The Study of Evasion of Packed PE from Static Detection

Mirza Baig, Pavol Zavarsky, Ron Ruhl, Dale Lindskog
Information Systems Security Management
Concordia University College of Alberta
Edmonton, Canada
khurram_azeem@hotmail.com, {pavol.zavarsky, ron.ruhl, dale.lindskog}@concordia.ab.ca

Abstract—**Static detection of packed portable executables (PEs) relies primarily on structural properties of the packed PE, and also on anomalies in the packed PE caused by packing tools. This paper outlines weaknesses in this method of detection. We show that these structural properties and anomalies are contingent features of the executable, and can be more or less easily modified to evade static detection.**

Keywords- obfuscation, static detection, import address table, entropy

I. INTRODUCTION

Packing, as originally conceived, is a method used for such purpose as compression, minimizing the number of files associated with an application, or in order to protect intellectual property against reverse engineering and all in such a way that the packed executable can be used exactly the same way as the original executable [2]. Packing Windows portable executables, although potentially legitimate, is often used as an obfuscation technique, in which a malicious file is inserted or 'packed' into another executable in order to bypass signature-based detection. Packing an executable for the purpose of obfuscation is not new but is still popular. Packing changes the file significantly, and can therefore be used to bypass signature-based detection.

Static investigation of executable files relies on features of the stored file, and is one of the methods used to detect malicious executables. Obfuscation is one of the techniques used to evade static detection, through the concealment of the intended meaning of the code, making it confusing, intentionally ambiguous and more difficult to detect [1]. Given that packing is often used for the purpose of obfuscating malicious executables, static detection of malicious files often depends upon the ability to detect that a file is packed executable. Static detection of packed PEs is normally signature-based or statistical. *PEiD* is a signature-based detection tool used to detect packed executables, and as of version 0.95 contains signatures associated with over 600 different packers and compilers. This approach to static detection of packed PEs relies on an examination of certain structural properties of the packed PE, such as its section access rights or the names of its section headers. However, just as packing itself can be used for the purpose of obfuscation, one must ask the question whether packed PEs themselves can be modified sufficiently to evade their identification as packed. In this paper we examine these structural properties, show that in each case we examined the properties are merely contingent features of the executable file, and thus can be modified while maintaining the file's function as a packed executable. Typical PE file structure of a normal executable is shown in Fig-I.

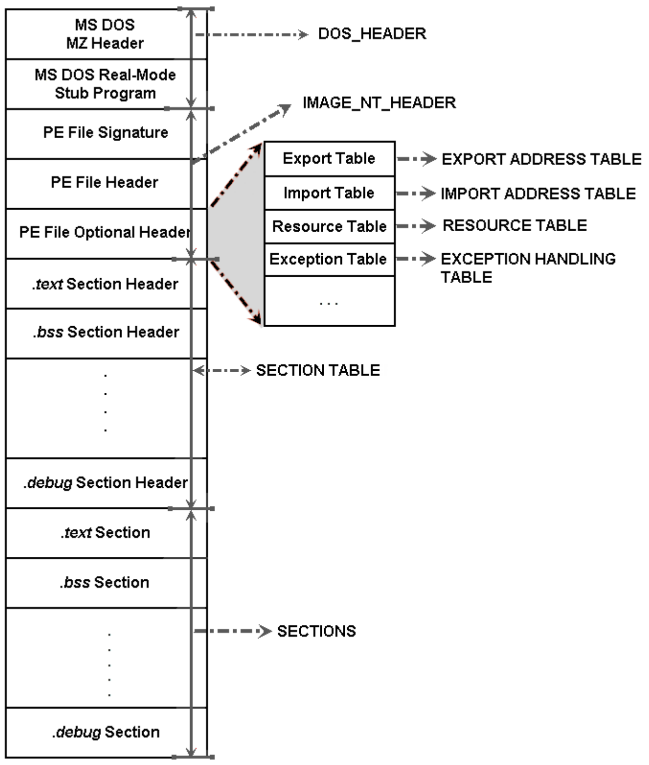


Figure 1. PE structure of normal executable

The structural properties of packed PEs that aid in identification are: (i) the number of import function calls, (ii) entropy values, (iii) section names, (iv) number of sections, and (v) access rights on each section. The number of import function calls in a portable PE is relevant because packed PEs tend to have fewer import function calls than unpacked PEs. Entropy values are also relevant, as they tend to be higher than 6.98 out of 8. Sections names assist in identification because of packed PEs because packers use non-standard section names, while unpacked PEs have standard section names. Normal PEs have predefined access rights, and there are sections that have read, write and execute permissions, but unlike a packed PE, no sections in a normal PE has all three (read, write and execute) permissions set for a single section [3]. In most packed PE, there is at least one section that has read, write and execute permissions. This is because the section that contains the unpacking routine has to be executed to unpack the compressed PE and thus the permissions on this section must be set to executable, readable. Moreover, most of the packers update and decrypt themselves, and thus require write access.

This paper examines all of the above structural properties of packed PEs. Methods of altering these properties for the purpose of evasion are investigated, and proofs or illustrations of the methods of obfuscation are provided. We suggest that, by analyzing these methods of obfuscation, we will assist developers of static detectors to increase the robustness and reliability of detection of packed PEs.

This paper is organized as follows: Section II, part A presents a literature review of static detection of packed PE; section II, part B describes the detection criteria used for static detection of packed PEs; section III presents our analysis of these packed PE properties, and how they can be modified to evade detection; section IV recommendations; section V concludes the paper.

II. Literature Review

This section is an overview of previous research into static detection of packed executables.

A. Detection Overview

Scott [4] presented a heuristic method for detection of windows based obfuscated malware. Scott focused on non-standard section names, flagged section names, the fact that an entry point resides outside the code section, thread local storage section checks, DLLs with no imports, low numbers of import calls and import address table checks. Research by Wei [5] focused on the mechanism of packing an executable file. Wei focused on the unpacking mechanism by addressing the properties of packed executables used in the static detection. Igor [6] proposed collective-learning-based packed executable detection system that based upon structural features and heuristics is able to determine when an executable is packed. Lyda [7] focused on entropy analysis of packed executables. Zubair [8] proposed an accurate malicious executable detection technique called PE-Probe, which has the ability to detect packed files and uses structural information about portable executables to detect malicious executables. Seungwon [9] provided a detection method using two characteristics of packed files; entropy of the entry point section and access-controls. Choi [10] proposed a detection technique based on a PE header analysis (PHAD).

The above mentioned researches are mainly focused on static detection of packed PE and structural properties of PE used in static detection of packed PE. Former researches did not focus on the aspect of evading packed PE from static detection by modifying the detectable structural properties, from static detection, of PE by packers during PE packing.

B. Review of Detection Criteria

Static detection of packed PEs is based on the structural properties of PE format. Properties of PEs relevant to their identification as packed are discussed in detail in the following sub-sections.

1) Import Address Table

The Import Address Table (IAT) consists of addresses of external functions that the PE imports from the Windows API as DLLs are loaded. Packers hide the original PE's IAT by compressing it, and only un-packer section is detectable. As a result, the number of IAT entries becomes significant small. An abnormally low number of import function calls in a PE indicate that the executable may contain a compressed executable. Zubair noted that packed PEs have a significantly lower number of import function calls than a normal PE, and used PE-Probe to detect packed PEs on the basis of this low number, and other characteristics.

2) Section Access Rights

There are three types of access rights in every section of a PE: read 'R', write 'W' and execute 'X' permissions. The standard access rights on each section of a PE are shown in the Table-I. Normal PEs have only one section with 'R' and 'X' permissions, while packed PEs have two sections with 'R', 'W' and 'X' permissions (see Appendix A, Fig-1 and fig-2 for details).

TABLE I STANDARD ACCESS RIGHTS ON SECTIONS OF PE

Section Name	Access Rights
.text	Read and Execute
.rdata	Read-Only
.data	Read and Write
.bss	Read and Write
.rsrc	Read Only
.reloc	Read Only

Seungwon indicated that the packed PE file is executable only when including WRITE property in section characteristics is the essential element for execution.

3) Entropy Value

A packed executable stores its data with compression and sometimes encryption also. Therefore, the packed data stored in a particular section is more or less random. Randomness is the information density of the contents of the file, expressed as a number of bits per character [4]. Legitimate software vendors use packers to protect and manage their software. Detecting packed PEs without identifying the compression algorithm is insufficient because you cannot un-compress and inspect the file contents. The reason to calculate entropy is that packed PEs have higher entropy values normal PEs. Lyda computed the entropy of a discrete random event *x* using the following formula [11]:

$$H(x) = -\sum_{i=1}^{n} p(i) \log_2 p(i) \ ,$$

Where *p(i)* is the probability of the i^{th} unit of information (such as a number) in event *x*'s series of *n* symbols. This formula generates entropy scores as real numbers; when there are 256 possibilities, they are bounded within the range of 0 to 8.

Lyda's dataset were 100 native PEs from Windows XP SP2 and Seungwon's dataset were of 200 native PEs. Choi used entropy analysis to analyze eight characteristics of portable executable (PE) binary files. *PE-Probe* is another entropy-analysis-based detector. Its detection rates in tests were 99.6 percent for packed files and 99.4 percent for non-packed files, with false-positive detection rates of 0.3 percent and 0.8 percent, respectively [9].

Sets of 200 PEs and 200 packed PEs were taken from Windows Server 2003 and Windows 7 respectively, to examine entropy values. Table-II shows that the average entropy value of a normal PE is 5.89, while the average entropy value after packing the same PE becomes 7.7. Benign PEs are used in packing PE therefore entropy range detected by Seungwon is different from the entropy range in Table-II. It is also possible to calculate entropy on specific sections, and again there are significant differences (see Table A from Appendix B).

TABLE II ENTROPY VALUES OF PE AND PACKED PE

S.No	PE Names	Entropy Range	Average Entropy Percentage % (0 to 1)	Average Entropy Value (0 to 8)
1	Benign PE	3.60 - 6.65	68.62%	5.89
2	Packed PE *(Compressed)*	7.07 - 7.99	96.32%	7.7

4) *Section Characteristics*

Microsoft has published [12] a set of standards and baselines for the creation of PE file formats for the Windows operating systems. Standard section names, such as *.bss*, *.data*, *.debug*, *.text*, *.rsrc*, *.reloc*, *.idata* and *rdata,* are used in PEs. Every section has specific tasks: the *.bss* section holds uninitialized data, the *.data* section hold initialized data, the *.debug* section holds information for symbolic debugging, the *.text* section holds the ''text'' or executable instructions, the *.rsrc* section contains all the resources for the module, the *.reloc* section holds a table of base relocations, the *.idata* section contains information about functions that the module imports from other DLLs (dynamic link libraries), and the *.rdata* section holds the debug directory and the description string [12].

Scott proposed relying on packed PEs non-standard section names criterion for detection. Scott also proposed checking the section's against their documented characteristics to ensure that the malware writers have not modified standardized sections for other purposes.

5) *Address of Original Entry Point (OEP)*

In a normal PE, the address of EP usually points to the *.text* section or *.data* section, and within the code area as defined in portable executable (PE) header by Microsoft [12]. For a packed executable, the address of the EP depends on the type of packer, but packers often pack the original PE in the *.text section* (although they can use other sections to store the compressed PE). Regardless, the important element to consider is the address of the original entry point (OEP), because the original entry point of the packed PE resides in the section pointed by the entry point (EP). As noted by Scott, one of the main methods to evade detection is to obfuscate the entry point of the packed PE. In order to hide malicious intent, malware authors modify the entry point to the non-code section and the EP points outside of the .text section – marked as code [4].

6) *Header Overlapping*

PEs may have a PE/COFF header offset that overlaps the MS DOS stub executable that is present at the start of the file. Compilers do not typically produce PE/COFF files with this characteristic, and the presence of this often indicates an attempt to evade security software [13]. Once the header overlapping is detected, executables are marked as packed and therefore suspicious. The header overlapping technique is well-known [14], and relied upon by detectors such as *Mandiant Red Curtain* (see Fig-3 of Appendix A).

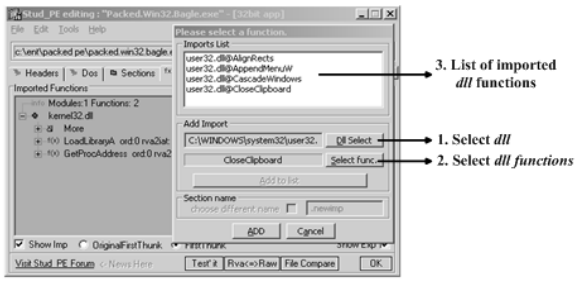
III. ANALYSIS OF PACKED PE PROPERTIES TO EVADE STATIC DETECTION

The research is focused on these properties of packed PEs is an important contribution to static detection, but in this section we focus on whether these same properties, intentionally modified, might allow for the evasion of packed PE detection.

A. *Import Address Table (IAT)*

Recall Zubair`s observation that packed PEs have lower numbers of import function calls than native PEs (see Fig-4 and fig-5 of Appendix A. Usually, packed PEs import functions from *kernel32.dll*, especially *LoadLibrary*, *GetProcAddress*, *VirtualAlloc*, and *VirtualFree.* There are various methods to increase the number of import functions calls. One method is to add innocent functions calls, which have nothing to do with the packing and unpacking routines, but rather simply populate the IAT in order to evade detection. This method could be incorporated into packers, or one might use a PE editor to populate the IAT of the packed PE, by editing the header information.

This method can be demonstrated with *Stud_PE v2.6.0.6* (a portable executable viewer/ editor) to add a dynamic link library (*dll*) with specific functions. Fig-II illustrates the method by adding *User32.dll* to the IAT.


Stud_PE editing : "Packed.Win32.Bagle.exe" - [32bit app]

c:\ent\packed pe\packed.win32.bagle.

Imported Functions
- Modules:1 Functions: 2
- kernel32.dll
  - More
  - LoadLibraryA ord:0 rva2iat:
  - GetProcAddress ord:0 rva2

Please select a function.

Imports List
- user32.dll@AlignRects
- user32.dll@AppendMenuW
- user32.dll@CascadeWindows
- user32.dll@CloseClipboard

Add Import: C:\WINDOWS\system32\user32.
CloseClipboard

Section name: choose different name [ ] .newimp

[✓ Show Imp] OriginalFirstThunk FirstThunk

File Edit Tools Help
Headers Dos Sections
Dll Select
Select func.
Add to list
ADD Cancel
Show Exp
Visit Stud_PE Forum <- News Here
Test' it Rva<=>Raw File Compare OK
3. List of imported *dll* functions
1. Select *dll*
2. Select *dll functions*

Figure 2. Add functions in IAT using Stud_PE

B. Section Access Rights

Microsoft defined standard access rights on each section of the PE [12]. Seungwon noted that, for packed PEs, WRITE access is essential for execution. Packed PEs have all three ('R', 'W', 'X') access rights in at least one section, and that section has the responsibility to uncompress and run the compressed executable (See Fig-6 in Appendix A.)

It is not trivial to eliminate these differences in access rights between packed PEs and normal PEs. One method is to introduce code obfuscation into the packed PE, by dividing the three (read 'R', write 'W' and execute 'X') access rights over multiple sections. For example, imagine a packed PE that has three sections: section 'A', section 'B' and section 'C'. Section'A' has read and write access while section 'B' has read and executes access. When the packed PE executes, code on section 'A' and code on section 'B' both run, and establish all three ('R', 'W', 'X') access rights onto section 'C', as shown below in Fig-III. Section C can then be used write the uncompressed original PE, and then run that executable.

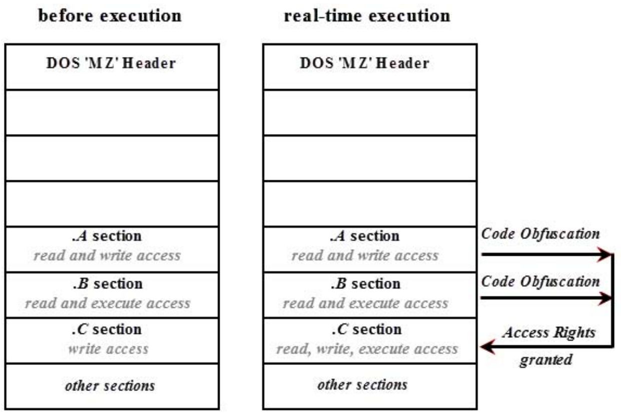


Figure 3. Code obfuscation used to hide access rights in packed PE

Another method to obfuscate section access rights is to set PAGE_NO_ACCESS rights [15] in the table header information. PAGE_NO_ACCESS allows no access to the specific memory page. Any attempt to read, write or execute the memory page with NO_PAGE_ACCESS rights results in an access violation. By setting PAGE_NO_ACCESS rights in the memory area where the table header information resides, processes will fail to read the table header information and thus evade this detection method.

C. Entropy Value

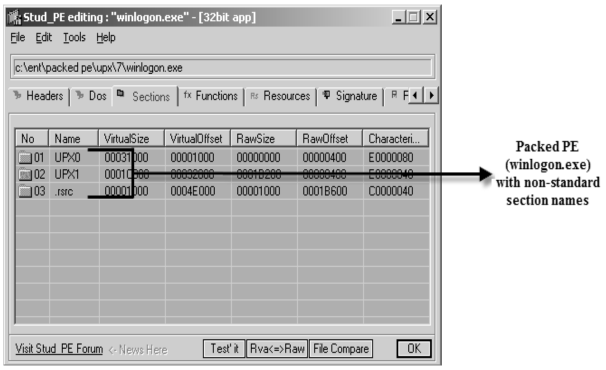
Entropy is one characteristic that is detectable without in-depth analysis. PEs have a stable histogram as compared to packed PEs (as generated by *Ent* - Entropy Level and FPU density measurement tool), see Fig-7 and fig-8 of Appendix A for more detail. Seungwon detected the average entropy values of PE and packed PE (sample taken from honeypot) are 5.86 and 7.33 respectively.

It is a trivial matter to lower the entropy of a packed PE, and there are different methods available. One method is to use the *truncated binary algorithm, entropy* encoding typically used for uniform probability distribution with finite alphabets [16]. Run-length encoding is another method that replaces a sequence of the same code by the number of occurrences [17]. Normalized Shannon entropy [18] and Minimum Entropy De-convolution (MED) [19] are also the methods available to lower entropy values.

D. Section Characteristics

Microsoft standardized section names for PEs [12], but packers often use predefined section names, and it is evident (see Fig-2 of Appendix A) that most packers use non-standard section names, by adding sections or by replacing sections with the packer's predefined sections. One interesting point is that some packers use empty section names (see Fig-9 of Appendix A) or repeat the same section name in different sections, and this is easily discovered by examining the section table of the packed PE. Every packer has its own pre-defined number of sections and the specifics vary from one packing tool to another. Section names are a trivial property of a PE, and packers could easily be modified to set them to standard section names during packing, or alternatively, one could rename them to standard section names using different PE editing tools. We use the *Stud_PE v2.6.0.6* tool to illustrate this: we are able to add new sections with standard names and also able to rename non-standard section names to standard section names, simply by editing the packed PE header information.

Fig-IV shows the packed PE (*winlogon.exe*) sections information in which non-standard names (*UPX0, UPX1*) are used.


Stud_PE editing : "winlogon.exe" - [32bit app]

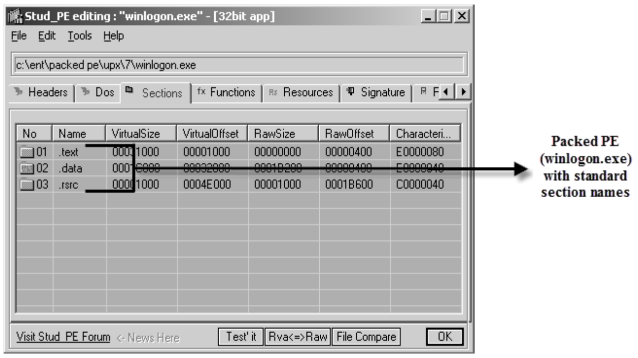
c:\ent\packed pe\upx\7\winlogon.exe

No	Name	VirtualSize	VirtualOffset	RawSize	RawOffset	Characteri...
01	UPX0	00031000	00001000	00000000	00000400	E0000080
02	UPX1	0001C000	00032000	0001B200	00000400	E0000040
03	.rsrc	00001000	0004E000	00001000	0001B600	C0000040

File Edit Tools Help
Headers Dos Sections Functions Resources Signature
Visit Stud_PE Forum <- News Here
Test' it Rva<=>Raw File Compare OK
Packed PE (winlogon.exe) with non-standard section names

Figure 4. Packed PE (winlogon.exe) with non-standard section names

Fig-V shows the same packed PE (*winlogon.exe*) sections information in which non-standard section names (*UPX0, UPX1*) are renamed with standard section names (*.text, .data*) by editing the packed PE header information.


Stud_PE editing : "winlogon.exe" - [32bit app]
File Edit Tools Help
c:\ent\packed pe\upx\7\winlogon.exe
Headers Dos Sections Functions Resources Signature F
Visit Stud_PE Forum <- News Here Test' it Rva<=>Raw File Compare OK
Packed PE (winlogon.exe) with standard section names

No	Name	VirtualSize	VirtualOffset	RawSize	RawOffset	Characteri...
01	.text	00031000	00001000	00000000	00000400	E0000080
02	.data	0001C000	00032000	0001D200	00000400	E0000040
03	.rsrc	00001000	0004E000	00001000	0001B600	C0000040

Figure 5. Packed PE (winlogon.exe) with standard section names

E. *Address of Original Entry Point (OEP)*

There are two points need to consider regarding the OEP address. First, as *Wei Yan* noted, if Windows API functions are found, which are not usually called by packers (such as *CreateWindowA*) then searching for the OEP should be stopped and the executable can be marked as benign. Second, the entry point should point into the *.text* or *.data* sections, and must be within the standards as described by Microsoft.

Malware authors might evade detection by using those Windows API functions which are not usually used by packers, and ensure the EP resides inside the *.text* or *.data* sections.

F. *Header Overlapping*

Microsoft avoids non-overlapping memory addresses to ensure high performance of the process associated with the PE. Packed PEs do not follow these specifications recommended by Microsoft, and hence anomalies are present in packed PE. Header overlapping is used to hide the malicious intent in packed PE, but are detectable, e.g. by Mandiant's Red Curtain and other tools.

Packers or malware authors can avoid header overlapping and simply follow the specifications recommended by Microsoft.

IV. RECOMMENDATIONS

Detection criteria include import address table, section access rights, entropy values, section characteristics, original entry point and header overlapping as discussed in analysis section. These structural properties are deviated from the standard defined by Microsoft for PE. Each criterion cannot be taken separately; proposed analysis of selected properties should be taken as a group in order to evade the static detection of packed PE. Static detection is found vulnerable in detection criteria and cannot be fully trusted until and unless the criteria of static detection are improved.

V. CONCLUSION

It is evident from our analysis of selected properties of packed PEs that static detection has definite flaws, and evasion is possible by modifying the properties of the packed PE. The study revealed serious flaws in the criteria used for static detection of packed PEs. Malware authors can exploit the contingent nature of these structural properties by modifying the packed PE as proposed to evade detection. *PE Explorer, IDA, Stud_PE* and other tools were used illustrate such possible modification of these properties of packed PEs. We conclude that static detection of packed PEs cannot be relied upon until new, more robust criteria are identified.

REFERENCES

[1] Branko Spasojević, "Using optimization algorith ms for malware deobfuscation", http://os2.zemris.fer.hr/ns/malware/2010_spasojevic/Diplomski_Spasojevic.pdf , June 2010, Zagreb, Croatia
(Access Date: 29th April, 2012)

[2] Malware Obfuscation Using Code Packing, http://www.foocodechu.com/?q=node/55, January 2012
(Access Date: 29th April, 2012)

[3] Matt Pietrek , "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format", http://msdn.microsoft.com/en-us/library/ms809762.aspx, March 1994
(Access Date: 29th April, 2012)

[4] Scott Treadwell, Mian Zhou, "A Heuristic Approach for Detection of Obfuscated Malware", Intelligence and Security Informatics, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arn umber=5137328, June 2009
(Access Date: 29th April, 2012)

[5] Wei Yan, Zheng Zhang and Nirwan Ansari, "Revealing Packed Malware", Security and Privacy, IEEE, http://ieeexplore. ieee.org/stamp/stamp.jsp?tp=&arnumber=4639028, October 2008
(Access Date: 29th April, 2012)

[6] Igor Santos, Xabier Ugarte-Pedrero, Borja Sanz, "Collective Classification for Packed Executable Identification", Perth, Western Australia, Australia, http://paginaspersonales.deusto.es/claorden/publications/2011/Santos_2011_CEAS_Collective%20Classification%20for%20Packed%20Executable%20Identification.pdf, September 2011
(Access Date: 29th April, 2012)

[7] Robert Lyda and James Hamrock, "Using Entrop y Analysis to Find Encrypted and Packed Malware", Security & Privacy, IEEE, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=414 0989, April 2007
(Access Date: 29th April, 2012)

[8] M. Zubair Shafiq, S. Momina Tabish, Muddassar Farooq, "PE-Probe: Leveraging Packer Detection an d Structural Information to Detect Malicious Portable Executables", Next Generation Intelligent Networks Research Center and National University of Computer & Emerging Sciences, http://nexginrc.org/nexginrcAdmin/PublicationsFiles/vb09-zubair.pdf, Islamabad, Pakistan, June 2009
(Access Date: 29th April, 2012)

[9] Han, Seungwon Lee, Keungi Lee, Sangjin, "Packed PE File Detection for Malware Forensics", Computer Science and its Applications, 2nd International Conference, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05404211, 12 Dec 2009
(Access Date: 29th April, 2012)

[10] Yang-seo Choi, Ik-kyun Kim, Jin-tae Oh, Jae-cheol Ryou, "PE File Header Analysis-Based Packed PE File Detection Technique (PHAD)", International Symposium on Computer Science and its Applications, October 2008, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04654055
(Access Date: 29th April, 2012)

[11] C.E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, http://www.press.uillinois.edu/books/catalog/67qhn3ym9780252725463.html
(Access Date: 29th April, 2012)

[12] “Microsoft PE and COFF Specification”, http://msdn.microsoft.com/library/windows/hardware/gg463125, Revision 8.2, September 2010

[13] IBM Internet Security System, XForceHelpFiles, http://www.iss.net/security_center/reference/vuln/PE _Overlapping_Header.htm
(Access Date: 29th April, 2012)

[14] IBM Internet Security Systems Ahead of the threat, “PE/COFF with overlapping header has been detected”, http://xforce.iss.net/xforce/xfdb/20909, 2009
(Access Date: 29th April, 2012)

[15] A Websense® White Paper, “Win32 Portable Executable Packing Uncovered”, http://securitylabs.websense.com/content/Assets/HistoryofPackingTechnology.pdf
(Access Date: 29th April, 2012)

[16] Truncated binary encoding, http://hyperfacts.co m/236048
(Access Date: 29th April, 2012)

[17] Run-length encoding, http://www.data-compression.info/Algorithms/RLE/index.htm
(Access Date: 29th April, 2012)

[18] EndMemo©, http://endmemo.com/bio/shannone ntropy.php, 2012
(Access Date: 29th April, 2012)

[19] Geoff McDonald, “Minimum Entropy Deconvolution (MED 1D and 2D)”, http://www.mathworks.co m/matlabcentral/fileexchange/29151-minimumentropy-deconvolution-med-1d-an d-2d, October 2010
(Access Date: 29th April, 2012)