Malware Behaviour Visualization

Article *in* Jurnal Teknologi · September 2014

DOI: 10.11113/jt.v70.3512

2 authors:

Syed Zainudeen Mohd Shaid
Universiti Teknologi Malaysia
9 PUBLICATIONS **71** CITATIONS
SEE PROFILE

Mohd Aizaini Maarof
Universiti Teknologi Malaysia
177 PUBLICATIONS **1,479** CITATIONS
SEE PROFILE

Jurnal Teknologi

Full paper

Malware Behaviour Visualization

Syed Zainudeen Mohd Shaid*, Mohd Aizaini Maarof

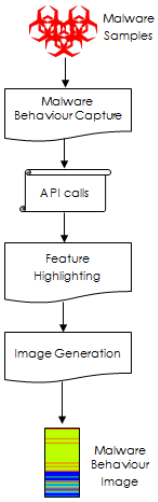
Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia

*Corresponding author: szainudeen@utm.my

Article history

Received :1 January 2014
Received in revised form :
1 June 2014
Accepted :10 September 2014

Graphical abstract



Abstract

The number of unique malware variants released each year is on the rise. Researchers may often need to use manual static and dynamic analysis to study new malware samples. Manual analysis of malware samples takes time. The more time taken to analyse a malware sample, the larger the damage that a malware can inflict. A lot of techniques have been devised by researchers to facilitate malware analysis and one of them is through malware visualization. Malware visualization is a field that focuses on representing malware features in the form of visual cues or images. This could be used to convey more information about a particular malware. Existing malware visualization techniques lack focus in visualizing malware behaviour in such a way that could enable better analysis of malware samples. In this paper, a new technique for malware visualization called 'Malware Behaviour Image' is presented. From the test results, the proposed technique is able to accurately capture and highlight malicious behaviour of malware samples, and can be used for malware analysis, detection and identification of malware variants.

Keywords: Malware; malware behaviour; malware visualization; malware behaviour visualization

Abstrak

Terdapat peningkatan di dalam jumlah pengeluaran sampel malware yang unik setiap tahun. Para penyelidik kadangkala terpaksa menggunakan cara manual untuk menganalisa sampel malware menerusi teknik analisa statik dan dinamik. Analisa malware secara manual memakan masa. Lebih banyak masa yang diperlukan, lebih parah akan kesan penularan sesuatu sampel malware. Pelbagai teknik telah dicipta oleh para penyelidik bagi memudahkan analisa terhadap sampel malware dilakukan dan salah satu daripadanya adalah menerusi visualisasi malware. Visualisasi malware adalah suatu cara untuk memaparkan malware dalam bentuk visual. Cara ini dapat memaparkan lebih banyak maklumat tentang sesuatu sampel malware. Teknik visualisasi malware sedia ada kurang fokus di dalam visualisasi perilaku malware yang dapat menyumbang ke arah analisa malware yang lebih baik. Di dalam kertas ini, kami paparkan hasil kajian kami di dalam visualisasi perilaku malware ke dalam bentuk gambar yang dikenali sebagai Gambar Perilaku Malware. Hasil ujikaji menunjukkan bahawa teknik ini mampu untuk menggambarkan perilaku hasad malware secara tepat, dan boleh digunakan untuk analisa malware, pengesanan dan identifikasi varian malware.

Kata kunci: Malware; perilaku malware; visualisasi malware; visualisasi perilaku malware

1.0 INTRODUCTION

Malware is software which runs much like other software. The key difference between malware and non-malware (benign) is in the behaviour of that particular software. If a software shows malicious activities like stealing user data, replicating, disabling certain security feature, serving as a backdoor, or executing commands not intended by the user, then it can be considered as malware.

It is estimated that more than 286 million unique variants of malware are released in the year 2010 alone, which roughly translate to an average of 784,000 unique variants per day according to a report by Symantec Corp[1]. The number of unique malware variants increased 41% in the year 2011[2], and continue to increase in the year 2012[3]. The statistics clearly show that malware is a serious problem and thus, it is not surprising that there are a significant number of studies being done by researchers in the field.

Analysing a lot of malware samples manually is something which is inevitable due to the growing number of malware samples each year[1,2,3]. Researchers need a technique that could enable quick and easy analysis of malware samples, especially on the behavioural aspects of the sample.

In this paper, a new technique for malware visualization that highlights the behavioural aspects of malware will be disclosed. The technique displays malware behaviour in the form of images (called Malware Behaviour Image). The presented method can be used for malware analysis, detection and identification of malware variants.

The next section discusses the problem background, the research strategy, and the related works in malware visualization followed by a number of sections that will describe in detail, the steps necessary in generating a Malware Behaviour Image. Finally, we will present some use cases to highlight the possibility of using Malware Behaviour Image in malware analysis, detection and identification of malware variants.

2.0 PROBLEM BACKGROUND

The process of detecting a malware and generating a signature for a newly released malware sample is time consuming and may require manual analysis of malware samples[4]. Whenever there is an outbreak of a new malware, researchers will grab hold of a sample, analyse it and then release a signature for that particular sample. There are two approaches towards analysing a malware sample. These are dynamic analysis, and static analysis. Dynamic analysis is a technique for studying the behaviour of a malware sample while the sample is being executed. Static analysis, on the other hand, is a technique that enables the study of a sample without the need for sample execution.

Often manual analysis of malware samples cannot be avoided, especially on new malware samples. The two approaches, when done manually, will require a lot of time. The longer the time it takes to analyse a malware sample, the longer the gap between the release of a new sample and the release of a new signature. With the ever increasing number of malware samples to process, a new technique that could aid in malware analysis is needed.

3.0 STRATEGY

One possible way to assist analysis of malware samples is through the use of malware visualization. Malware visualization is a field of knowledge that focuses on representing malware features in the form of visual cues that could be used to deliver more information about a particular malware in a more compact manner. Visualization helps researchers to better understand malware graphically, highlighting certain interesting aspect of malware which might not be conceivable in other forms of malware analysis. This could lead to more knowledge being extracted from the same amount of data, and thus contributes towards better understanding of the workings of a malware.

Existing malware visualization techniques seems to be able to generate similar images for the malware of the same family[5,6,7]. This is probably due to the fact that these images are derived from features of malware samples. Malware from the same family tends to share, at a different degree, the same feature[5,6,7] and this consequently leads to the creation of similar looking images. This trait can be exploited to create a technique that could help in better understanding of malware samples through visualization.

4.0 RELATED WORK

There are currently 4 documented malware visualization techniques in malware research. These are Malware Treemap[5], Malware Threadgraph[5], Malware Image[6], and VERA[7].

Malware Treemap is a visualization technique that visualizes malware behaviour in the form of an image of nested rectangles by Trinius *et al.*[5] It is a behaviour-based, malware visualization technique that takes API calls of malware as input and visualizes it into a colour image. The API calls are obtained by executing malware samples inside a VM. API calls of malware samples are not used in raw form. Instead, all API calls are grouped into several sections[5]. Such grouping increases the abstraction level represented by the image since the input used in generating the image is of low granularity behaviour data (grouped API calls of similar functionality).

Malware Threadgraph, as its name suggests, is a graph that plots the activity of each of the threads in a malware by Trinius *et al.*[5] 'Thread' here refers to the execution threads of a malware sample. The number of threads may differ among malware samples. Single-threaded malware will have only one line plotted in the threadgraph while multi-threaded malware will have 2 or more lines plotted (depending on the actual number of threads in use). Malware Threadgraph shares a lot of common attributes with Malware Treemap. For instance, both uses API calls obtained through the same method (CWSandbox), and grouped into sections of behaviour with similar functionality. However, instead of representing the percentage of sections, Malware Threadgraph represents the chronological order of the sections and the transition between the regions that was represented by an API call. The graph is plotted from left to right and is limited to showing only the first 550 operations (and hence can only show 449 section changes)[5].

Unlike Malware Treemap and Malware Threadgraph, Malware Image is a static feature based, malware visualization technique and thus, did not require execution of malware samples for feature extraction[6]. Malware samples are visualized based on raw malware data available on the malware binary itself. Each byte of the malware binary is interpreted as an 8-bit unsigned integer value ranging from 0 to 255, where a value of 0 will represent black while a value of 255 will represent white.s Once each byte of the malware binary has been converted into values that represent a grayscale color, a grayscale image will be plotted, from left to right, top to bottom. The width of the image depends on the size of the malware sample. The bigger the sample size, the larger the width of the image will be[6].

VERA (Visualization of Executables for Reversing and Analysis) is actually a framework by Quist and Liebrock[7] for visualizing malware samples in the form of a 3D image. VERA was not meant for malware classification but rather, it was aimed at assisting malware researcher in doing malware analysis, especially on manual malware unpacking. VERA helps differentiate code section entropy and monitor the creation, deletion, and modification of code sections of malware in memory by representing executable code blocks as colour coded nodes. VERA uses a modified hypervisor-based VM for monitoring the execution of malware sample. Details such as memory address, memory state, code entropy and several other state details were recorded. This information is later used in creating a 2D image consisting of colour coded nodes (that represent code blocks) and branches (that represent flow of code execution). This 2D image is later converted to 3D for better visual appeal.

Table 1 shows a comparison between existing malware visualization techniques. Malware Treemap, Malware Threadgraph, and VERA are malware visualization techniques based on the use of dynamic feature of malware. Malware Image on the other hand, is a static feature based malware visualization technique. The use of raw malware binary for the creation of Malware Images includes visualization of non-behaviour related data such as the PE header (metadata), and resources (*e.g.* icons, bitmaps, xml files, *etc.*)[8]. This could affect the overall accuracy of the generated image, especially in cases where the size of non-behaviour related data is greater than the size of behaviour related data in a malware binary (*e.g.* malware with lots of resources). Therefore, Malware Image is not a good candidate for a visualization technique that could accurately visualize malware behaviour.

Malware Treemap did not capture information on malware behaviour sequence and represents behaviour in the form of sections. These traits cause the technique to represent low granularity malware behaviour, which is not sufficient for use in differentiating very similar malware families or groups. The same is true for Malware Threadgraph that makes use of limited number of behaviour sections.

The 3D malware model by VERA is not suitable for representing malware behaviour. This is because the VERA framework is only interested in capturing memory location and other memory state information. While the technique is good for malware analysis, especially in malware unpacking, it does not capture any data that could be related to malware behaviour and therefore, could not be a good candidate for a malware behaviour visualization technique.

All of the abovementioned malware visualization techniques have restrictions that prevent them from being used in visualizing the behavioural aspects of malware samples for malware analysis. Therefore, a new technique that can clearly highlight the behavioural aspects of a malware sample is needed. The next section presents a new technique for malware behaviour visualization. The technique will have malware samples as input and generates images of malware behaviour called 'Malware Behaviour Image'.

Table 1 Comparison of existing malware visualization techniques

Visualization Technique	Type of Feature Used	Feature Used	Limitations
Malware Treemap	Dynamic	API calls	Low granularity, No sequence information
Malware Threadgraph	Dynamic	API calls	Low granularity, Limited to 550 operation
Malware Image	Static	Raw malware binary	Does not represent actual malware behavior
VERA	Dynamic	Memory address, memory state, code entropy, *etc.*	Not meant for representing malware behaviour

■5.0 VISUALIZING MALWARE BEHAVIOUR

The process of visualizing malware behaviour can be summarizes as in Figure 1. There are 3 important processes in generating a malware behaviour image. We start by capturing malware behaviour. Before the captured behaviour are transformed into images, a behaviour-to-colour mapping needs to be created. This mapping is responsible for highlighting certain features in the behaviour image. In case of malware, we want to highlight malicious features and therefore, assign colours that will protrude malicious behaviour inside the behaviour image. Once the behaviour-to-colour mapping is ready, the behaviour image is generated by converting each captured behaviour into colours that together, forms the behaviour image. Details on each of the processes involved are explained in the next section.

5.1 Malware Behaviour

There have been a lot of attempts by researchers to find the perfect candidate that could represent malware behaviour. Malware behaviour refers to what malware does, exhibits, or causes to its environment during live execution. Among the candidates for representing malware behaviour includes monitoring changes to operating system resources during malware execution[9], capturing malware's API call sequence[10,11], malware's I/O request packets (IRP)[9], and malware's network activity[13,14].

In monitoring changes to operating system resources, Jiang *et al.*[9] has outlined a method similar to taking snapshots of a system state at certain time intervals. The outlined method works best for capturing changes in operating system's resources, but not the order in which these changes happened. Besides that, the technique is only limited to analysing well-known structures like the process list, partition table, or the file system table.

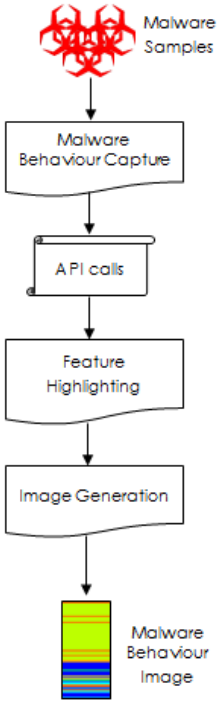


Figure 1 Processes in Malware behaviour visualization

API call monitoring is a rather effective approach for capturing malware behaviour. An API is self-explanatory and could (very clearly) tell if an application is trying to access a file, a network service or even attempt to modify memory content of a remote process. Currently, according to Nataraj *et al.*[15] the technique proved to be the one that provides the most accurate result for representing malware's behaviour, especially in the case of malware classification.

Zhang *et al.*[12] later presented a new indicator for malware behaviour by using I/O Request Packet (IRP). An IRP can be generated whenever a user mode application requests for an I/O operation, usually through the use of windows API. Not all APIs will generate an IRP, especially in the case of pure user mode API. The paper claimed that IRP is better than API call monitoring because API call monitoring could only be used on user mode applications, but this is far from true. Kernel mode applications do use APIs (albeit different from user mode APIs) and these API calls can be monitored.

Capturing malware's network activity, while useful in studying spyware and botnets, is not suitable for representing malware behaviour. Some malware might have very little or no network activity (*e.g.* Virus) and this could hinder malware analysis due to lack of data.

In our research, API call is used to provide malware behaviour data for malware behaviour visualization due to its proven effectiveness in representing malware behaviour[15].

5.2 Capturing Malware Behaviour

The first step towards behaviour visualization is to obtain behaviour data. An API call monitoring utility is executed on each of the malware sample. The utility executes malware samples in real time and collects all user mode API calls made by each malware sample. The reason we opt for user mode API calls is because of its conciseness in representing malware behaviour.

User mode API is stable in the sense that it rarely changes as opposed to the lower level system call, which differs a lot between different versions of the same operating system. The sequence of lower level system call that is used to implement a user mode API might change and this could render analysis of malware on one version of an operating system to be unusable on other versions.

Besides being stable, capturing user mode API is also crucial in speeding up the process of understanding the workings of a malware since user mode API reflects the exact nature of a particular malware. In Windows for example, it is far easier to understand the user mode API call *CreateFileA* than several system calls that does memory allocation and de-allocation. Running a kernel mode API call monitor on a malware will result in a flood of APIs which vaguely reflects the behaviour of the malware. Capturing user mode API on the other hand will result in a clearer description of malware behaviour. This is similar to viewing the source code of the malware, where one can see high level description (user mode APIs) of a malware.

5.3 Feature Highlighting

There are several approaches towards mapping API calls to colour. One way to do this is by creating a colour mapping whose colours are randomly selected. Assuming that we use 8-bit colour per channel (RGB), we could represent around 256^3 or ~16 million colours which would be more than enough to represent captured APIs. However, during our initial test, we find that the behaviour image generated using this mapping technique lack intrinsic value because the colours in the behaviour image did not convey any meaningful information.

We then settle for another approach, by grouping and sorting APIs based on the level of maliciousness. There are a lot of APIs (in Windows) and each APIs varies in its level of maliciousness. Some API like *DeleteFileA* (which deletes files in a file system) is considered malicious while others like *malloc* (which allocates memory from the heap) could be considered as less harmful. We enumerated a list of APIs and group them according to their level of maliciousness. We then assign certain colour (*e.g.* red) to malicious APIs and some other colour (*e.g.* blue) to non-malicious APIs. By doing this, we would have a visual representation of the level of maliciousness of a malware sample. In our test, we find that this approach yield better visual information compared to the previous one.

In order to map all API calls to colour, we need a set of colours that can uniquely represent all of the sorted API calls in the previous phase. To do this, we select the hot-to-cold colour ramp (see Figure 2). Hot colours (*e.g.* red, orange, *etc.*) will represent malicious APIs while cold colours (*e.g.* cyan, blue, *etc.*) will represent non-malicious APIs.

Once we have the sorted APIs and the colour ramp, we map the APIs (as in Figure 3) so that APIs that are in the malicious region will have hot colours and APIs that are in the non-malicious region will have cold colours. The hot-to-cold colour ramp enables the Malware Behaviour Image to highlight malicious API calls made during the lifetime of a malware.

The colours are represented using the RGB (Red, Green, Blue) colour model. In the model, the chosen colour ramp starts from red (1,0,0) to yellow (1,1,0) to green (0,1,0) to cyan (0,1,1), and to blue (0,0,1) as in Figure 4.

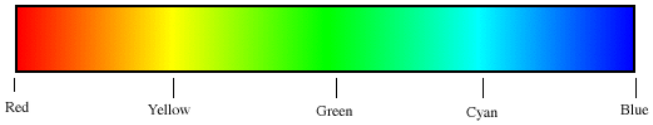


Figure 2 Hot-to-Cold colour ramp

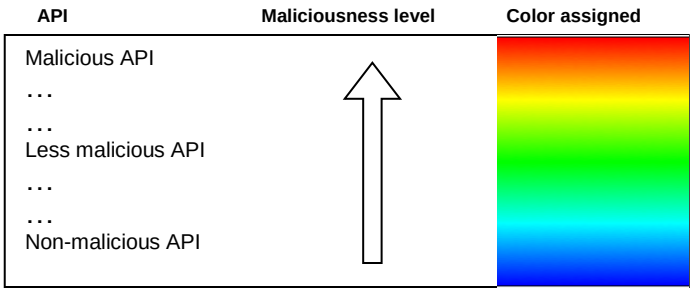


Figure 3 API-to-Colour mapping

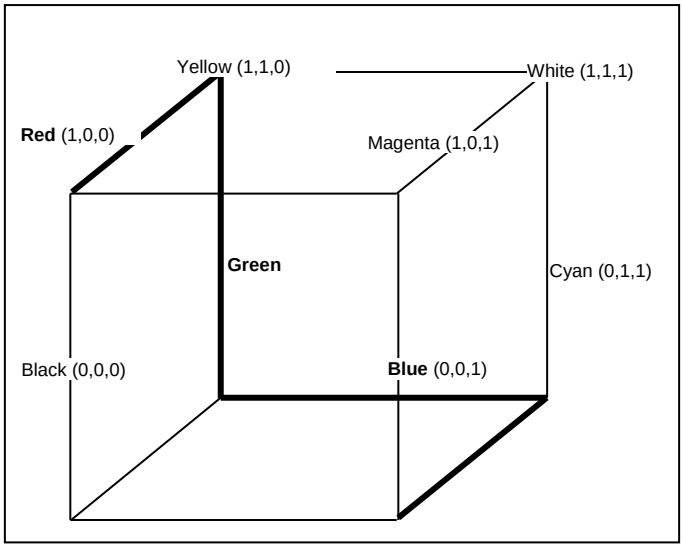


Figure 4 RGB colour cube depicting the line that represents the hot-to-cold colour ramp

Therefore, given a list of APIs (sorted based on the level of maliciousness) with an index ranging from 1 to n, API_1 will map to RGB colour 1,0,0 while APIn will map to 0,0,1. We calculate the value of each colour element R,G,B of index i (where each colour channel is represented with a value scale of 0 to 1) using the following formula (1).

$$Color(i)=\begin{cases} R = 1 & ,1 \le i < 0.25(n) \\ G = 4i \div n & \\ B = 0 & \\ & \\ R = 2(n-2i) \div n & ,0.25(n) \le i < 0.5(n) \\ G = 1 & \\ B = 0 & \\ & \\ R = 0 & ,0.5(n) \le i < 0.75(n) \\ G = 1 & \\ B = 4(i-(n \div 2)) \div n & \\ & \\ R = 0 & ,i \ge 0.75(n) \\ G = 4(1-(i \div n)) & \\ B = 1 & \end{cases} \quad (1)$$

5.4 Image Generation

Once we had the API-to-colour map ready, we then proceed to generate a behaviour image for each sample. Each API will cause a 64 x 4 pixel image to be painted to the behaviour image. We find that the use of a 64 pixel (width) by 4 pixel (height) result in an image that could be easily compared against each other while being easy to the eyes (not too small or too big). The malware image will be painted top-to-bottom, which means earlier APIs will be at the top while API calls near the end of the API call monitoring will be at the bottom of the behaviour image.

Figure 5 shows some example of malware behaviour image generated using our proposed technique.

6.0 EVALUATION

It is common for malware authors to modify certain aspects of a malware sample as a defensive mechanism, to thwart detection by signature-based malware scanners. It is therefore important to have a malware visualization technique that is resistant towards such modification. This section presents test results that evaluate the proposed technique's ability in visualizing malware behaviour without being affected by static-based changes (metadata alterations) and dynamic-based changes (code packing) made to malware samples.

6.1 Metadata Alteration Test

This test measures the techniques ability in resisting static-based changes made to malware sample. There are 2 types of changes made to these samples. Details on the type of metadata changes made are listed in Table 2.

Table 3 below shows differences of samples before and after the modifications made to them. Each sample has different MD5 hashes which means that no two sample are the same (even if they have the same size). Each sample is tested to make sure that the modifications done do not corrupt the sample and that the samples are working normally (e.g. no premature termination).

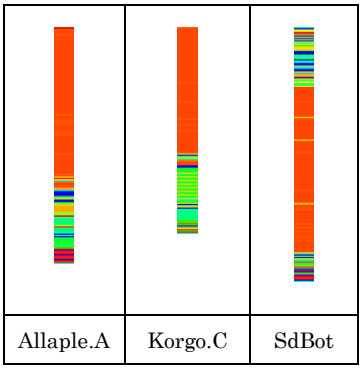


Figure 5 Malware behaviour image sample for 3 different type of malware

Table 2 Metadata changes made to malware samples

Type of metadata change	Detail
PE header change	1. Values in the DOS Header 2. Values in Section Headers
Resource modification	1. Resource substitution 2. Addition of new resource

Table 3 Sample malware set with metadata changes

Sample	Modification	Size (KB)	MD5 hash
1	No modification	62	216acc5aafc68fa1b442d396e1a79db9
	PE header change	62	5265b4ebd3e12e85ff2f4d83094cb4f8
	Resource Modification	63	7ff52fe0179da517e0c78aff9f7c91f5
2	No modification	57	00a93869f8f009fb16d1a7d8f2657639
	PE header change	57	a217f0f9739d25a34177fa8ea8890bff
	Resource Modification	58	1131cfbc706c65c9b795e37b744cbb63
3	No modification	42	470b88083967272ebbeb0167381199e8
	PE header change	42	617fa53401e4047ef5140fc9921b638e
	Resource Modification	44	2ca075d0fccae4fd3990e498d3313d57
4	No modification	71	12b684aa174471206ec5edaaef40b309
	PE header change	71	4eda98ca416a1bbb4108f89ffd5bfa8f
	Resource Modification	72	e650123076c4777168680a5d52139904
5	No modification	115	2c87e39be193e46e8d88e6900b1bdd1e
	PE header change	115	e2cc201dc037b8c40a142990cd4eaad8
	Resource Modification	117	9c38abbd791a7d59ff47fdb58ced180d

Figure 6 shows the generated MBIs for each of the above test cases, before and after modification. It was clear that metadata changes made to malware samples do not result in any change in the generated MBIs. This is because the proposed technique does not use static based data for visualization and hence, any static-based modification made to malware samples does not affect the output MBIs.

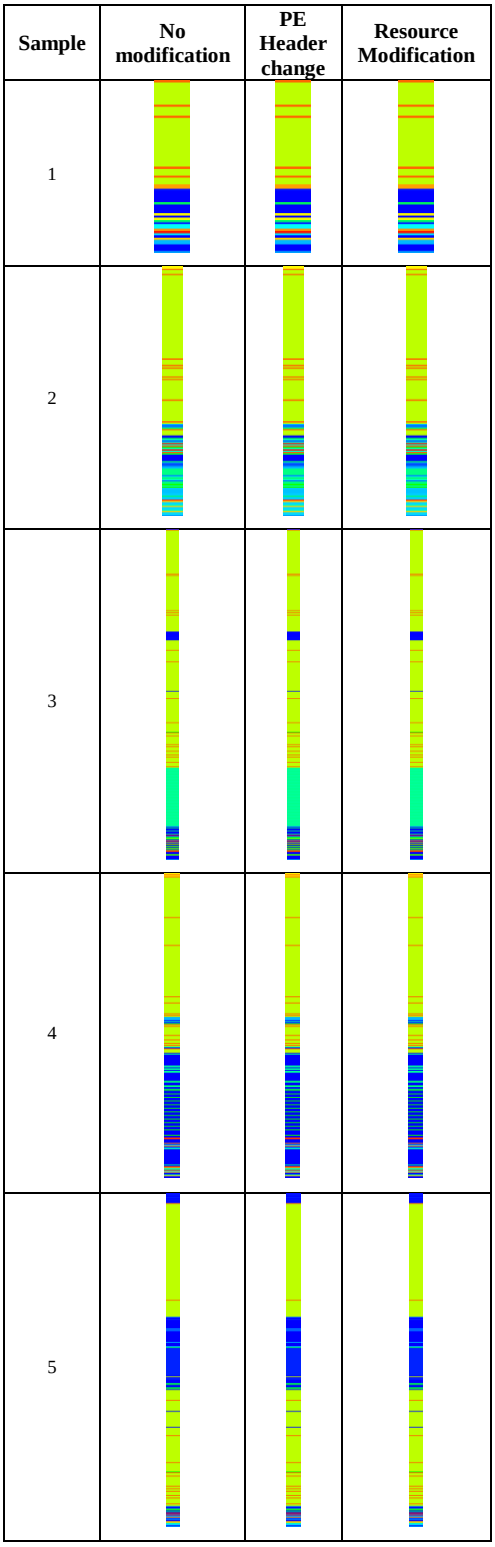


Figure 6 Metadata alteration test results

6.2 Code Packing Test

This section presents the result of dynamic-based changes made to malware samples or specifically, code packing. Most of the malware samples for the research (obtained from CyberSecurity Malaysia) are already packed with various packers. Of all the malware samples which were not packed, only 2 seems to be stable enough to be packed and run without problems. All others crashed during execution. For the test, the two stable samples were packed using UPX[16]. Table 4 shows the differences of each malware samples before and after UPX code packing.

From the table, it can be seen that samples packed with UPX are a lot smaller in size compared to the original ones. This is because UPX compresses executable files, making it smaller[13]. The MD5 shown in the table is just meant to show that each samples are unique.

Table 4 UPX on malware samples

Sample	Modification	Size (KB)	MD5 hash
1	No modification	231	011bbba2e3be13ea2aa84a1242e0d47b
	UPX	93	dc77226ae6c851c3cea006b0b05a04be
2	No modification	286	449c163674eeaffd44ca3f1eb62d670d
	UPX	114	27e92c792b4f72f5bdc30b979c2b8fc4

MBIs for each sample were generated and compared. Figure 7 shows the generated MBIs for each test case. Through visual inspection of the MBIs, it was clear that the proposed technique is able to show behaviour added by UPX and the original behaviour of the malware sample. In fact, in the experiments conducted, it was possible to identify the presence of a specific packer by comparing the behaviour on the top of the MBIs against a known pattern of a packer.

■7.0 APPLICATION OF MALWARE BEHAVIOUR VISUALIZATION

Malware Behaviour Image could possibly open up a new paradigm for malware research. We have tested several possible uses of Malware Behaviour Image in the field of malware research. Below we highlight two possible uses; malware detection, and malware variant identification.

7.1 Malware Detection

We wanted to see if there is a difference between images of benign samples and images of malicious samples. Therefore, we generated behaviour images for benign samples and malicious samples. The benign samples are taken from a newly setup 32-bit Windows XP system. They are composed of 7 different types of application, characterized based on their functionality. The list of application category is listed in Table 5.

In a test environment, we took 20 benign samples (from 7 different categories) and 20 malicious sample (from different malware variants), mix them together and generated their behaviour images. We then try to visually determine if an image is of malicious sample or of benign sample.

Table 5 Application category for benign samples

Application Category	Sample count
File browser	1
System Tools	4
Games	4
Network Utility	2
Internet Application	4
Utility Application	3
Text Editors	2

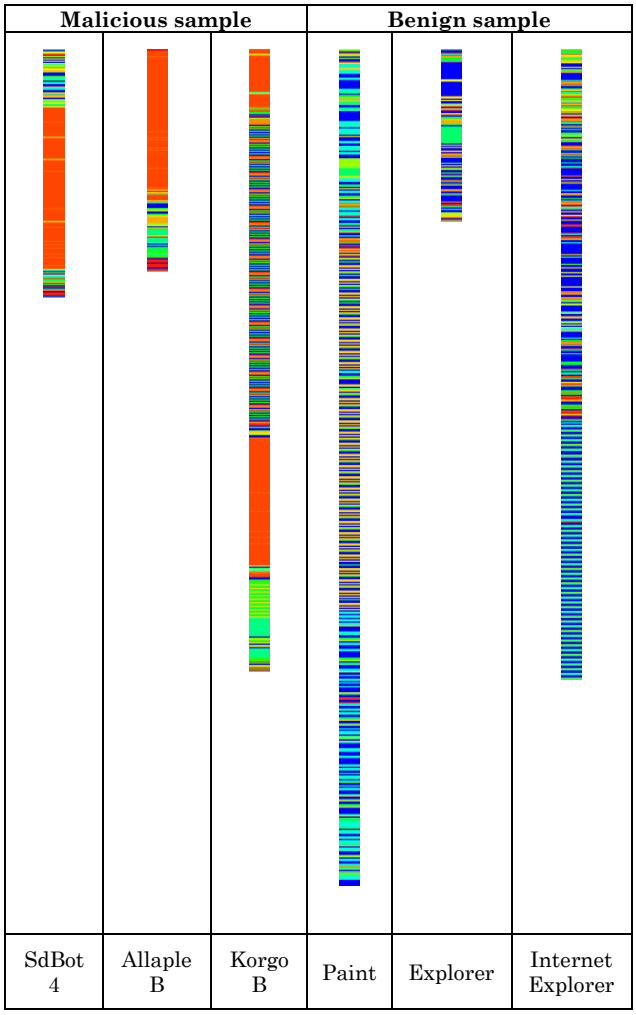


Figure 7 Code packing test result

Figure 8 shows some of the behaviour image that we have generated. It is easy to see that behaviour images of malicious samples have more hot colours in them compared to benign samples (that are mostly composed of cold colours). From visual inspection, we find that we could easily differentiate between malicious sample and benign sample with 100% accuracy by spotting the presence of hot and cold colours in the behaviour images.

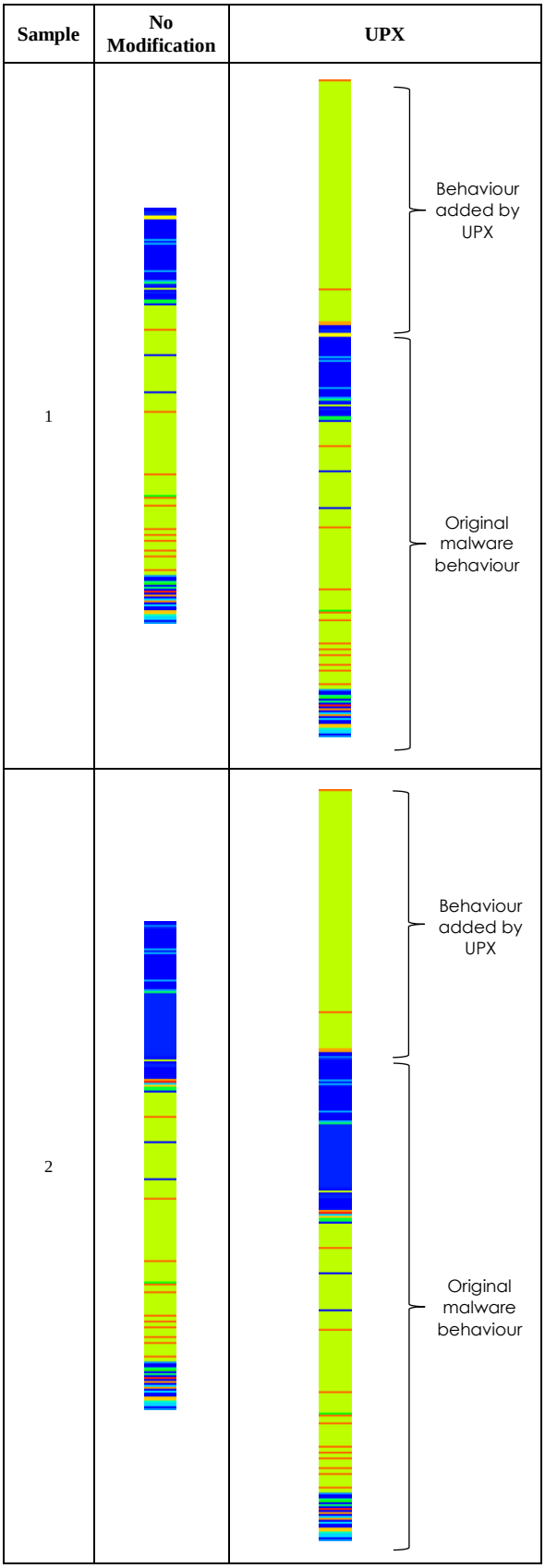


Figure 8 Behaviour image for malware and benign sample

The reason for such an easy differentiation of malicious and benign sample can be understood further from the colour histogram below. Figure 9 shows 2 histograms, where one represents the frequency of API calls made by malicious sample and the other one is the frequency of API calls made by benign samples, generated using matplotlib[17]. The bars to the left of the histograms (with hot colours) represent the frequency of malicious API calls while the bars to the right of the histogram (with cold colours) represent the frequency of non-malicious API calls.

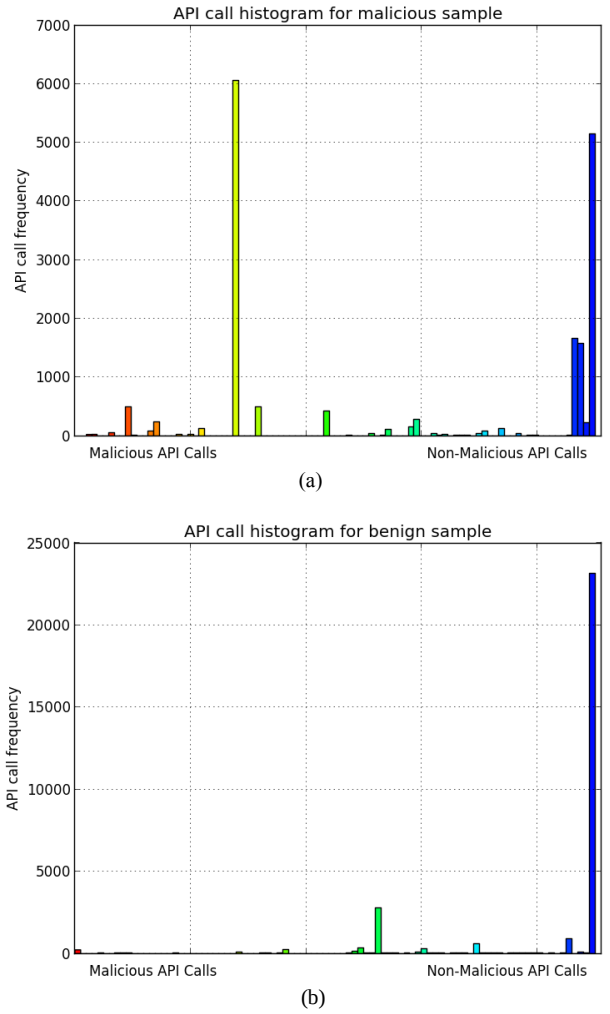


Figure 9 (a) API call histogram for malicious sample, (b) API call histogram for benign sample

In Figure 9(a), we can clearly see that the malicious samples generated more malicious API calls compared to benign samples (in Figure 9(b)). While benign samples do invoke APIs of malicious nature, the frequency of the API calls is nothing near to the frequency of malicious API calls made by malicious samples. It is therefore logical that behaviour image of malicious samples will have more hot colours in them, thus leads to easier recognition of malicious and benign samples.

7.2 Malware Variant Identification

We notice that malware from the same family tends to have a recognizable pattern in them. The similarity of behaviour images of a malware family varies among each family. Some malware family have similar looking behaviour image like the Allaple family (see Figure 10) even though each of the samples in the family have different sizes and different hashes.

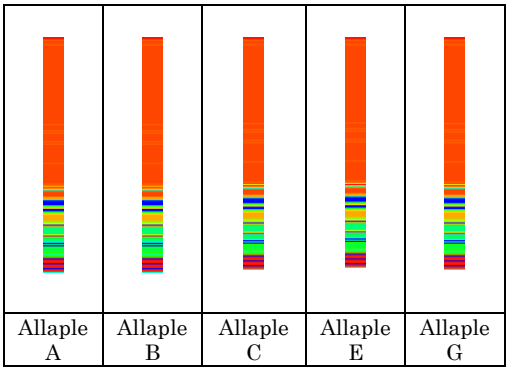


Figure 10 The Allaple family with a set of similar looking behaviour images of its variants

The reason for similar looking behaviour images can be fully understood by manually inspecting the malware through static and dynamic analysis. Malware family with similar looking behaviour images (but with different malware sizes and hashes) might suggest that variants of the malware family are created purposely for evading signature based identification by anti-virus (AV) software.

There are also family with minor changes between behaviour image of its samples (see Figure 11(a)) and some have significantly more changes among its variants (see Figure 11(b)). These changes could suggest that either these variant have some level of randomness in determining the path of code execution, or there might be certain behavioural alterations among malware variant, like features being improved, added, or removed from previous variants. However, these changes did not seem to hide the overall pattern and the similarity between variants of the same family.

In our test, we find that given enough time, we can visually identify variants of malware from the same family through behaviour images with high accuracy. This is no surprise since each malware family is created differently by various authors, with probably different compiler and coding style. Behaviour image seems to be able to visually highlight these aspects and this contributes to the possibility of accurate visual identification of malware variants.

8.0 CONCLUSION AND FUTURE WORKS

We have presented a technique for visualizing malware samples that highlights the behavioural aspects of malware. The behaviour images can be used to visually identify malicious and benign samples, and can also be used to visually identify malware variants with high accuracy. The proposed technique is not meant as a replacement for existing malware analysis technique, but rather, a new paradigm, a technique that complements existing techniques for malware analysis. The technique is able to visually highlight malicious behaviour in malware samples. This enables it to be used in malware detection. It was also discovered that malware variants have unique set of colours of pattern when visualized using the proposed technique. This makes it suitable for use in classification of malware samples. We plan to extend the current work to automate the process of malware detection and malware variant identification, removing the need for manual

visual inspection. Hopefully, this could contribute towards better automated analysis of malware samples.

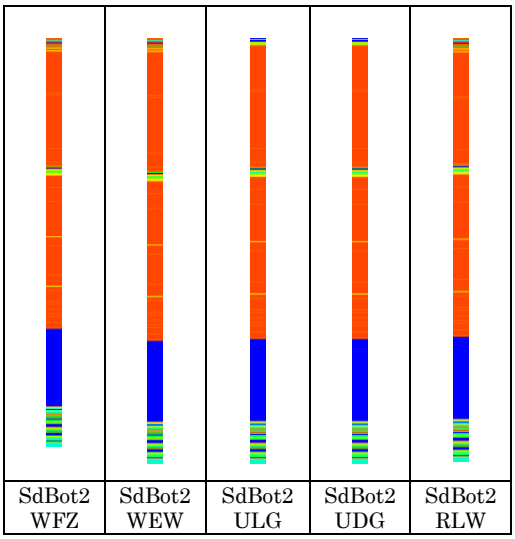


Figure 11 Behaviour images of the (a) SdBot2 malware family and the (b) Korgo malware family

Acknowledgement

The authors would like to thank CyberSecurity Malaysia for their utmost cooperation in providing malware samples for the research.

References

[1] Symantec Corp. 2011.Symantec Internet Security Threat Report–2010. *Internet Security Threat Report Volume 16*. Technical Report. http://www.symantec.com/business/threatreport/.

[2] Symantec Corp. 2012. Symantec Internet Security Threat Report 2011. *Internet Security Threat Report, Volume 17*. Technical Report. http://www.symantec.com/business/threatreport/.

[3] Symantec Corp. 2013. Symantec Internet Security Threat Report 2012. *Internet Security Threat Report, Volume 18*. Technical Report. http://www.symantec.com/business/threatreport/.

[4] Egele, M., Scholte, T., Kirda, E., and Kruegel, C. 2011. A Survey on Automated Dynamic Malware Analysis Techniques and Tools. *ACM Computing Surveys*. 1–49.

[5] Trinius, P., Holz, T., Gobel, J., and Freiling, F. C. 2009. Visual Analysis of Malware Behaviour Using Treemaps and Thread Graphs. 6th International Workshop on Visualization for Cyber Security, 2009 (VizSec 2009). Oct 2009. 33–38.

[6] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. 2011. Malware Images: Visualization and Automatic Classification. Proceedings of Visualization for Cyber Security (VizSec). 2011: 1–7

[7] Quist, D. A. and Liebrock, L. M. 2009. Visualizing Compiled Executables for Malware Analysis. In International Workshop on Visualization for Cyber Security (VizSec). 27–32.

[8] Microsoft. 2010. Microsoft PE and COFF Specification. Technical Report, Microsoft.

[9] Jiang, X., Wang, X., And Xu, D. 2007. Stealthy Malware Detection through vmm-based "out-of-the-box" Semantic View Reconstruction. Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07). New York, NY, USA: ACM. 128–138.

[10] Nair, V. P., Jain, H., Golecha, Y. K., Gaur, M. S., And Laxmi, V. 2010. MEDUSA: Metamorphic malware Dynamic analysis Using Signature from API. Proceedings of the 3rd international conference on Security of information and networks (SIN '10). New York, NY, USA: ACM. 263–269.

[11] Trinius, P., Holz, T., Gobel, J., And Freiling, F. C. 2009. Visual Analysis of Malware Behaviour Using Treemaps and Thread Graphs. 6th International Workshop on Visualization for Cyber Security, 2009 (VizSec 2009). Oct. 33–38.

[12] Zhang, F. Y., Qi, D. Y., and Hu, J. L. 2010. Using IRP for Malware Detection. Recent Advances in Intrusion Detection in Lecture Notes in Computer Science.Springer Berlin/Heidelberg. 514–515.

[13] Ahmed, I., and Lhee, K. S. 2011. Classification of Packet Contents for Malware Detection. *Journal in Computer Virology*. 279–295.

[14] Skrzewski, M. 2011. Flow Based Algorithm for Malware Traffic Detection. *Computer Networks in Communications in Computer and Information Science*. Springer Berlin Heidelberg. 271–280.

[15] Nataraj, L., Yegneswaran, V., Porras, P., and Zhang, J. 2011. A Comparative Assessment of Malware Classification Using Binary Texture Analysis and Dynamic Analysis. In Proceedings of the 4th ACM workshop on Security and artificial intelligence (AISec '11). ACM, New York, NY, USA. 21–30.

[16] Oberhumer, M. F., and Molnár, L. 2013. The Ultimate Packer for eXecutables (UPX). UPX. Retrieved June 13, 2013, from http://upx.sourceforge.net/.

[17] Hunter, J. D. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*. 9(3): 90–95.