



# *Inspection and Sanitization Guidance for the Graphics Interchange Format (GIF)*

Version 1.0  
3 February 2015



**National Security Agency  
Information Assurance Capabilities  
9800 Savage Rd, Suite 6699  
Ft. George G. Meade. MD 20755**

**Authored/Released by:  
Unified Cross Domain Capabilities Office  
[cds\\_tech@nsa.gov](mailto:cds_tech@nsa.gov)**



**DOCUMENT REVISION HISTORY**

Date	Version	Description
3 Feb 2015	1.0	Final
12/13/2017	1.0	Updated Contact information, IAC Logo, Cited Trademarks and Copyrights, Expanded Acronyms, and added Legal Disclaimer

**Disclaimer**

The information and opinions contained in this document are provided “as is” and without any warranties or guarantees. References herein to any specific commercial product, process, or service by trade name, trademark, manufacturer or otherwise, does not constitute or imply its endorsement, recommendation or favoring by the United States Government and this guidance shall not be used for advertising or product endorsement purposes

## EXECUTIVE SUMMARY

The Graphics Interchange Format (GIF) Inspection and Sanitization Guidance (ISG) document provides guidelines and specifications for developing an inspection and sanitization software filter for use with the GIF file format as defined by CompuServe Inc. GIF was developed by CompuServe in 1987 as a replacement for an older Run-Length Encoding (RLE) image format that only supported black and white images. The first version of GIF was developed as version 87a, which supported color images and higher compression ratios to allow for faster downloads.

The GIF 89a format is an extension of the original GIF 87a format and has the capability to use transparency and animated frames. The GIF file type utilizes 8-bit color with a maximum of 255 colors per frame. The GIF file format supports lossless compression using the Lempel-Ziv-Welch (LZW) compression method.

This guidance document examines the GIF specifications for data attack, data hiding, and data disclosure risks that exist within the file structure. It provides a breakdown of each component of a GIF file and provides recommendations that can help assure that the GIF file is not only compliant with the specifications but also mitigates these various risks.

TABLE OF CONTENTS

1. Scope ..... 1-1

    1.1. Purpose ..... 1-1

    1.2. Introduction ..... 1-1

    1.3. Background ..... 1-1

    1.4. Document Organization..... 1-1

    1.5. Actions ..... 1-2

    1.6. Document Limitations ..... 1-3

        1.6.1. Covert Channel Analysis ..... 1-3

2. Constructs And Taxonomy..... 2-1

    2.1. Constructs..... 2-1

    2.2. Taxonomy ..... 2-1

3. Overview Of Graphics Interchange Format..... 3-1

4. GIF Constructs..... 4-1

    4.1. GIF Header ..... 4-1

    4.2. Logical Screen Descriptor (LSD) ..... 4-2

    4.3. Global Color Table (GCT) & Local Color Table (LCT)..... 4-3

    4.4. Application Extension ..... 4-5

    4.5. Graphic Control Extension (GCE)..... 4-7

    4.6. Image Descriptor ..... 4-10

    4.7. Plain Text Extension (PTE)..... 4-12

    4.8. Image Data ..... 4-13

    4.9. Comment Extension..... 4-14

    4.10. Trailer ..... 4-15

5. Summary of Risks ..... 5-1

6. Acronyms ..... 6-1

7. References ..... 7-1

**LIST OF FIGURES**

Figure 3-1. GIF File Structure [3]..... 3-1

Figure 3-2. GIF Sub-block structure ..... 3-2

Figure 4-1. Valid GIF Headers ..... 4-1

Figure 4-2. Logical Screen Descriptor Location..... 4-2

Figure 4-3. Global Color Table ..... 4-3

Figure 4-4. GIF with Original GCT..... 4-4

Figure 4-5. GIF with Hidden Data in GCT ..... 4-4

Figure 4-6. Application Extension ..... 4-5

Figure 4-7. Application Extension in GIF ..... 4-6

Figure 4-8. Graphic Control Extension ..... 4-8

Figure 4-9. QR hidden through use of transparency ..... 4-8

Figure 4-10. Data Hidden in GCE by extending the sub-block..... 4-8

Figure 4-11. Frame Stacking with Short Delay Time ..... 4-9

Figure 4-12. Image Descriptor..... 4-10

Figure 4-13. Image Descriptor with Truncated Frame ..... 4-11

Figure 4-14. Plain Text Extension ..... 4-12

Figure 4-15. Image Data Table ..... 4-13

Figure 4-16. Comment Extension with Hidden Message..... 4-14

Figure 4-17. Valid Trailer ..... 4-15

Figure 4-18. Data before Trailer ..... 4-16

Figure 4-19. Data after Trailer ..... 4-16

Figure 4-20. PHP after Trailer..... 4-16

Figure 4-21. JAR after Trailer (GIFAR.gif)..... 4-17

## LIST OF TABLES

Table 1-1 - Document Organization .....	1-2
Table 1-2 - Recommendation Actions.....	1-3
Table 2-1 - Document Taxonomy .....	2-1
Table 3-1 - GIF Block Descriptions .....	3-2
Table 4-1 - Disposal Method Options.....	4-7
Table 5-1 - Summary of Risks.....	5-1

# **1. SCOPE**

## **1.1. Purpose**

The purpose of this document is to provide guidance for the development of a sanitization and analysis software tool for Graphics Interchange Format (GIF) files. This document analyzes the various constructs contained within the GIF specification and then discusses data hiding, data attack, and data disclosure risks. It describes how these constructs can then be a cause for concern when considering the hiding of sensitive data or attempts to exploit a system. This document provides numerous recommendations and mitigations that can be used to ensure the use of a GIF image is safer and more accurately conforms to the specification. The intended audience of this document includes system engineers, designers, software developers, and testers who work on file inspection and sanitization applications that involve processing GIF files.

## **1.2. Introduction**

GIF files were first introduced in 1987 (GIF87a, [1]) by CompuServe. They were one of the first image formats to use color. The GIF file format utilizes Lempel-Ziv-Welch (LZW) compression. In 1989, a new version (GIF89a, [2]) of the GIF file format was released that added support for many features such as transparency and animation delays [2]. Both versions of the GIF specification can be found on the website of the World Wide Web Consortium (W3C) [1, 2].

## **1.3. Background**

The GIF file format was created by CompuServe and has been in use since 1987. Over the last 20 years, it has been primary used to display animated images on the World Wide Web. The compression algorithm that GIF uses, LZW was originally patented in 1985, before the creation of the GIF file. The Portable Network Graphics (PNG) file format was designed to replace the GIF file format to avoid the patent issues revolving around GIF's LZW compression. [5]. Despite the patent issues, which have now expired, GIF files are commonly found on the World Wide Web. GIF files are frequently used today for company logos, low-color images, and small animations.

## **1.4. Document Organization**

This section summarizes the organization of this document.



Table 1-1 – Document Organization

Section	Description
<b>Section 1:</b> Scope	This section describes the purpose, introduction, background, organization, and recommendations related to this document.
<b>Section 2:</b> Constructs and Taxonomy	This section describes the constructs and taxonomy that are used throughout this document.
<b>Section 3:</b> Overview	This section introduces the basics of the GIF file structure.
<b>Section 4:</b> GIF Constructs	This section contains the GIF constructs that have risks and the options for mitigation.
<b>Section 5:</b> Acronyms	This section lists the acronyms in this document.
<b>Section 6:</b> Referenced Documents	This section lists the sources that were used to prepare this document.


1.5. Actions

Each construct description lists recommended actions for handling the construct when processing a message. Generally, inspection and sanitization programs will perform one of these actions on a construct: *Validate*, *Remove*, *Replace*, *External Filtering Required*, *Review*, or *Reject*.

The recommendation section in each construct lists each action that is applicable along with an explanation that is specific to the construct. Not all actions are applicable or appropriate for every context. As such, implementers are not expected to implement all the actions for a given risk; instead, they are expected to determine which action – or perhaps actions – applies best to their context. Definition of the criteria used to determine which action is “best” and of the specific method used to execute the action is left to the implementer.

Recommendations such as remove and replace may alter the integrity of the GIF file. It is important to address these issues in order to retain functionality.

NOTE



The recommendations in this document are brief explanations rather than a How-To Guide. Readers should refer to the construct description or official documentation for additional details.

The following table summarizes the recommendation actions:

Table 1-2 – Recommendation Actions

Recommendation Action	Comments
Validate	Verify the data structure's integrity, which may include integrity checks on other components in the message. (This should almost always be a recommended action.)
Replace	Replace the data structure or one or more of its elements with values that alleviate the risk (e.g., replacing a username with a non-identifying, harmless value or substituting a common name for all authors).
Remove	Remove the data structure or one or more of its elements and any other affected parts.
External Filtering Required	Note the data type and pass the data to an external action for handling that data type (e.g., extract text and pass it to a dirty word search).
Review	Present the data structure or its constructs for a human to review. (This should almost always be recommended if the object being inspected can be revised by a human.)
Reject	Reject the file.

NOTE



No recommendations for logging all actions and found data are included here because all activity logging in an inspection application should occur “at an appropriate level” and presented in a form that a human can analyze further (e.g., the audit information may be stored in any format but must be parsable and provide enough information to address the issue when presented to a human.)

1.6. Document Limitations

This document covers the file format GIF 89a file format. The GIF 89a is an extension of the GIF 87a format. This Document does not cover all variants of the GIF 89a file format.

1.6.1. Covert Channel Analysis

It is impossible to identify all available covert channels, whether in a file format or a communication protocol. Because they contain free-form text, searching for hidden data becomes increasingly difficult. No tool can possibly analyze every channel, so this document highlights the highest risk areas to reduce or eliminate data spills and malicious content.

Additionally, this document does not discuss steganography within block text or media files, such as a hidden message that is embedded within an innocuous image or paragraph. Separate file format filters that specialize in steganography should be used to handle embedded content, such as text, images, videos, and audio.

## 2. CONSTRUCTS AND TAXONOMY

### 2.1. Constructs

This document describes many of the constructs used in GIF, but it does not describe every construct, thus this document is not to be treated as a complete reference. Developers of a GIF filter should consult the official specifications [1, 2] alongside this documentation for the full context. For each construct that is mentioned, the following sections exist:

- **Overview:** An explanation of the construct with examples.
- **Risks and Recommendations:** An explanation of potential risks posed by the construct with corresponding mitigation strategies.
- **Product:** The specifications in which the construct is found.
- **Location:** A textual description of where to find the construct.

### 2.2. Taxonomy

The following table describes the terms that appear in this document:

Table 2-1 – Document Taxonomy

Term	Definition
Construct	An object that represents some form of information or data in the hierarchy of a GIF image.
Inspection and Sanitization	Activities for processing files and protocols to prevent inadvertent data leakage, data exfiltration, and malicious data or code transmission
ISG	A document (such as this) that details a file format or protocol and inspection and sanitization activities for constructs within it.
Recommendations	A series of actions for handling a construct when performing inspection and sanitization activities.

### 3. OVERVIEW OF GRAPHICS INTERCHANGE FORMAT

GIF files are composed of blocks and sub-blocks. The GIF specification defines blocks that are both mandatory and optional. Some blocks can implement a list of sub-blocks that are appended to the block. Sub-blocks are a part of the block and provide a list of variable length data that supplement the original block. Some blocks may only appear once in the file, while others may appear multiple times. There are blocks that begin with a block identifier (a single or multi-byte value) that introduces the beginning of the block. However, some blocks may not implement an identifier and must immediately follow an existing block that does implement an identifier. The following image shows each required and optional block by their names from the GIF specification.

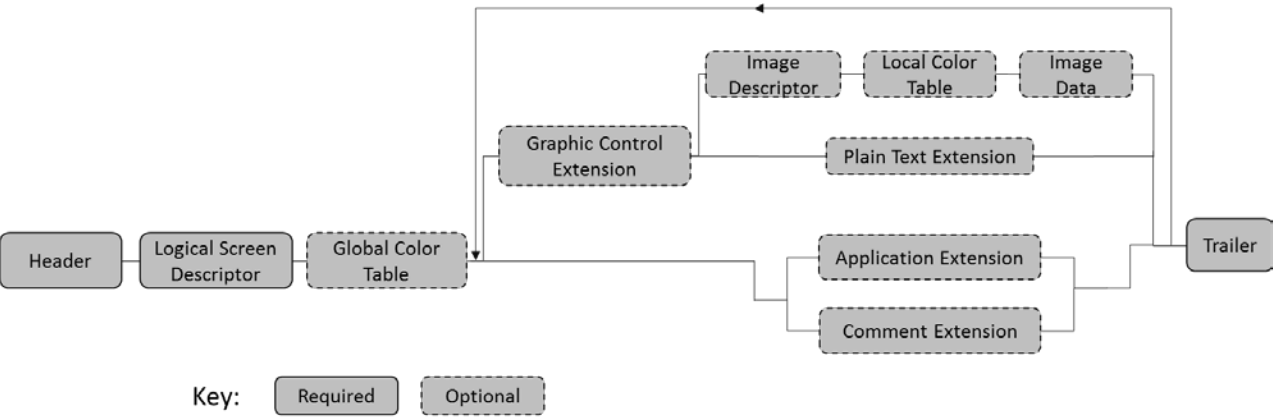
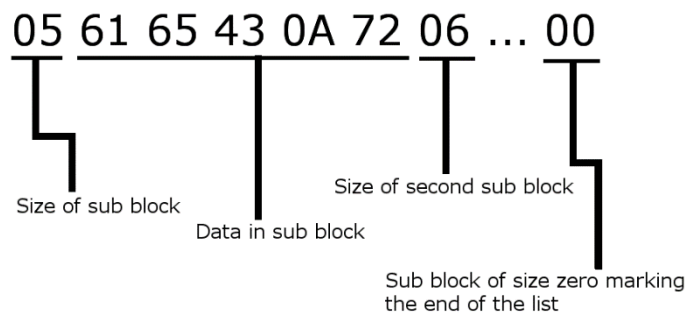


Figure 3-1. GIF File Structure [3]

There are three different groups of blocks defined in the specification: Control, Graphic-Rendering, and Special Purpose blocks. Control blocks contain information to “control the process of the Data stream or information used in setting hardware parameters” [2]. Graphic-Rendering blocks “contain information and data used to render a graphic on the display device” [2]. Special Purpose blocks are “neither used to control the process of the Data Stream nor do they contain information or data used to render a graphic” [2].

There are blocks that are fixed length in size, while others are variable in length. A variable length block will implement a number of sub-blocks that follow the block. The length and number of sub-blocks is unknown until all the sub-blocks are parsed. Sub-blocks are implemented in a linked-list manner as shown in Figure 3-2. Following the block definition is a list of sub-blocks. The first byte in the sub-block list will be the length of the first sub-block. After that number of bytes (data of the first sub-block), the next byte will be the length of the next sub-block. This process will continue until the

next length of the next sub-block is zero, which will terminate the list of sub-blocks. Following the last sub-block, a new block must begin.



**Figure 3-2. GIF Sub-block structure**

Each block has its own structure and criteria defined in the specification. Table 3-1 is derived from the GIF specification and shows the requirements and identifier (if one exists) for each block. When a block does not have an identifier, it can be located following another block that does implement a block identifier. The table below provides a Location field that describes the where or how (if using an identifier) a block can be found.

**Table 3-1 - GIF Block Descriptions**

Block Name	Required/Optional (Occurrence)	Block Identifier	Location	Block Type	Contain Sub-blocks
Header	Required (1)	GIF89a (or GIF87a)	First 6 bytes of the file	Control	N
Application Extension	Optional (*)	0x21FF	Located by identifier	Special Purpose	Y
Comment Extension	Optional (*)	0x21FE	Located by identifier	Special Purpose	Y
Logical Screen Descriptor	Required (1)	N/A	Immediately follows Header	Control	N
Global Color Table	Optional (1)	N/A	If present, immediately follows Logical Screen Descriptor	Graphic-Rendering	N
Image Descriptor	Optional (*)	0x2C	Located by identifier	Graphic-Rendering	N
Image Data	Optional (*)	N/A	If present, immediately follows Image Descriptor or Local Color Table (if present)	Graphic-Rendering	Y

Local Color Table	Optional (*)	N/A	If present, immediately follows Image Descriptor	Graphic-Rendering	N
Plain Text Extension	Optional (*)	0x2101	Located by identifier	Graphic-Rendering	Y
Graphic Control Extension	Optional (*)	0x21F9	Located by identifier	Control	Y (1 sub-block only)
Trailer	Required (1)	0x3B	Last byte of the file	Control	N

From the previous table there are a number of blocks that begin with the byte value 0x21. The value 0x21 is known as the Extension Introducer. This means that the block that follows is an extension (a type of block). All extensions from the table have a second byte that is part of the identifier. This allows the parser to determine the type of extension block that follows.

The Image Descriptor is unique as it begins with the single byte value 0x2C. This byte is known as the Image Separator which is only used to identify the beginning of the Image Descriptor block. Another special byte value, defined by the GIF specification, is the Block Terminator. It is equal to the byte value 0x00. The Block Terminator is used to terminate the linked list of sub-blocks. Since a value of 0x00 means that there are no sub-blocks that follow, it terminates the list of sub-blocks and the block.

## 4. GIF CONSTRUCTS

This section discusses specific features and risks of GIF files and provides recommendations for each section.

### 4.1. GIF Header

#### OVERVIEW:

The header contains the type and version of the file. The header is the first 6 bytes of the file and defines the version of the GIF file. The header should be one of two versions GIF89a or GIF87a. This is also known as the “magic number” for this file format.

Examples of proper headers are shown below in Figure 4-1. These must be the first 6 bytes of the file. The following examples illustrate valid GIF headers along with their representative hexadecimal values.

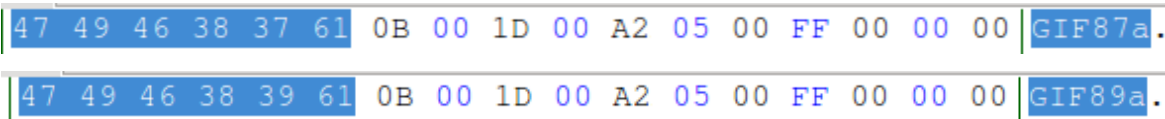


Figure 4-1. Valid GIF Headers

#### RISKS AND RECOMMENDATIONS

**Data Hiding and Data Attack:** Risks involving the header section of the GIF format are files masquerading as GIFs or GIFs masquerading as other file formats.

1. **Validate:** Check that the first 6 bytes are either GIF87a or GIF89a.
2. **Reject:** Reject files that do not contain a valid GIF header.

#### PRODUCT

GIF87a and GIF89a

#### LOCATION

The header is located in the first 6 bytes of the file.

## 4.2. Logical Screen Descriptor (LSD)

### OVERVIEW:

Following the header is the LSD of the GIF file format. The LSD is 7 bytes long and is illustrated in Figure 4-2. There is no special marker indicating the start of the LSD, it exists immediately after the GIF header and is a constant 7 bytes in length. The first 4 bytes of the LSD, which are shown in green below, are the size of the logical screen (LS) or area where frames are organized. The first two bytes indicate the LS width and the second two bytes indicate the LS height, both defined in the number of pixels. The next byte of data (the 5<sup>th</sup> byte) is a Packed Field of the global color table flag (1 bit), color resolution (3 bits), sort flag (1 bit), and the size of the global color table (3 bits). The size of the global color table is calculated by  $3 \cdot (2^{(N+1)})$ , where  $N$  is the number represented by the last 3 bits in the packed field. The result from this calculation is the number of bytes in the color table. The Global Color Table (GCT) flag (most significant bit of packed field) indicates if the GCT exists. When zero, the GCT does not exist. When set to one, the GCT does exist. The next byte after the Packed Field contains the Background Color Index, which is a value selected from the GCT. This field is always present but is only meaningful when the GCT exists. The last byte of the LSD is used to compare the Aspect Ratio. The Aspect Ratio field is sometimes referred to as the pixel aspect ratio field, which is defined to be the quotient of the pixel's width over its height [2]. It is almost always 0x00, which means that no ratio information is given. The actual aspect ratio of the GIF image is calculated by  $(N+15)/64$ , where  $N$  is the single-byte value defined in the aspect ratio field in the LSD. A diagram of the LSD is in Figure 4-2.

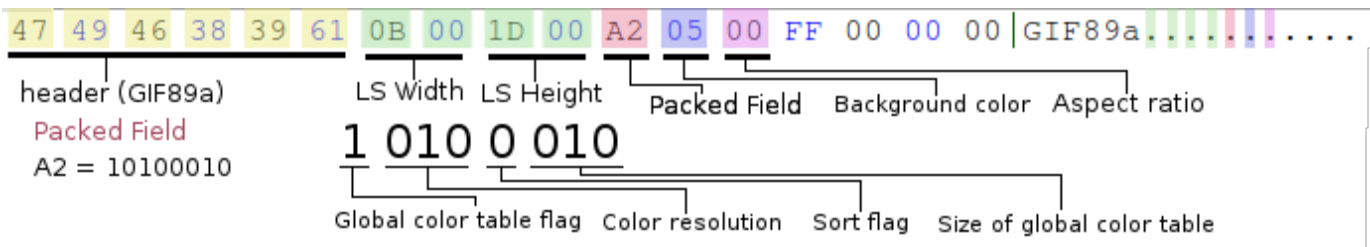


Figure 4-2. Logical Screen Descriptor Location

### RISKS AND RECOMMENDATIONS

**Data Attack:** Setting the height and width fields to very large values will cause some applications to crash and in some cases result in the user's session crashing.

1. **Validate:** Check that the values in LS width and LS height are valid according to the rest of the image.



- 2. **Replace:** Replace the LS width and LS height with their correct values according to the rest of the image, if possible.
- 3. **Reject:** Reject files with either an invalid LS width or invalid LS height.

**Data Attack:** The Aspect Ratio field is used in a computation when its value is nonzero and it is possible to have an impact on image rendering if the value is incorrect. Typically the value is 0x00, which means that no aspect ratio information is given.

- 4. **Validate:** Check that the aspect ratio field is 0x00.
- 5. **Replace:** Replace the aspect ratio field with 0x00, this may have an adverse effect on the image.

**Data Attack:** Setting the size of the Global Color Table too high will cause problems with some applications that try to parse it, but in others, it will cause the file to not load and possibly return the notification: “file truncated”.

- 6. **Validate:** Check that the size of the global color table is valid.

**PRODUCT**

GIF87a and GIF89a

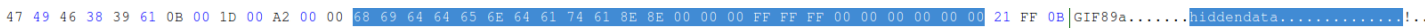
**LOCATION**

The LSD starts after the first 6 bytes of data and is 7 bytes long.

**4.3. Global Color Table (GCT) & Local Color Table (LCT)**

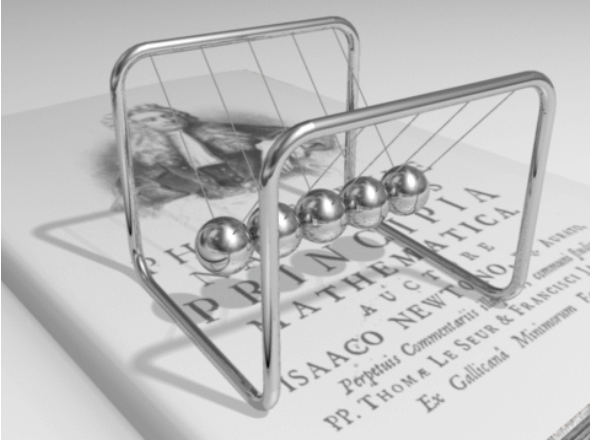
**OVERVIEW:**

The GCT contains an index and information about the color represented in the image. These colors are represented by 3 bytes, with a single byte representing Red, Green, and Blue (RGB). This section of the file is not necessary and can be replaced by each individual frame’s Local Color Table (LCT). Use of the GCT is indicated when the GCT flag is set in the LSD. An example GCT that has been overwritten with hidden data is shown in Figure 4-3 below. While the colors of the image may appear different, the structure of the image should not.

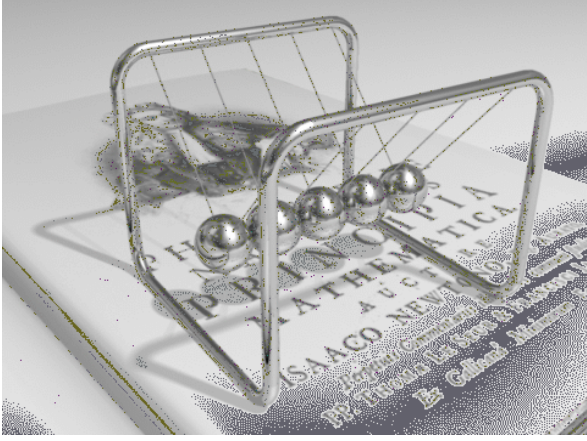


**Figure 4-3. Global Color Table**

The example above manipulates the colors in the GCT, which has an effect on the image, illustrated in the two figures below. Figure 4-5 on the right is from the example in Figure 4-3. Figure 4-4 on the left is the original image with no hidden data. The image on the right is distorted because the color table fields are textual characters, but interpreted as color values. Manipulating color tables is a way to hide data in images without requiring to change each individual pixel value. Depending on the colors defined in the original color table, overwriting them with hidden data may go undetected.



**Figure 4-4. GIF with Original GCT**



**Figure 4-5. GIF with Hidden Data in GCT**

## RISKS AND RECOMMENDATIONS

**Data Hiding:** Manipulation of the LCT or the GCT would allow data to be hidden in the fields. Colors in the image would be distorted but it would not affect the shape of the image.

1. **Replace:** Replace each color table with a shuffled table with new index numbers and update the pixel values such that no color information is lost, if possible.
2. **Replace:** Combine and merge all local tables into a single global table (remove each local) if possible.

## PRODUCT

GIF87a and GIF89a

## LOCATION

GCT is located immediately after the LSD.

LCT is located immediately after the image descriptor if the LCT flag is set.

## 4.4. Application Extension

### OVERVIEW:

The Application Extension contains information for applications to add unique identifying information to GIFs. It is an optional extension in GIF and there can be zero or more occurrences in the file. The Application Extension block is identified by the Extension Introducer byte (0x21) and then by the Extension Label (0xFF). The first two bytes of the Application Extension are always 0x21FF. The intent of this block is for applications to control some application-defined aspects of the GIF (e.g., animation). The Application Extension block consists of the Extension Introducer, Extension Label, the size of the block, an application name field, and an application authentication code field. Following these fields is a list of data sub-blocks that may define application specific properties. A diagram of the Application Extension is shown below in Figure 4-6. In this example, the application name is “NETSCAPE” and the authentication code is set to “32 2E 30”. The specification defines this field as a “sequence of three bytes used to authenticate the Application Identifier.” In this example, the value of “32 2E 30” is equal to “2.0” in ASCII. After these fields, the list of data sub-blocks begin. The next byte is equal to 0x03 which means that the first data sub-block is 3 bytes long. The data sub-block is in a custom format. This format specifies a data sub-block index as the first byte which defines the data sub-block type. The next two bytes define the number of animation loops (0x0000). The next sub-block length is 0x00, which means that the sub-block list has been terminated. For reference, the next two bytes in Figure 4-6 are 0x21F9, this starts the Graphic Control Extension block.

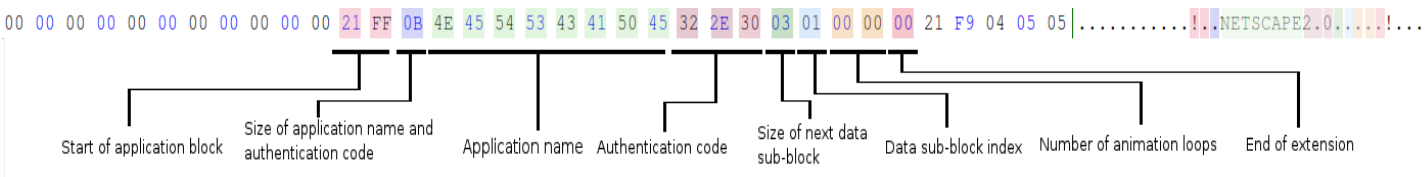


Figure 4-6. Application Extension

An example of Adobe’s<sup>1</sup> Extensible Metadata Platform (XMP) metadata inserted into a GIF file through an application extension is shown below in Figure 4-7.

<sup>1</sup> Adobe is a registered trademark of Adobe Systems, Inc..



Starts at 21 FF and ends with a 0x00.

4.5. Graphic Control Extension (GCE)

OVERVIEW:

The Graphic Control Extension (GCE) is an extension used to control the transparency and the animation functions of the GIF image. The GCE is identified by 0x21 F9. The next field is the byte size, which defines the size of the single GCE sub-block and should always be set to 0x04. This is considered to be one and only sub-block that the GCE block implements. The next field is the packed byte, which consists of 4 sub-fields encoded within the single byte. The first 3 bits of the packed byte are reserved for future use. The next 3 bits are used for defining the disposal method. The display method indicates the way the graphic is to be treated after it is displayed. There are four defined and four undefined options for the disposal method shown in the table below derived from the GIF specification [1].

Table 4-1 - Disposal Method Options

Disposal Method Value	Description
0	No disposal is required, decoder is not required to take any further action.
1	Do not dispose, the graphic is to be left in place.
2	The area used by the graphic must be restored to the background color defined by the LSD block.
3	Restore the area overwritten by the graphic to the content that was previously there.
4-7	Undefined.

The next bit in the packed byte following the disposal method is the user input flag. This flag indicates if user input is required before continuing (i.e., return, mouse click, etc.). The final bit, the least significant bit, of the packed byte is the transparent color flag which states if this GIF has a transparent color. If set, the transparency index is given in the Transparent Color Index field (defined later in the GCE). After the packed byte is the 2-byte delay time field, which is the number of milliseconds to wait between frames in an unsigned integer format. According to the specification, the Transparent Color Index is a field that will provide a valid index if the Transparent Color Flag is set to 1. The Transparent Color Index field contains the index of the color that will be displayed as transparent in the GIF. The last byte of the GCE is the block terminator and will always be 0x00. The GCE is used to define frames and plain text extensions. There can be any number of GCEs in a GIF file. After the GCE there must either be an image



descriptor block or a Plain Text Extension block (discussed in Construct 4.7). A diagram of the GCE is shown in Figure 4-8.

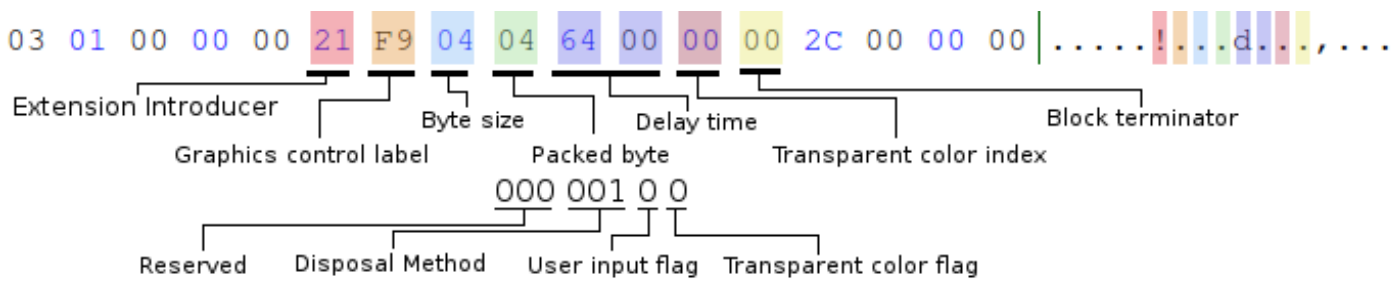


Figure 4-8. Graphic Control Extension

Data can be hidden in a frame by making the background transparent and stacking the frames on top of each other, which constructs the data so that it is not easily visible. An example of this is shown in Figure 4-9, which illustrates how to hide a Quick Response (QR) code within a GIF file.

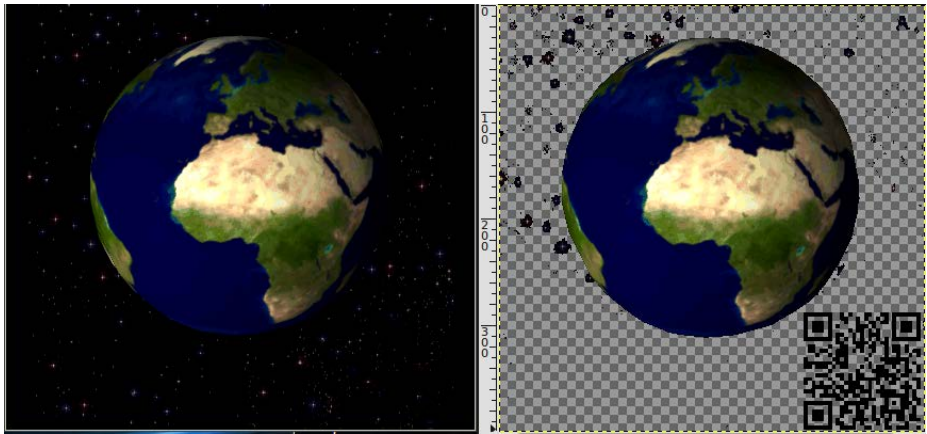


Figure 4-9. QR hidden through use of transparency

Data can be hidden in the GCE by extending the size of the sub-block and using the extra space to store data. Figure 4-10 shows how textual data can be hidden within the GCE by extending the sub-block.

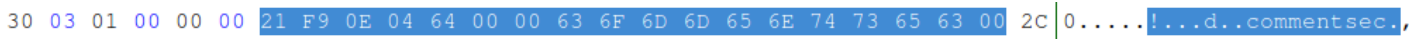


Figure 4-10. Data Hidden in GCE by extending the sub-block

Having individual frames containing small sections or portions of the QR code and using very low time delays between frames, a user can make an image by stacking frames on top of each other. An example is shown in Figure 4-11. Each frame is divided into a smaller portion of the QR code, while the overall animated image resembles the full QR code shown on the right.



**Figure 4-11. Frame Stacking with Short Delay Time**

## RISKS AND RECOMMENDATIONS

**Data Hiding:** Modification of the byte size section could allow a user to hide data after the transparent color index until the block terminator. An example is shown in Figure 4-8.

1. **Validate:** Verify that the byte size is 0x04.
2. **Validate:** Verify that if the Transparent Color Flag bit is set that a Transparent Color Index byte is the final byte of the block before the block terminator.
3. **Remove:** Remove data between the Transparent Color Index and the last byte of the block, the block terminator (0x00). These two bytes should be adjacent to one another.

**Data Hiding:** Through use of the delay time flag a user may try to bypass human review by setting the value high enough that the image may appear to not be an animated image.

4. **Validate:** Verify the value of delay time is set to less than a predetermined value (e.g., 0xFF 00, which is 65,280 milliseconds).
5. **Replace:** Replace the delay time value to a predetermined value.

**Data Hiding:** The Transparent Color Index can be used to hide simple images inside of another image by forcing them to be transparent. An example is shown in Figure 4-9. If the disposal flag is set to combine, it will combine the frames of an animated GIF resulting in frame overlap, which in conjunction with transparency could allow a user

to hide information in some frames when the overlapping occurs. An example is shown in Figure 4-11.

6. **Review:** Have a human review the image to check for hidden information.

**PRODUCT**

GIF89a

**LOCATION**

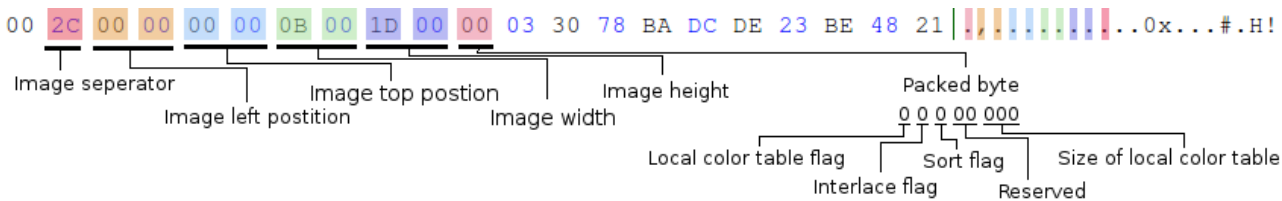
The GCE is identified by the hex codes 0x21 F9 and continues until the Block Terminator. The section immediately following a GCE is either a Plain Text Extension starting with 0x21 or an image descriptor (0x2c).

**4.6. Image Descriptor**

**OVERVIEW:**

Each image in a GIF has an Image Descriptor block. The Image Descriptor block begins with a single byte field called the Image Separator (0x2C). The Image Descriptor block has fields defining the image left position, image top position, image width, image height, and a packed byte with several smaller fields. The image left position is defined as the number of columns (in pixels) from the boundary of the Logical Screen from the LSD block. The image top position is similar but relative to the top of the Logical Screen from the LSD block. The packed byte field's first bit is set if there is a LCT flag. The second bit is set if interlacing is used. The third bit is the sort flag which defines whether the LCT is sorted. The fourth and fifth bits are reserved for future use. The last three bits are used to define the size of the LCT.

Modifying the image descriptor's location parameters, image top and image left position, a user can hide a frame's data off the Logical Screen space. Detecting this requires keeping track of the Logical Screen space from the LSD block. Modifying the image descriptor's size parameters, width and height, you can hide a frame's data by making the frame too small to see. This will not change the image data in the frame; however, it will keep it from being seen. An example is shown in Figure 4-12.



**Figure 4-12. Image Descriptor**



The individual frame's image height and width can be manipulated to show or hide data that may exist in the image but not appear to the user. An example is shown in Figure 4-13 where the bottom image contains a short hidden message. In this example, the size of the second frame (shown in the bottom of Figure 4-13) is truncated to remove the word "Hi" from its boundaries. The end result is a "static" GIF that shows only the red, gray, green image shown in the top portion of Figure 4-13. Since the second frame has a smaller frame size, only the top two colors are shown, and the green is leftover from the previous frame.

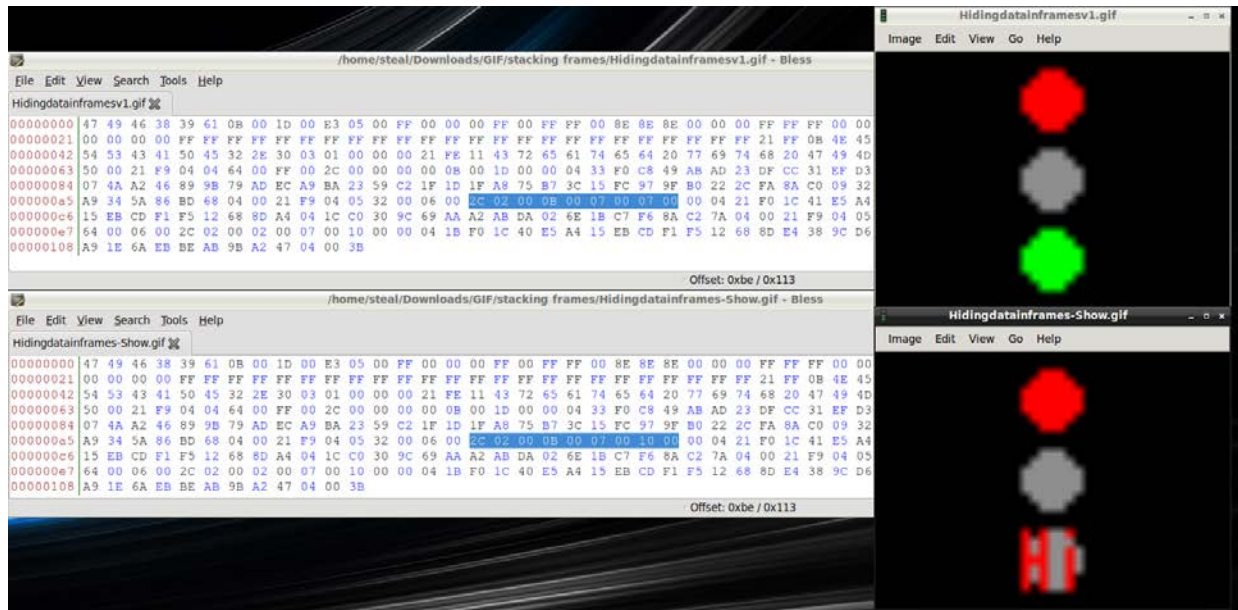


Figure 4-13. Image Descriptor with Truncated Frame

## RISKS AND RECOMMENDATIONS

**Data Hiding:** The individual frame's image location using the left and top position indexes can be manipulated to hide data that may exist in the image but that may not be visible. In addition, the individual frame's image height and width can be manipulated to hide data that may exist in the image but that may not be visible.

1. **Validate:** Check that the image height and width are within the Logical Screen and that the size reflects the number of pixels in the image data.
2. **Validate:** Check that the image location top and left are located inside of the GIF's logical space as defined in the LSD and that the image does not exceed the dimensions set in the LSD.
3. **Review:** Display the image data and have a human review it.

- 4. **Reject:** Reject files with invalid height or width fields that mean an invalid offset into the Logical Screen (off the screen).

**PRODUCT**

GIF87a and GIF89a

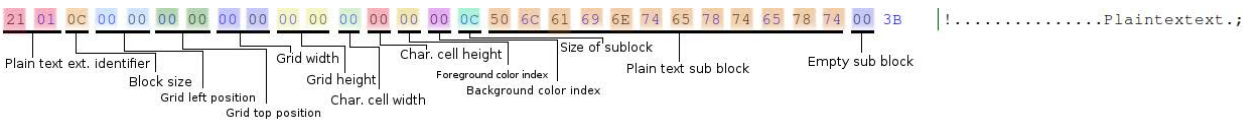
**LOCATION**

Located after a GCE and begins with 0x2C. The Image descriptor section is 10 bytes long.

**4.7. Plain Text Extension (PTE)**

**OVERVIEW:**

The plain text extension (PTE) allows text to be rendered in GIF as part of the image frame. This feature is not supported in most modern browsers and is ignored by image processors. The section is identified by the hex values 0x21 01. The first byte of the PTE indicates the number bytes until you get to first sub-block of text. (The text is broken into sub-blocks of text.) The first byte of a sub-block contains the length of the current sub-block. The length of the data will be read in by the image viewer application, which can include any free text. The next byte after the sub-block will then be the length of the next sub-block. The section ends when it reaches a sub-block of zero or 0x00. An example Plain Text Extension containing plain text is shown in Figure 4-14 below.



**Figure 4-14. Plain Text Extension**

**RISKS AND RECOMMENDATIONS**

**Data Hiding:** The PTE could allow a user to place an arbitrary amount of data in a GIF file.

- 1. **Remove:** Completely remove all plain text extensions from the file as they are not used on most systems.

**PRODUCT**

GIF89a

## LOCATION

The beginning of the plain text extension is 0x21 01 and continues till you reach a sub-block of zero.

## 4.8. Image Data

### OVERVIEW:

The image data section of the GIF file format contains a table of compressed color indexes in sub-blocks. The image data is a series of sub-blocks. Each sub-block is at most 255 bytes each, since the first byte of the sub-block is the length. The first byte before the list of sub-blocks is the LZW Minimum Code Size as shown in Figure 4-15 below. This value defines the initial number of bits that are used for LZW codes in the image data. GIF uses a Variable-Length-Code LZW compression, which grows in size when more codes are used during the compression.



**Figure 4-15. Image Data Table**

Once all the bytes of each sub-block are extracted and decompressed using LZW, each color index corresponds to a color index located in the LCT or the GCT.

## RISKS AND RECOMMENDATIONS

**Data Hiding:** Manipulating the color table values to contain hidden data is possible. Each index points to a particular value in the color table. Index values could also be chosen with information hidden in certain parts of the index value.

1. **Validate:** The image data should consist of indices into the tables. Check that each index is a valid entry from a color table.
2. **Replace:** Shuffle the color tables such that all colors have new indices. Replace the image data with new indices that line up to the previous color values, such that there is no visual change for the image.
3. **Replace:** Modify a select number of bits from color values in each color table. The indices from the image data do not need to change if the colors they point to do not significantly disrupt the image.

**Data Attack:** Inserting a 0x00 3B anywhere into the image data section and deleting the data from this point to the end of the file can cause instability in some applications.

- 4. **Validate:** Check that all image data exists as expected with the image height and width and that no sub-block is unaccounted for in the image data table.

**PRODUCT**

GIF89a

**LOCATION**

Image data is located after an Image Descriptor block when the LCT flag is not set. If the LCT flag is set, then the image data follows the LCT (which immediately follows the Image Descriptor).

**4.9. Comment Extension**

**OVERVIEW:**

The comment extension can contain textual information about the GIF file. Applications may store data about the application that created the GIF in this section. Some steganography tools use this section to hide data. The comment extension starts with 0x21 FE. The data in the comment extension is organized in sub-blocks and continues until it reaches a sub-block with size zero. Each sub-block starts with a single byte defining the length of the current sub-block; data is then read up to the length designated. The comment extension is optional. An example Comment Extension is shown below in Figure 4-16 containing a hidden message.

CE 27 1E 62 69 9E A3 19 82 47 02 00 21 FE 10 41 20 68 69 64 64 65 6E 20 6D 65 73 73 61 67 65 00 3B |'.bi....G...!..A hidden message.;

**Figure 4-16. Comment Extension with Hidden Message.**

**RISKS AND RECOMMENDATIONS**

**Data Hiding and Data Disclosure:** Data can be hidden in comment sections, and this might be legitimate sensitive data placed at this location.

- 1. **Validate:** Check the length of each sub-block is valid in a Comment Extension.
- 2. **Remove:** Remove all comment sections from GIFs.
- 3. **External Filter Required:** Extract the comment section text and pass to an external filter.

**PRODUCT**

GIF89a

**LOCATION**

The comment extension begins with 0x21 FE. It can be located after image data, GCT, application extension, other comment extensions, and plain text extensions. The comment extension ends with 0x00.

**4.10. Trailer****OVERVIEW:**

The trailer is a single byte that indicates the end of the file. This byte is 0x3B. The byte value 0x3b can appear in other locations throughout the image (e.g., image data). The trailer should be detected at the right location, when the previous block terminates. This location should be the last byte of the file, following a valid block. The byte before the trailer should terminate the previous block with the Block Terminator (0x00). This means that the last two bytes of the file should be 0x003b, but the specification states the real trailer is simply 0x3b. Risks involved with the trailer of the GIF format are data hiding as well as data attack if large amounts of data are appended before or after the trailer. Examples of these risks are shown in Figure 4-18 and Figure 4-19. In some cases data after the trailer can be executed when it is interpreted as a different file type. As shown in the example in Figure 4-20, a user can upload a GIF file that also contains PHP code appended to the end of the file. When the server reads the file, it may also try and execute the PHP code.

A client-side model of an attack could be in the form of a GIFAR file. In this case, the browser renders the GIF file but also tries to execute the JAR file appended to the end. The JAR file can then execute code in the browser [4]. An example of this is in Figure 4-18.

The following image in Figure 4-17 is an example of a valid trailer located at the end of the file. There should be no data following the trailer byte.

45 4D 11 01 01 00 3B

EM....;

**Figure 4-17. Valid Trailer**

The following example in Figure 4-18 shows data hidden before the trailer byte but after the sub-block of image data.

```
1D BC 3E AF F7 F3 04 04 00 48 69 64 64 65 6E 20 64 | ..>.....Hidden d
61 74 61 20 62 65 66 6F 72 65 20 74 68 65 20 65 6E | ata before the en
64 20 6F 66 20 66 69 6C 65 3B | d of file;
```

**Figure 4-18. Data before Trailer**

The example in Figure 4-19 shows trailing data hidden after the trailer byte (0x3B).

```
1D BC 3E AF F7 F3 04 04 00 3B 48 69 64 64 65 6E 20 | ..>.....;Hidden
64 61 74 61 20 74 68 65 20 69 73 20 61 70 70 65 6E | data the is appen
64 65 64 20 74 6F 20 74 68 65 20 66 69 6C 65 2E | ded to the file.
```

**Figure 4-19. Data after Trailer**

A common vector for exploiting file upload vulnerabilities on a server is to append executable code after the trailer byte that would be executed server side. The example shown in Figure 4-20 shows a small portion of PHP code that follows the trailer byte of the GIF file.

```
47 49 46 38 39 61 01 00 01 00 80 00 00 FF FF FF 00 | GIF89a.....
00 00 21 F9 04 01 00 00 00 00 2C 00 00 00 00 01 00 | ..!.....,.....
01 00 00 02 02 44 01 00 3B 00 3C 3F 70 68 70 20 65 | .....D..;.<?php e
63 68 6F 20 22 68 65 6C 6C 6F 22 3B 20 3F 3E | cho "hello"; ?>
```

**Figure 4-20. PHP after Trailer**

Shown in Figure 4-21 is a data attack risk where a Java Archive (JAR) file is appended to the end of a GIF and having browsers execute the JAR file after rendering the GIF. This example is commonly referred to as a GIFAR (GIF+JAR = GIFAR).

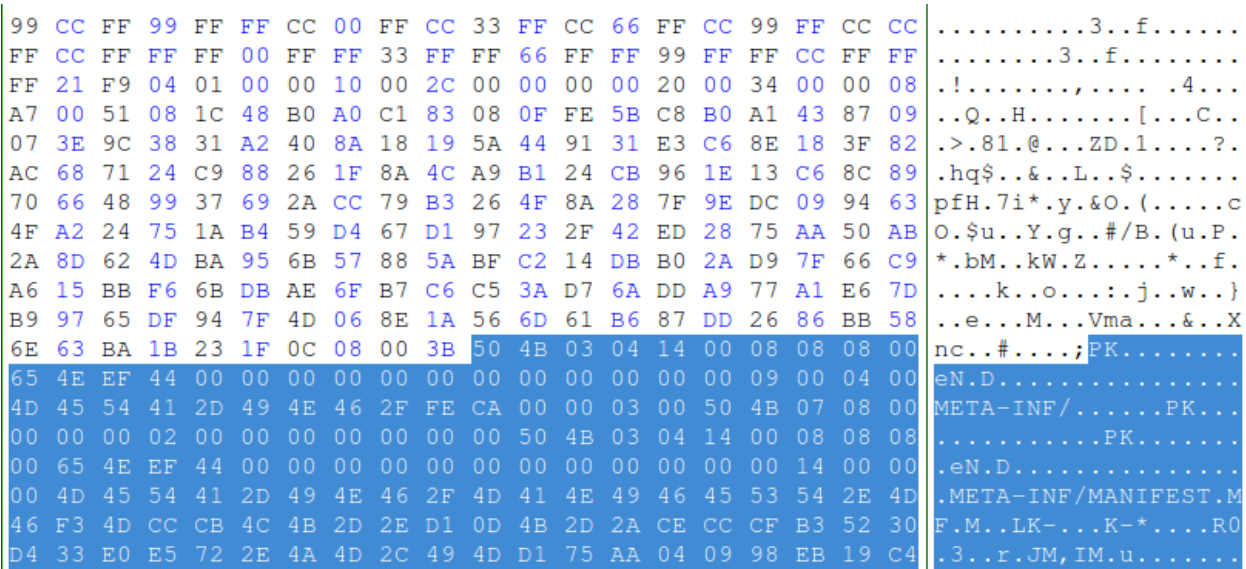


Figure 4-21. JAR after Trailer (GIFAR.gif)

RISKS AND RECOMMENDATIONS

**Data Hiding and Data Attack:** A user can hide arbitrarily large amounts of data after the trailer byte. Extremely large amounts of data appended to GIF file can cause some applications to run out of memory or execute arbitrary code.

- 1. **Remove:** Remove any data after the first trailer byte.
- 2. **Reject:** Reject files with any illegal trailing data.

**Data Hiding:** It is possible to hide arbitrarily large amounts of data before the trailer byte but after the last extension ending in 0x00.

- 3. **Validate:** Verify that there is no data between the last extension and the trailer byte.
- 4. **Remove:** Remove all data between the last block and the trailer byte.
- 5. **Reject:** Reject files with data between the trailer and last block.

PRODUCT

GIF87a and GIF89a

LOCATION

Last byte of the file (0x3B). The previous byte before the trailer should terminate the last block, which should be equal to 0x00.

5. SUMMARY OF RISKS

Table 5-1 - Summary of Risks

Construct	Data Attack	Data Hiding	Data Disclosure
4.1. GIF Header	X	X	
4.2. Logical Screen Descriptor	X		
4.3. Global Color Table and Local Color Table		X	
4.4. Application Extension	X	X	X
4.5 Graphic Control Extension		X	
4.6. Image Descriptor		X	
4.7. Plain Text Extension		X	
4.8. Image Data	X	X	
4.9. Comment Extension		X	X
4.10. Trailer	X	X	
Total	5	9	2



6. ACRONYMS

Acronym	Description
EOF	End of File
GCE	Graphics Control Extension
GCT	Global Color Table
GIF	Graphics Interchange Format
ISG	Inspection and Sanitization Guidance
JAR	Java Archive
LCT	Local Color Table
LS	Logical Screen
LSD	Logical Screen Descriptor
LZW	Lempel-Ziv-Welch Compression
PCX	Personal Computer Exchange
PNG	Portable Network Graphics
PTE	Plain Text Extension
QR Code	Quick Response Code
XMP	Extensible Metadata Platform

## 7. REFERENCES

- [1] CompuServe (1987). GRAPHICS INTERCHANGE FORMAT. Retrieved from The World Wide Web Consortium: <http://www.w3.org/Graphics/GIF/spec-gif87.txt>
- [2] CompuServe. (1990). *GRAPHICS INTERCHANGE FORMAT*. Retrieved from The World Wide Web Consortium: <http://www.w3.org/Graphics/GIF/spec-gif89a.txt>
- [3] Flickinger, M. (2005, January 24). *What's In A GIF - Bit by Byte*. Retrieved from <http://www.matthewflickinger.com/lab/whatsinagif/>
- [4] McFeters, N., Carter, R., & Heasman, J. (2008, August 2). *Extreme Client-Side Exploitation*. Retrieved from Blackhat: [https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-McFeters/BH\\_US\\_08\\_Mcfeters\\_Carter\\_Heasman\\_Extreme\\_Client-Side\\_Exploitation.pdf](https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-McFeters/BH_US_08_Mcfeters_Carter_Heasman_Extreme_Client-Side_Exploitation.pdf)
- [5] Roelofs, G. (1997, January). *History of the Portable Network Graphics (PNG) Format*. Retrieved from libpng: <http://www.libpng.org/pub/png/pnghist.html>