937–948, 2008.

[11] B. Wilson, MAMA: Key findings (dev.opera.com/articles/view/mama-key-findings), 2008.

[12] X. Zhang and X. Zhang, Information hiding algorithm based on flash animation, *Computer Engineering*, vol. 36(1), pp. 181–183, 2010.