MITRE TECHNICAL REPORT

MITRE

File Format Identification

Report on MITRE Sponsored Research

Kent Vidrine

January 2010

MITRE TECHNICAL REPORT

File Format Identification

Report on MITRE Sponsored Research

Kent Vidrine

December 2009

Abstract

Digital forensics examiners acquire large numbers of files as they carry out their investigations. Effective exploitation of the files found on seized media depends upon accurate file format identification. However, file format identification is a hard problem. Existing tools and techniques fail to identify all of the files that an investigator may have interest in. This paper describes the state of the art in file format identification, existing tools and evaluations thereof, and some of the new techniques developed for the File Format Identification MITRE Sponsored Research project.

Table of Contents

List of Figures

List of Tables

1 Introduction

In the news, we read stories with headlines like: “Libraries’ computers seized in Mariam case” [thestar.com], “Official: Investigators seized computer from Markoff” [boston.com]. A brief Internet search reveals many similar stories. What happens with the seized evidentiary material? Engineers employ systematic processes to extract files and other data from the media. And then analysts search for clues among the seized data.

Modern computer disks have large capacities and can contain tens of thousands of files, or more. With so much space available to them, users consume the space with software, documents, music, cached data from Internet activity, and other items. This means that forensic examiners acquire lots of files, too many to manually inspect. Because of this large volume of data, automated tools assist in the data exploitation job. These tools develop a catalog of files and then an attempt is made to classify each file by format.

Effective and efficient discovery of meaningful information exploitation depends on file format identification because each file format uses its own unique combination of data structures and internal layout. Restated, to understand the information contained within a file one must first understand its structure and then use the right data exploitation tool for that structure. For example, to understand an image file, one must first recognize that it is an image file and then use an image viewer (or other processing software) to view the contents of the file. Similarly, to read an electronic document, one must first discover what kind of document it is (Adobe Portable Document Format, Word Document, WordPerfect Document, etc) and then use the appropriate document viewer (or text extractor) to render the file’s contents.

Accurate file format identification is crucial for effective data exploitation. However, current file format identification tools consistently fail to identify a substantial percentage of the files contained on seized media. This failure manifests itself in two ways: the tool makes no format assertion, or the format assertion is erroneous.

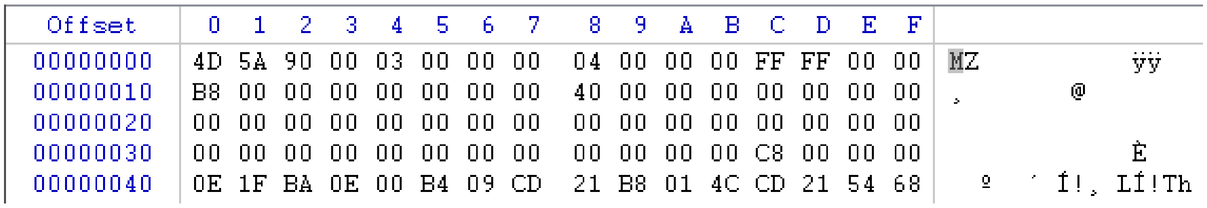
File format identification is a hard problem. Why? There are many reasons, including:

- There are thousands of formats, many of which are undocumented or proprietary.
- The features of a file format that can be used to identify conformant files may be difficult to discover, describe, and detect. In fact, software vendors, who are the ones responsible for creation of formats and the files that conform to those formats, have no incentive to build distinguishing features into files.

1.1 Existing File Format Identification Techniques

There are some existing techniques, and tools that implement those techniques, which were created to accomplish file format identification. They generally rely either on the existence of specific values at predictable locations within a file, or on some file naming convention, or a combination thereof.

The most fundamental way of determining what format a file conforms to is to simply open the file with some kind of viewer. Using this technique, one quickly discovers that some files, namely those that contain plain text, are viewable with a simple text editor such as Windows Notepad. However, many files have unintelligible contents when viewed with a text editor. The next tool, after a text editor, that a user can use for such files is a hex viewer, which renders the file's contents by displaying the file's hexadecimal byte values. A hex viewer may reveal file format clues, such as magic bytes (described below). Figure 1 shows a sample hex viewer rendering of a Windows Portable Executable file.



```
 Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000   4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00   MZ          ÿÿ
00000010   B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00   ¸       @
00000020   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00000030   00 00 00 00 00 00 00 00  00 00 00 00 C8 00 00 00               È
00000040   0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68     º  ´ Í!¸ LÍ!Th
```

Figure 1: Hex Viewer Sample

1.2 Magic Bytes

Some formats require the presence of specific byte values at specific locations within the contents of the file. For example, the ZIP file format requires that the file begins with the hexadecimal byte sequence 50 4B (or PK in ASCII). A file format identification tool might examine the first few bytes of a file, discover that byte sequence, and then render the assertion that the file is a ZIP file. Similarly, the WAV file format requires that the file begins with the hexadecimal byte sequence 52 49 46 46 (or RIFF in ASCII). A tool might discover those bytes at the beginning of a file and render the WAV format assertion. These identifiable byte sequences are commonly called *magic bytes* or file *signatures*. Many file formats have magic byte sequences. However, many of the most common formats lack such sequences, or in some cases, a sequence of bytes within a file may match more than one magic byte sequence, which leads to multiple format assertions for a single file. Reliance on magic bytes also fails to identify formats that lack magic byte sequences.

1.3 Filename Extension

There is a file naming convention in use on most computers where the last few characters of a file's name, those that follow the last period in the name, indicate its format. For example, PDF indicates that a file conforms to the Adobe Portable Document Format; TIF indicates that a file conforms to the Tagged Image File Format, and so on. This portion of the file's name is commonly referred to as the filename extension. Many of the most commonly found formats have a well-known filename extension. However, there are files that lack an extension. Further, there are many extensions for which more than one candidate format is indicated, and some files have names that incorrectly indicate their formats.

1.4 Technique Summary

To summarize, there are two primary techniques for file format identification: placing trust in the naming convention, and brief inspection of file contents to look for identifying sequences. The techniques are summarized in Table 1.

Technique	Advantages	Disadvantages
Magic byte sequence discovery	Fast detection Easy to implement and extend	Some formats lack magic Ambiguity among some formats No comprehensive catalog Performs poorly on text files
Filename extension naming convention	Fast detection Easy to implement and extend	Some filenames lack extension Extension collisions No comprehensive catalog Some files are improperly named

Table 1: Summary of File Format Identification Techniques

1.5 Tool Assessment

In order to evaluate the extent to which existing tools correctly identify files' formats, we assessed several tools.

The first tool that we assessed was the UNIX file command and its [libmagic] API. The results show that the tool accurately identifies about 80% of the files that we submitted to it. However, this rate varies widely, depending on the distribution of formats contained in the sample of files. The tool uses a large file, named "magic", that contains a list of magic byte sequences and the formats that the sequences indicate. The tool tries each sequence in turn, looking for a match within a file to be identified. When a magic byte match is found within a file, the tool asserts the associated format.

Another tool that we evaluated was Oracle [Outside In File ID]. This proprietary tool inspects the contents of a file to discover features that can be used to identify the file's format. The tool rarely makes mistakes, but only identified about 60% of the files that we submitted to it.

We also evaluated [DROID], [TrID], [mime-util], and various other tools. No tool was able to consistently and correctly identify more than 80% of the files that we submitted to it.

With that evaluation done, we developed some prototype tools that rely on other techniques.

2 Research

It is unlikely that any single tool or technique will ever be able to identify file formats with 100% accuracy. However, a combination of tools, each of which can be tuned for low error rates, can be used to positively identify far more files than any single tool in existence today. What follows is a description of some prototype tools that were developed for the File Format Identification research project.

2.1 Filereg

The first new tool that we developed was **filereg**. This tool is based on the fact that individual personal computers running Windows function properly by simply relying on filename associations. The Windows Operating System stores its registry entries in a series of files. The Software hive resides in a file named "Software". In a disk image (dd, Encase, FTK, etc), the registry files are readily accessible for data extraction. We used the PERL CPAN Parse::Win32Registry module to acquire the filename associations. Once these associations are mined, we then acquire a list of filenames *from the same disk image* and we assert formats accordingly. The bottom line for this tool: it rarely makes a mistake (error rate less than 1%), but there are many filename extensions for which there is no mapping in the registry. Also, the tool cannot assert a format for filenames that have no extension. The real value of this tool is that its list of filename extensions, and the formats that they indicate, is *context-sensitive*. Each computer has its own set of installed software and filename associations. This tool disambiguates this kind of collision.

2.2 Mfile

Another tool that we developed was **mfile**. This tool uses a painstakingly-compiled rule set to arbitrate tool output. Many format identification tools consistently get some formats right and some formats wrong. So for a given file, if tool A says X and tool B says Y, then those behaviors are consistent enough to have an arbitrator assert the actual format, based on the combination of those tool inputs. Mfile is based primarily on libmagic output, filename extension, and directory location, and it can easily be extended to include input from Oracle Outside In File ID or any other tool. When run on ground truth, the tool asserts with high accuracy, which makes sense given the fact that it was tuned using that very same ground truth. When run on files from the research corpus, the assertion rate is in the low 90% range.

2.3 Validate

Yet another tool is **validate**. The theory with this tool is that many files can be validated by attempting to parse each file with a library that knows how to handle that file. So we have a library for parsing XML, another for HTML, another for Windows INI files, another for images, etc. While developing this tool we found that the best results are obtained when first determining whether a file is composed of text, then attempting either text parsing or binary parsing, but not both. This tool's best use case is for the purpose of easily identifying the most common files and removing the *files and formats* from the set of files left to be identified. For example, consider an arbitrary file. We use the validate tool to determine whether it is an image file (JPG, GIF, BMP, etc). If we identify the file as an image, then the job is done. If the file does not conform to one of those formats, then we continue to try to identify the file, but we no longer consider any of those image formats as a possibility. Consequently, this tool either tells you what a file is, or it tells you what it is not.

2.4 File Fingerprints

[Mason McDaniel] described several file format identification techniques in his Masters degree thesis. The techniques are based on statistical analysis of the files' contents and derivation of a file format *fingerprint*. The most valuable tool from among the McDaniel techniques is one that can quickly identify magic byte sequences, if they exist. The value of this technique is that, when investigators encounter a set of files that are clearly of the same format, but for which no previous experience exists, then the magic byte sequence can be identified and cataloged for future use.

2.5 Tokens

Early in Computer Science curricula, students learn about text processing. Specifically, the technique of handling strings of text and chopping the strings into individual words, or *tokens*, is learned. Based on the theory that one method of identifying files is to look for expected tokens that are present in all of the files of a given format, we experimented with that technique. We expect this technique to be especially useful for identifying source code files because most programming languages require the use of a well-defined set of keywords. If we find those keywords in a file, then we can assert that the file contains the appropriate kind of source code. Just as importantly, if a given text file lacks the keywords that a programming language requires, then we can exclude that programming language from further consideration. We used the Natural Language Toolkit (NLTK) to parse text files into tokens and then wrote two tools to explore this technique.

2.5.1 TokenValidation

We gathered text files from ground truth that were of a known format and used a script called **intersection** to discover the tokens that were common to all of the files in that training set. Using that list of tokens, we then used another script called **tokenvalidation** to check for the presence of those tokens in the test set. This technique works well for text-based formats that always require the presence of a core set of tokens.

2.5.2 Probabilistic Token Validation

The problem with the token intersection technique is that it relies on there being at least one token in common for all files of a given text-based format. For example, the only tokens that MUST be present in a Java source code file are the curly braces {}. Additionally, one of two keywords, either *class* or *interface* must be present. So the token intersection problem becomes one of finding *some* tokens from a set. But unlike the previous intersection/validation technique, this approach does not require any specific token to be present in order to make a format assertion. For example, revisiting the Java source code identification problem, if we find either *class* or *interface* in the contents of the file, along with other Java keywords, then we may have enough confidence to assert that the file does in fact contain Java source code. To test this theory and supplement the token intersection technique, we developed a set of scripts called **nnn_sc**, where nnn is the name of a source code language such as Java or PERL. These contain lists of programming language-specific reserved words and other tokens that appear frequently in those files. A score is computed for each candidate source code file. While the scores' magnitudes are not necessarily meaningful, the numbers at the extremes, both small and large numbers, indicate files that are either very unlikely or very likely (respectively) to be source code of the target language.

2.5.3 Summary

The following table summarizes the tools, their techniques, pros, and cons.

Tool	Description	Pros	Cons
Filereg	Registry mining	Low error rate, context-sensitive	Fails to identify files that lack filename extensions or associations
Mfile	Tool output arbitration	Low error rate, uses input from many tools for better results	Tedious to tune
Validate	Sequential parsing with special-purpose libraries	Low error rate	Limited library availability
Fingerprint	Statistical analysis	Finds magic byte signatures	Relies on other tools to use the discovered magic bytes
Token intersection	Finds required keywords	Usually asserts correctly when keywords are involved	Depends on standardized text encoding
Probabilistic token validation	Finds probable keywords	Usually asserts correctly when keywords are involved	Depends on standardized text encoding

Table 2: Summary of Prototype Tools

3 Conclusion

This paper describes the file format identification problem, its context, existing identification techniques, tools that use those techniques, and our research prototype tools. No single tool or collection of tools will ever be able to identify every file that a forensic examiner or search engine indexer may encounter. However, these prototype tools do illustrate the value of several techniques that enable identification of many files that would otherwise go unidentified and unexploited.

4 Bibliography

thestar.com. http://www.thestar.com/news/gta/article/716761--libraries-computers-seized-in-mariam-case, retrieved on December 1, 2009.

boston.com. http://www.boston.com/news/local/breaking_news/2009/04/official_invest.html, retrieved on December 1, 2009.

libmagic. I.F. Darwin, ftp://ftp.astron.com/pub/file, 2008.

Outside In File ID. http://www.oracle.com/us/technologies/embedded/025613.htm, retrieved on December 1, 2009.

DROID. http://www.nationalarchives.gov.uk/aboutapps/pronom, retrieved on January 4, 2010.

TrID. Marco Pontello, http://mark0.net/soft-trid-e.html, retrieved on December 1, 2009.

mime-util. http://www.medsea.eu/mime-util, retrieved on January 4, 2010.

Mason McDaniel. An Algorithm for Content-Based Automated File Type Recognition, 2001, Master Thesis submitted to James Madison University.