Global Information Assurance Certification Paper

Images as a Threat Vector

SANS/GIAC GSEC Practical V1.4b
Option 1: Research on Topics in Information Security

David J. Wilson
Re-submitted
Monday, April 05, 2004

Abstract

There has been considerable anxiety created publicly and in the security community by the appearance of a new threat from image files. The widespread use of image formats on the Internet and now the explosion of picture sharing from digital photography has many people (and organizations) worried about "infection" from image files. The format of different image files will be reviewed and methods that might be used to carry malware analyzed.

The conclusion is that, although image files could be used as a carrier for malware, there is nothing inherent in these files (any more than any other file that a user would click on) that makes them dangerous. All the threats essentially rely on exploiting another, pre-existing threat vector.

1. Introduction

In 1994 there was an April Fool's hoax about a virus contained in a ".jpg" file. Possibly because they are so common, speculation has continued as to whether JPEG files could be used as a threat vector. In June of 2002, a hacker, supposedly from the Philippines[1], created a proof-of-concept exploit and sent it to the security community to "prove" that a JPEG file could deliver a payload. The details of PERRUN-A will be analyzed in section 3, but the controversy raged in the community, with the debate ranging from opinions of doom to the exploit being labeled "lame".

Some of the common image formats will be briefly described and opportunities for hiding a payload in them discussed. Note that this is distinctly different from Steganograpy, where a message is sent from a source to a destination agent that is expecting the message and has the complementary Stego tool to extract it. Here we are considering the delivery of a malicious payload to an unsuspecting recipient.

Although no known occurrences of a true virus in an image file have been found in the wild, several exploits using image files will be discussed to demonstrate how the hype is maintained about the dangers of image files.

Some theoretical possibilities will be discussed about how a true virus might exist in an image file and the probability of it succeeding.

The conclusions will include some comments on the social engineering aspects of the image file threat.

2. Types of Image Formats

PGM

PGM bills itself as the "lowest common denominator" grayscale file format, and indeed, it contains only the minimum information to define an image: a "magic number" (P2 for PGM), the width, the height, and the maximum gray level, followed by the width*height values, all in ASCII and separated by whitespace. The lines are even limited to 70 characters so it is easy to view on the screen!

Even this minimalist format does have a comment field, delimited by the octothorpe "#". Exploits where the payload is hidden in comment fields have been reported. The man page pleads for leniency in interpreting the format.

A variant, with a magic number of “P5” represents the gray values as a stream of bytes.

PNG

PNG[2] (pronounced “ping” by the authors) is the Portable Network Graphics format and was officially released in 1997 as RFC 2083[3]. The file structure consists of a fixed 8-byte Signature followed by a series of “Chunks”. Chunks consist of a 4-byte unsigned integer Length, a 4-byte Chunk Type, the Chunk Data, and lastly a 4-byte CRC, calculated on the Chunk Type and the Chunk Data. The Chunk Type is ASCII and contains embedded flags to indicate further attributes of the Chunk: critical/ancillary, private/public, reserved, or “safe-to-copy”.

The format goes to great lengths to ensure file integrity, as it is intended for images transmitted over the network. However, in their efforts to provide flexibility, the authors have allowed for “Private” chunks, designed to provide information for specific applications that does not have wide applicability and therefore does not need to be part of the standard. Applications are admonished to ignore any unknown or non-conforming chunk and only treat malformed Critical chunks, which are necessary to render the image, as fatal errors. Thus, applications or viewers using PNG are subject to the usual exploits if they are not careful to follow the rules.

BMP

The BitMaP format is the native MS Windows file format. It can store 1, 4, 8 or 24-bit pixels and optionally use 4 or 8-bit Run Length Encoding. It has a simple structure: a BITMAPFILEHEADER containing the code “BM” for BMP format, the total file length and an offset to the pixel data; next comes the BITMAPINFOHEADER which contains the height and width of the image, the image depth, the compression flag, the actual size of the pixel data, and the number of colour table entries; and next comes the colour table of BGR values arranged as a list of 4 byte values; and lastly the byte stream of pixel data.

TIFF

TIFF is the Tagged-Image File Format, a format originally put in the public domain by Aldus/Adobe[4]. A new variant is the GeoTIFF format[5], designed by the GeoTIFF working group to include geo-location information for the

georeferenced or geocoded raster images increasingly being demanded by the remote sensing community. Tiff uses “tags” and a platform-independent data format representation to describe the structure of raster data. GeoTIFF uses MetaTags (or GeoKeys) to encode multiple geographic elements into 6 new registered TIFF tags. These are essentially another level of indirection from the TIFF Tags. For instance, the GeoKeyDirectoryTag defines the new GeoKeys. These, in turn, are defined in terms of existing TIFF tags or the 5 other new tags for GeoTiff data types needed to codify coordinate systems, coordinate transformations and projections. For example, IEEE double precision was required for the mathematical accuracy of the transformations using the new GeoDoubleParamsTag.

Although it is discouraged, the format is extensible and can include proprietary or private data within the allowable structure. The GeoAsciiParamsTag contains ASCII data. Although the strings in this Tag were originally intended to be NULL terminated, to accommodate naïve readers, TIFF replaces the nulls with the vertical bar (“|” or “pipe” symbol). The image reader is responsible for replacing the “|”s with NULLs when returning the values of the Tag back to the application. This is another opportunity for sloppy programming to allow a buffer overrun. Also, since the directory uses offsets to point to the data within each Tag, arbitrary data could be stored at the beginning of the ASCII Tag.

JPEG

The latest evolution of the Joint Photographic Experts Group (JPEG2000)[6] is a much more complex format. The format is hierarchical, consisting of a number of “boxes”, some of which may be “superboxes” containing sub-structures. However, there is a restriction:

> “Note that the file is a strict sequence of boxes. Other boxes may be found between the boxes defined in this Recommendation/International Standard. However, all such data shall be in the box format; no other data shall be found in the file.”

The wish to make the format flexible and enable the incorporation of image metadata leads to the creation of new boxes not defined in the spec, as long as it does not change the appearance of the image. Thus “private boxes” can be created and will be ignored by readers that do not understand them. This opens the door to inserting malware!

Each box consists of a 32-bit box length “LBox”, a 32-bit box type “TBox”, an optional 64-bit Extended box length “XLBox” and the binary contents of the box DBox. Special values of the Lbox field are 1, indicating the length is actually in the XLBox field, and 0, which means that the length was unknown

when the field was written (the file is a potentially very large sequential bytestream). When LBox is zero, this box contains all the data up to the end of the file. This is another opportunity for a buffer overrun!

The format also allows for an XML box, containing arbitrary xml code. UUID boxes allow for vendor binary data and arbitrary URLs to describe the data format. These are more openings for mischief!

CEOS

CEOS is a family of formats designed by the Committee on Earth Observing Satellites specifically for the exchange of satellite Remote Sensing data and is customized for each satellite mission, as there is generally great dissimilarity between the data from each satellite. For Synthetic Aperture Radar missions, such as ERS and RADARSAT[7], for instance, the format is a complex hierarchy contained in five separate files: the Volume Directory File, the SAR Leader File, the SAR Data file, the SAR Trailer file and the Null Volume file.

The Volume Directory File describes which files are contained within the dataset. The SAR leader and Trailer Files contain many different records depending on the operating mode of the satellite. These records try to be self-describing, but experience dictates that the format document is required for any detailed analysis of the contents.

The contents of the files and records are tightly defined by the format document and there is essentially no room for insertion of any extraneous data into the files. This format is fairly obscure and readers of the format tend not to be particularly lenient in interpretation, so this is actually an example of a “safe” format.

HDF

The Hierarchical Data Format was developed by the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign in collaboration with the Information Technology Institute of Singapore in 1988[8] for sharing different types of scientific data, including raster and JPEG images. It is a portable, flexible, public domain, hierarchical, self-describing format with an index and an interface library between the physical file format and the NCSA application layer. The format is supported by many commercial Remote Sensing systems such as PCI and AVS and public domain systems such as GRASS and their own NCSA Visualization tools.

The format supports an “Annotation” data model, which can contain labels as null-terminated strings or more complicated structures, including source code. The call to write the annotation is passed a buffer and length, so the possibility of inserting arbitrary code in the file is possible. The relatively obscure nature of the format, however, reduces the risk of a viewer being tricked into doing something with this data.

3. Threat Analysis

The format of image files runs from almost trivial to very complex. Although there are places in some file formats to hide malware, which will not invalidate the format, the trick is to get the image file interpreter to do something nasty with whatever it finds in the format.

Although many interpreters are built to be forgiving, generally they either complain about or ignore data that they do not understand.

Simple buffer overflows are certainly possible as in the buffer overflow in Debian GNU/Linux Window Maker[9]. This window manager made a simple oversight in not checking for overflow when it calculated the buffer size by multiplying the width * height of the image to be displayed. This could apply to any image format.

The Internet Explorer Object Data Remote Execution Vulnerability illustrates another generic vulnerability[10]. Microsoft uses Object Tags to embed Active-X into HTML. A Trojan can be loaded by hiding it in a file with a “safe” extension, but once loaded by Internet Explorer (5.01, 5.5, 6.0), IE ignores the extension and uses the MIME type returned by the server to decide what to do with the file. For example, an image or other html file could be specified in the /object tag, IE will download the supposedly safe file type which actually contains “subseven”, and if the server says the MIME type is “HTML Application”, the client will execute it, infecting the host with the back door.

An earlier IE exploit[11] was similar, but IE was tricked into downloading a “bad” file, such as a .exe. by embedding MIME headers that described the file as a JPEG file, but then IE would use the actual extension to decide to execute the file. This is the inverse of the previous example!

In the case of W32/PERRUN-A, the virus was a “proof-of-concept” sent to the virus community in June of 2002 and first reported by Sophos[12]. The author wanted to prove that JPEG files could “infect” a system. The malicious code was indeed tacked onto a jpeg file so that it would not interfere with the display of the image, however, the infection depended on the receiving system already being infected with a virus. This pre-existing infection, usually called “proof.exe”, would change the default program to display jpeg files to a new file called EXTRK.EXE.

Thus, when the unwitting user clicked on the JPEG file, EXTRK would look for a new payload, load it into memory and execute it, then pass the jpeg to the normal image viewer code. The user would probably not be aware that anything had happened. Normal jpeg files would display properly, without incident.

The fear was not so much the destructive nature of the payload, but the concept that viri could be carried by common image files. The social engineering possibilities were endless, as millions of users could be sucked into opening these files, whether they be pictures of grand-children or nasty porn.

No "In the Wild" occurrences of W32/PERRUN-a have been found, but speculation still continues that some other way than the pre-existing virus will be devised to load the malicious code embedded in the jpeg file. One such method would be to exploit the GIF/ JPEG Comment Vulnerability[13] that afflicted versions of Netscape prior to version 4.77. The problem, reported in March 2002, affected both JPEG and GIF89a files, as they both allow inline comments, which the standard libraries ignore. However the FreeBSD 4.5 Ports Collection shipping at that time contained code that bypassed the standard libraries and allowed execution of JavaScript code if it was found in the comment fields. This "feature" supported a non-standard URL, "about:" which would display information about the browser configuration. For example, "about:cache" would display all recent files in the user's disk cache, with a timestamp. This is not exactly the information one would want relayed back to a hostile web server!

Although these Netscape ports are not installed by default, nor is the enabling of JavaScript recommended in security-conscious installations, the user community is wont to enable any feature that improves their "web experience". Indeed, JavaScript is required for some sites.

Another method would be to exploit the Netscape JPEG-Comment Heap Overwrite Vulnerability[14]. This was a classic example of the buffer overrun trick, where the browser did not use the standard JPEG library, loading the comment field into memory using a custom function. Setting the two-byte comment length field to 1 results in a memory allocation of 0 (1 – 2 for the length field + 1 for the terminating null). The malloc will succeed and then sloppy programming uses the unsigned buffer size of -1 to overwrite memory, including the heap, with the entire JPEG file.

Of course, it would take considerable skill to construct a payload that would execute after the exception caused by the over-run. However, this does supply the necessary "EXTRK" function to get the malware from the JPEG file into memory. This Netscape exploit would also apply to use of Communicator for mail or news.

A detailed scenario of such a payload construction exists at www.openwall.com/advisories/OW-002-netscape-jpeg.txt.

This analysis, written by “Solar Designer”, gives a plausible approach for overwriting heap pointers so that a debug hook in a specific version of malloc in the GNU libc will give control to the malicious code, stored on the stack. Although described for Linux/x86, the scenario would work for other OSs as well, and the author points to one Win32 example.

Solar Designer stops short of giving a cook-book explanation which could be used by “script-kiddies” wanting to graduate to more intricate code design, however the detail laid out actually does provide the “proof-of-concept” so sought after by the writer of PERRUN-A.

Another method would be to try an attack like the xloadimage Remote Exploit[15], which employed a weakness in the way Netscape uses “helper applications”. This is another proof of concept presented by “zen-parse” in July 2001, which describes how Netscape could be tricked into passing a file labeled as a TIFF to the xloadimage helper, which then examines the file and determines it is actually a FACES Project format image. A coding error (buffer overflow) in the FACES processing code then causes a Segment Violation, which in turn could be used to execute the malicious code. He proposes inserting some binary code from bind.c onto the heap and hoping it will execute and bind a certain port to /bin/sh when the exception occurs.

Zen-parse notes that many of the file formats handled by xloadimage can be aborted due to a Segment Violation due to a single byte modification to the file.

One example of a JPEG file apparently containing malware was the Coppermine Photo Gallery Remote Compromise[16]. In this exploit, the web-based photo gallery software was designed to upload JPEGs to a server for sending e-cards, etc. The problem was that the extension checking had a bug where it would allow double extension “.jpg.php” files. Thus, if one could construct a file which was a valid jpeg file AND a valid PHP script, Coppermine would upload it to the server. Then the user could execute it with the privileges of the user the Coppermine PHP script was running under on the server.

An example of an exploit of PNG images was the RealPlayer PNG Deflate heap corruption vulnerability[17]. This affected certain versions of RealPlayer 8 and RealOne, popular software for playing audio and video over the net. The software supports many formats, including PNG. The problem in this RealNetworks application was a bug in the Deflate algorithm that incorrectly interpreted a run-length code as 2*32. The program space and heap would be overwritten. When an exception occurred at the end of the program space, a malloc/free call could be misused to overwrite memory, such as the unhandled exception function pointer, gaining execution control.

4. Conclusions

We have seen that there are a wide variety of image formats available. Indeed this paper only looks at a small sample. Looking at the format of these files, we see a wide range of sophistication from simple blocks of bytes to complex structures with pointers and encoded attributes. The naïve view would be that it's all just binary and text data and is never "executed" so must be harmless. This is the "Don't shoot the messenger" attitude.

We dismiss the tempting JPEGs sent with hidden extensions to be blithely executed by unwitting Windows users as coarse tricks perpetrated on the great unwashed. And we may grumble at Microsoft for making such pranks so easy.

We may also dismiss the PERRUN-A proof-of-concept as being "lame" as it required a previous infection to make it work. A closer look at the two heap vulnerabilities shows that providing the pre-existing virus may not even be necessary if hackers can continue to find bugs in the OS, the interpreters, the browsers, or the helper apps, all provided by "trusted" sources.

Of course, "defense in depth" applies here, as we should apply all the techniques of user awareness, disabling JavaScript, running anti-virus software on all files and all attachments, keeping virus files up-to-date, and keeping patch levels for all software up-to-date to prevent the "helper" vulnerabilities.

An interesting aside is the continuing debate over whether it is better to have open source so that vulnerabilities can be detected by the community and fixed versus the proprietary way, where it may be more difficult to exploit a vulnerability if one does not have the source code. Similarly, the authors gave varying degrees of detail about the exploits they were explaining, to avoid giving too much information to "script-kiddies". "Security by obscurity" is a definite part of the defense mindset.

Although the Coppermine exploit also depends on a double extension ploy that we previously dismissed as "coarse", the assertion of the author of the vulnerability warning, Berend-Jan Wever, that "It is trivial to create a file that is a valid JPEG and a valid PHP script" should give one pause.

So, are image files dangerous?

We have seen that, with some notable exceptions, image file formats do not contain actual malware, but the social engineering possibilities of image files in this age of digital photography and rampant file sharing make them a prime medium for attempting to exploit other vulnerabilities. It is a long time since the "glass house" where all software was controlled. It is now a jungle out there and there are a seemingly infinite number of combinations of data and software available on the web so that nothing can be guarantee to be safe. However, if

one follows all the recommended procedures, implementing defense in depth, and is aware that there are possible risks, then those risks can be minimized.

Click carefully!

5. References

Boliek, Martin; Christianopoulos, Charilaos; Majani, Eric. JPEG 2000 Part 1 Final Committee Draft Version 1.0, ISO/IEC JTC1/SC29 WG1. March 2000.

Kientzle, Tim. Internet File Formats. Scottsdale: Coriolis Group, 1995.[18]

---

[1] "Filipino claims to be JPEG virus author". Sophos> Virusinfo> Articles. 18 June 2002 . URL: www.sophos.com/virusinfo/articles/perrun2.html (2003-09-09).

[2] Roelofs, Greg. "Portable Network Graphics. A Turbo-Studly Image Format with Lossless Compression". 24 July 2003. URL: http://www.libpng.org/pub/png/png.html (2003-09-09).

[3] Boutell, T, et. al. "PNG (Portable Network Graphics) Specification, Version 1.0". Request for Comments: 2083. March 1997. URL: http://www.ietf.org/rfc/rfc2083.txt (2003-09-09).

[4] Adobe Developers' Association. "TIFF Revision 6.0 Final". June 3, 1992. URL: http://partners.adobe.com/asn/developer/pdfs/tn/TIFF6.pdf (2003-09-09).

[5] Ritter, Niles; Ruth, Mike. "GeoTIFF Format Specification, GeoTIFF Revision 1.0". Specification Version: 1.8.2. Modified: 28 December, 2000. URL: http://remotesensing.org/geotiff/spec/geotiffhome.html (2003-09-09).

[6] Boliek, Martin; Christianopoulos, Charilaos; Majani, Eric. JPEG 2000 Part 1 Final Committee Draft Version 1.0, ISO/IEC JTC1/SC29 WG1, Menlo Park, 2000.

[7] RADARSAT Data Products Specifications. RSI-GS-026. Revision 3/0. Vancouver: RADARSAT International Inc. May 08, 2000. URL: ftp://ftp.rsi.ca/ceos_dfn/D4_3-0.zip

[8] The National Center for Supercomputing Applications. "Frequently asked Questions about HDF". May 19, 2003. URL:

http://hdf.ncsa.uiuc.edu/HDF-FAQ.html (2003-09-09).

[9] Akkerman , Wichert. "Debian Security Advisory DSA-190-1: Buffer Overflow in Window Maker". Neohapsis Archives. November 7, 2002. URL: http://archives.neohapsis.com/archives/linux/debian/2002-q4/0500.html (2003-09-09).

[10] Maiffret, Marc. "Internet Explorer Object Data Remote Execution Vulnerability". Neohapsis Archives. August 20, 2003. URL: http://archives.neohapsis.com/archives/ntbugtraq/2003-q3/0174.html (2003-09-09).

[11] Cohen, Cory F. and Lanza, Jeffrey P. "Vulnerability Note VU#443699, Microsoft Internet Explorer Does Not Respect Content-Disposition and Content-Type MIME Headers". Document Revision 21. CERT Vulnerability Database. June 07, 2002. URL: http://www.kb.cert.org/vuls/id/443699 (2003-09-09).

[12] "W32/Perrun-A". Sophos Virusinfo Analyses. June 2002. URL: www.sophos.com/virusinfo/analyses/w32perruna.html (2003-09-09).

[13] Wesch, Florian. "GIF/JPEG comment vulnerability in Netscape". AUSCERT External Security Bulletin Redistribution. 13 March 2002. URL: http://www.auscert.org.au/render.html?it=1788 (2003-09-09).

[14] "Netscape Communicator JPEG-Comment Heap Overwrite Vulnerability". Securityfocus Vulnerabilities Discussion. URL: www.securityfocus.com/bid/1503/discussion (2003-09-09).

[15] zen-parse@gmx.net . "xloadimage remote exploit - tstot.c". Bugtraq Mail-list Archive. 10 Jul 2001. URL: http://cert.uni-stuttgart.de/archive/bugtraq/2001/07/msg00160.html (2003-09-09).

[16] Wever, Berend-Jan. "Coppermine Photo Gallery remote compromise". Security Corporation Articles. April 07, 2003. URL: http://www.security-corporation.com/articles-20030407-003.html (2003-09-09).

[17] Rizzo, Juliano; Friedman, Agustin Azubel; Acselrad, Bruno and Sarraute, Carlos. "RealPlayer PNG deflate heap corruption vulnerability". CORE-2003-0306. Mar 28, 2003. URL: http://archives.neohapsis.com/archives/vulnwatch/2003-q1/0156.html

[18] Kientzle, Tim. Internet File Formats. Scottsdale: Coriolis Group, 1995.