Utilizing Entropy to Identify Undetected Malware

Tom Davis, Product Manager, Cybersecurity Solutions

Guidance® SOFTWARE
The World Leader in Digital Investigations™

1. INTRODUCTION

This paper provides a methodology to create a more secure environment against the various forms of malware that can infiltrate computer networks. This type of malware is computer code that has the ability to change itself in order to evade traditional detection methods. Similarly, cyber attackers often make manual changes to their malware code for the same purpose. Given most malware detection engines use some form of signature comparison to identify malware, these on-the-fly and manual versioning changes make detection extremely problematic, if not nearly impossible, on a consistent and reliable basis. Because traditional cybersecurity methods are often time and labor intensive, this paper focuses on the use of entropy as a means of enabling more rapid incident response.

With the increasing use of the Internet as a delivery mechanism of polymorphic malware, identifying authorship of a malicious file, as well as ascertaining the propagation routes of constantly changed code, is extremely challenging.

Polymorphic malware changes itself in seemingly random ways to avoid detection by traditional signature-based detection routines. Attempts to address this issue, using available technology, have not been very successful on the whole. The right mix of process and technology automation has not yet completely solved the problem of determining with a high degree of assurance that one file is similar enough to another to be a version of the original. Also, with the aid of certain databases, that the morphed malware was originated by a given hacker.

A more accurate and reliable detection method, particularly when paired with the need to attribute subsequent versions (morphed copies of an original) is with the use of Guidance Software's Entropy Near-Match Analyzer feature and its application of entropy value comparison. This approach has significant advantages over traditional signature detection paradigms due to the entropy characteristics of these morphing files.

2. NEAR-MATCH METHODS

The Information Security industry has spent a significant level of effort in solving the problems with polymorphism in malicious code and has largely been unsuccessful for a number of reasons.

2.1 Diff Analysis

Early attempts at solving the problem of identifying iterations of polymorphic malware involved manually comparing files to identify similarities in their content. The process can be automated somewhat by using "diff". Diff is a file comparison utility that outputs the differences between two files. It is typically used to show the changes between a file and a former version of the same file. Diff displays the changes made per line for text files.[1]

[1] Website: http://en.wikipedia.org/wiki/Diff, last modified on 12 October 2009

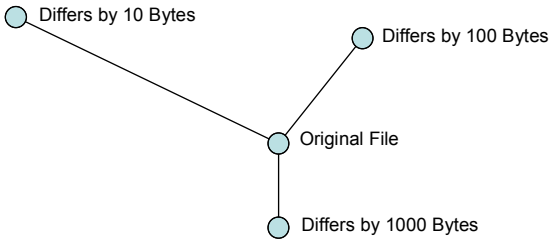
While visually (or virtually) comparing the contents of one file to another is exceptionally accurate, the process is extremely labor intensive and does not scale to enterprise operations.

2.2 Hashing

The utility of hash values in computer forensic applications is well known. Hash algorithms take an unlimited number of bytes from a file and produce a fixed size number, called a hash value. If even a single byte of a file changes, the value of the hash changes. Whether the algorithm is MD5, SHA1 or SHA256, hashing algorithms are designed to change about 50% of their digits in response to every byte from a file. By design, there is no relationship between the contents of the file and its respective hash value. The hash value itself has no "meaning" and one cannot infer properties of the file from the value of its hash.

For this and other reasons, it is nearly impossible to change the contents of a file such that it has a specific hash value. This means that if the hash values of two files are the same, it is almost certain that the contents of both files are the same as well. The odds of accidentally finding two files with different contents that have the same hash value, is proportional to about half of the key space of the hash.

It is common in computer forensics to gather large numbers of file hash values into hash sets. These sets can be used to quickly identify files, without having to do a byte-by-byte comparison with the original file contents. To better understand this, consider this figure illustrating possible hash values as points on a grid:



The lines represent the numeric distance between the hash values of four files. For the reasons described earlier, a completely different file could end up having a hash value that is closer numerically to the original, than a file with only a single byte of difference. This is by design, and it is the principal strength of the hash algorithm. However, if one is looking for a tool to find near-misses, one must look elsewhere.

2.3 Fuzzy Hashing

Sometimes referred to as Context Triggered Piecewise Hashing (CTPH), fuzzy hashing describes a process-based application of traditional hashing. The process involves parsing a file into a number of smaller parts and hashing these individually. The premise is that a

person could potentially use these multiple hash values to ascertain some likelihood that it is similar to the hashed part(s) of other files. This likelihood is usually expressed as a percentage match.

CTPH can match inputs that have homologies[2]. Such inputs have sequences of identical bytes in the same order, although bytes in between these sequences may be different in both content and length.

This fuzzy hashing process is a way to perform attribution of malware against a *static* repository of previously analyzed and attributed code. For instance, if during forensic and subsequent intelligence activities, an agency was able to pinpoint the nexus of a piece of malware and determine that a specific hacker or group was responsible for creating that code, then it stands to reason that any future code discovered on a network that shared that same similar signature could be attributed to a source (author).

CTPH, however, is fairly process intense in that it must first perform a diff analysis on a set of files, parse out those elements that are alike in each, and then hash these alike pieces to indicate a potential match. Though conceptually sound, this method brings with it a host of functional issues beyond the time and overhead requirements: text formatting may logically change the content of a piece enough that it will be parsed out; large files, and especially binary code, may contain significant amounts of "canned" content, and yet still have no meaningful relationship to the pieces of other files.

Fuzzy hashing is not a particularly complex or labor intensive science to provide an initial indication of content similarity that will require further analysis. This capability originated as a freeware tool named "ssdeep" released in 2006 by Jesse Kornblum at Mantech, and was a derivative of a tool to detect spam called spamsum. The ssdeep tool is used by military and intelligence cyber defense teams to perform attribution. While helpful, the limitations are many. Chief among them is that it is strictly limited to offline analysis. Further, the user must possess both the source and target files on an offline repository. The process is also computationally intense and time consuming. In its current state, cyber defense teams are unable to perform scans against large source repositories, as the tool is not suited for such a task. As this process is static in nature, the teams are limited to performing analysis on drives sent to them from the field, severely limiting both the speed and proper analysis of target drives.

2.4 Entropy

The origin of the concept of entropy comes from the science of thermodynamics. Entropy is a measure of the amount of disorder in a closed system. For instance, an ice cube is an ordered array of molecules and has relatively low entropy. As the ice melts, there is a higher degree of freedom amongst the water molecules, and therefore greater entropy. As the water evaporates, the molecules are free to move in the air, and the entropy is still greater.

Claude Shannon borrowed the concept of entropy to describe the amount of randomness in an information stream in a 1948 paper called "A Mathematical Theory of Communication[3]".

[2] The quality of being similar or corresponding in position or value or structure or function. Website, http://wordnetweb.princeton.edu

Shannon was interested in communications channels, but his results are applicable to computer files.

The key difference between the thermodynamic entropy theory and information entropy theory is that, in thermodynamics, you cannot know all the possible states so statistical methods are used as an approximation. In digital information entropy theory, the number and probability of each state is known precisely, since the exact content of the file is known.

In information analysis, we are concerned with bytes of data (each having 256 possible values), and we would like our result expressed in bits per byte. Therefore the value of the entropy of a given file will come out to be a value between 0 and 8, with the value extremes expressed as:

0.00000000 = Orderly 8.00000000 = Random

3. GUIDANCE SOFTWARE'S ENTROPY NEAR-MATCH ANALYZER

Guidance Software's Entropy Near-Match Analyzer feature provides the capability to perform near real-time attribution of the files present on a computer anywhere it resides in a networked environment. Entropy Near-Match Analyzer enables the user to calculate entropy values remotely, without the need to be connected to a source repository, and includes a proprietary process (patent pending) that does not have the time overhead of other tools. While other tools have a fuzzy hashing capability, most are a derivative of Jesse Kornblum's ssdeep, which is solely reliant on CTPH.

Guidance Software's approach is to analyze the desired end state, and perform a byte transition analysis that yields exponentially quicker results with far better accuracy in detecting malware.

Any communication, regardless of its language, will simultaneously have both order and random distribution. Communication consists of many characters that are formed into words using a well established convention (spelling). These words are further grouped into sentences (grammar, syntax, etc) and then into paragraphs, chapters and books. There is a well defined and recognizable order to all communications.

There is also a natural and unique randomness to communications, particularly when compared to the randomness of other unrelated communications. For example, consider the following set of characters:

aaaaa aaaaa

In this example, there is no randomness to the distribution of the letter "a". It is an ordered set. The entropy value (the randomness of the distribution of "a" within this set of letters) is zero – it has no entropy.

Now consider this set of characters:

[3] C. E. Shannon, ``A mathematical theory of communication," *Bell System Technical Journal,* vol. 27, pp. 379-423 and 623-656, July and October, 1948.

afkwe aewfk

This example, represents a random string of characters. While this is true, it's not quite as random as one would think. Since we are concerned with the randomness of distribution, we can calculate a relatively low entropy value.

The likelihood that the letter a will be present anywhere in the string is 2 in 10. The same is true with the letter f and all the rest. The distribution then of each of the characters is consistent throughout the string of characters and has little entropy.

Finally, consider the entirety of text in this document. There are hundreds of characters grouped together in a very ordered way. One would say this paper's entropy value should be at or near zero since it does not appear very random. However, the *distribution* of each character within all the rest does have natural entropy. It is different from the natural entropy of say, the Declaration of Independence. Each set of characters is formed differently (different words using the same character set) in each document, even though they both follow the same language conventions.

This calculation of the entropy of characters is called z order entropy. Another method is called byte transition or first order entropy. Rather than determining the entropy of single characters within the whole, we are calculating the randomness of the distribution of the transition of character pairs within the whole. Take the following example:

Now is the time for all good men to come to the aid of their country.

The likelihood that "n" will follow "o" elsewhere in the string of letters is 0 (it occurs only once), where the likelihood that the letter "e" will follow the letter "m" is 1, as it occurs twice in the string.

Of particular note is the fact that calculating the entropy of a set of bytes is wholly independent of either what the set is or what it may represent. The calculation does not recognize that this string is a sentence or that it has some meaning when read. It simply calculates the likelihood that characters, represented as bytes, will occur in the string at any frequency.

Why is this important when comparing two files for similarity? Principally, any structured set of characters (or byte values) has a generally ordered pattern of byte transitions following the rules of language. Letters (bytes) will have a predictable distribution and frequency when the string is meant to communicate something and the distributions and frequency are similar. From this we intuit that any two like files will have similar entropy values. The more alike they are, the closer those entropy values will be.

3.1 File Types

As a general rule, specific file types will have entropy values within smaller value bands between 0 and 8. For example, an ASCII text file will usually have an entropy value between 2 and 4, where zip file archives will generally have a value between 7 and 8. All file types generally follow this banding rule. Since the aim of searching for modified versions of a file (polymorphic malware, for example) is to identify potential versions, comparing a binary

executable to a PDF document would only confirm that the two files are not similar – their respective entropy values are expected to be significantly different.

The Entropy Near-Match Analyzer takes this value banding into consideration by default to help focus the search for potentially similar files.

3.2 File Size Change

Files of a specific type generally have entropy values within bands along the spectrum of possible values. Because of the nature of entropy, however, it is possible for two completely unrelated files to have similar entropy. This is not a failing in the way entropy calculation works. To counter this, the file size differences are also weighed when calculating the "likeness" values.

In the context of polymorphic malware, file size changes between iterations of the original (apex) executable will usually be small, on the order of bytes on average. The smaller the change, the more likely the compared files will be similar.

To illustrate how file change gives weight to Entropy Near-Match Analysis, consider the following notional executables:

Control---------	FileA.exe	1,024 bytes
	FileB.exe	1,048 bytes
	FileC.exe	1,240 bytes
	FileD.exe	102,400 bytes

In this example list, FileB.exe is larger than FileA.exe by 24 bytes. If expressed as a percentage of change, FileB.exe is 2.3% larger than FileA.exe.

FileC.exe is 216 bytes larger than the control or 17.5% larger than FileA.exe. There is a slightly more significant change, lessening the confidence weight that this file is similar to the control file.

FileD.exe is 100 times larger than FileA.exe – off the scale as a percentage of difference. This drastic difference in size from the control file suggests little, if any, similarity. In the context of searching for potential malware, it has little weight.

In the context of polymorphic malware, the size difference between FileA and FileB is a reasonable expectation. The same is true of FileC to a somewhat lesser degree. It would also be a safe assumption that FileD has no reasonable similarity to FileA based on the extreme size difference.

When assigning weight to the closeness of two entropy values, this closeness in file sizes lends another measure of credibility to the applications claim of likeness.

4. BRINGING IT ALL TOGETHER

The end product of Entropy Near-Match Analyzer is a single number: the “likeness” value. This number is between 0 (no similarity) and 100 (exact match) and is intended to assign a level of confidence in the assertion that files are similar. It is computed by analyzing:

- The difference in the compared files entropy values – Entropy Delta
- The type of files being analyzed
- The differences in file sizes – Size Delta

To test this analysis process, a number of versions of an executable file were analyzed in the Guidance lab to determine their likeness to the original (control) file. The actual percentage of difference between the versions was known. When the entropy values were calculated on these executables, they were strikingly similar.

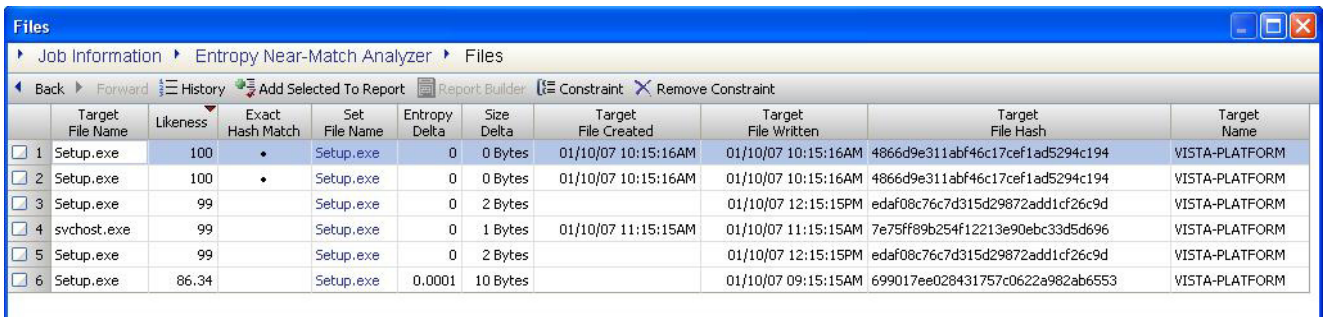
When the size differences were applied to the likeness weight, a more accurate likeness value was computed, more closely matching the actual differences in these files.

A test to ascertain the accuracy of Entropy Near-Match Analyzer in the detection of morphed and propagated malware on computers systems is shown below.

This test involved an intentional infection of a Windows Vista 32 Bit Virtual Machine with Backdoor.IRC.Bot. This is a generic detection for a group of backdoor Trojans which use IRC channels to launch Denial of Service attacks. The sample was obtained from a download using a popular P2P application and then executed on the VM.

In this report, the file in line 1 is the “control” sample – the executable that was originally executed on the VM and archived onto separate removable media. By first comparing the hash values, we are assured that the file in line two is the exact same file – the file we intentionally dropped on the VM. This is confirmed by the, matching hash, zero value entropy and file size deltas, and by the matching created and written dates.

The remaining four files in this report were copies created by the Trojan. Of particular note is line four, where we confirmed that svchost.exe had also been infected by the Trojan.



	Target File Name	Likeness	Exact Hash Match	Set File Name	Entropy Delta	Size Delta	Target File Created	Target File Written	Target File Hash	Target Name
1	Setup.exe	100	•	Setup.exe	0	0 Bytes	01/10/07 10:15:16AM	01/10/07 10:15:16AM	4866d9e311abf46c17cef1ad5294c194	VISTA-PLATFORM
2	Setup.exe	100	•	Setup.exe	0	0 Bytes	01/10/07 10:15:16AM	01/10/07 10:15:16AM	4866d9e311abf46c17cef1ad5294c194	VISTA-PLATFORM
3	Setup.exe	99		Setup.exe	0	2 Bytes		01/10/07 12:15:15PM	edaf08c76c7d315d29872add1cf26c9d	VISTA-PLATFORM
4	svchost.exe	99		Setup.exe	0	1 Bytes	01/10/07 11:15:15AM	01/10/07 11:15:15AM	7e75ff89b254f12213e90ebc33d5d696	VISTA-PLATFORM
5	Setup.exe	99		Setup.exe	0	2 Bytes		01/10/07 12:15:15PM	edaf08c76c7d315d29872add1cf26c9d	VISTA-PLATFORM
6	Setup.exe	86.34		Setup.exe	0.0001	10 Bytes		01/10/07 09:15:15AM	699017ee028431757c0622a982ab6553	VISTA-PLATFORM

Entropy Near-Match Analyzer’s utility is not limited to malware detection. When entropy values are correlated with other metadata, an analyst is able to reliably determine the progression of versioning of malware across time, potentially identifying the “Zero Patient” – i.e., the original first iteration of the attack.

Entropy Near-Match Analyzer also proves to be valuable in the search for misappropriated or stolen Intellectual Property (IP) in that text from an original document, even if changed in a number of ways, will retain much of its natural entropy value in potential copies.

5. CONCLUSION

There is great value in unambiguous detection of specific files through the use of a conventional hash value. Yet organizations can significantly reduce risk and increase operational integrity when they are able to quickly and accurately find files that are similar to the files in a set, but not identical. For instance:

- Polymorphic malware - The executable malware “mutates” itself slightly as it spreads throughout the network, in order to defeat hash-based detection schemes. Every copy of the file on the network has a different hash value, making detection and cataloging difficult.
- Different executables with the same code - Executables that have the same source code, but are compiled with different settings, or with a different version number, will have distinct hash values.
- Near Matches - Documents that have been changed slightly will have a completely different hash value. If you have a copy of a document, simply opening the document and saving it again, without making any changes to the text, is usually enough to change the hash value of the document, due to the changing values of the embedded metadata.
- Email Forwarding - Email software often concatenates “quoting” sequences to an email body when you reply or forward the email. Although the text is “essentially the same,” those additional quoting characters will change the hash of the text, making it tough to identify in an automated fashion.

Despite the many uses for the classic hash value, there are many situations where its “all or nothing” characteristic makes it unsuitable. In these situations, alternative tools and techniques for file attribution yield better results. A combination of character masks, repetition constraints, differential filters, and entropy order can be brought together to yield a numeric measure of a file that scales in proportion to the frequency of essential elements in the file’s contents. Furthermore, the numeric difference between the values of files in the same category yields a value that is in proportion to the difference between the two files. Entropy value calculation and comparison is a powerful technique for file attribution in computer forensics. It offers a unique capability to identify and remediate malware to increase the security of an enterprise network.