Malware Secrets: De-obfuscating in the Cloud

Arsh Arora, Thomas Stallings, Ragib Hasan, and Gary Warner {ararora, tds2, ragib, gar}@uab.edu
University of Alabama at Birmingham, AL 35294

***Abstract*—Malicious software, universally known as malware, is typically used to cause disruption as it tries to steal sensitive information such as passwords, credit card numbers and other pertinent information. Malware infections have increased tremendously over the last decade. Recent reports indicate that around 70% of malware infections go undetected by the antivirus software. The infections that remain undetected fall into the category of zero-day malware, which is defined as malware that is new and is essentially an undiscovered and undisclosed threat. Furthermore, its substructure or the functioning has not been understood, and no signatures have been defined to distinguish the zero-day malware from others. Moreover, an average enterprise receives 17,000 malware alerts per week, and if 70% goes undetected, then one is certain to be infected by the zero-day malware every week. Therefore, the low detection rates and increasing vulnerabilities have created an unmet need for the researchers to try and develop an algorithm that will help in timely and efficient detection of malware. Moreover, in our approach researchers used the cloud for malware detection, which is a safe, cost-effective and user-friendly environment.**

I. Introduction

Malware threats have been persistently increasing in the last few decades [1]. The rise in the malware families and the sheer volume of uniquely hashed malware samples are directly attributed to the development of new obfuscation strategies, toolkits, and release or theft of source code from previous malware/botnet [2]. Accordingly, there is a need for developing proper patches and signatures to overcome the new vulnerabilities in the future. This rise in the vulnerabilities encouraged researchers to try and develop techniques to detect malware at a rapid pace and reduce the zero-day [3] paradox of malware. By zero-day paradox, the researchers usually mean the time taken to discover and understand the substructure and organization of the malware. Due to the formation of new malware in the landscape, it becomes extremely challenging to keep track of the newly developed techniques and build preventive measures to protect against them.

Several well-known anti-virus software companies [4] across the globe attempt to provide protection from different forms of malware. However, they are not able to fulfill the task of identifying the malware appropriately and swiftly[5] due to the ongoing arguments about the conventional nomenclature and classifications of malware. For example, the family that is the focus of our study is BrowseFox, which has been referred by various names such as MultiPlug/ Mikey [6]. The process of identifying the malware by the antivirus vendors is traditionally based on the analysis of the signature or the analysis of the code, but it is not 100% confirmatory.

Another important cause of hindrance in malware detection is packed malware. Packing is the process of encrypting, obfuscating, or compressing the bytes of an executable and having some form of the restoration process to get it back into its executable form [7].

In our research, we present unique sets of features that are used to calculate a distance metric. A statistical algorithm called as the 'nearest neighbor search' was used to determine the similarity or likeness to other executables by employing the distance metric. This algorithm is referred as Dante Algorithm. Once the cluster is established, the focus is solely on statistical analysis of a particular cluster to find the similarities and differences within the cluster. Thus, helping to develop a prototype for further research. The prototype range will be used for future experiments and better prediction of the samples that classify in that range for a particular malware family.

The clustering of malware samples is a costly affair as it includes the overhead of maintaining malware database and computation cost of running to experiment to cluster malware. It takes more than a couple of hours on a desktop to cluster a small set of malware, making it impossible to analyze large data sets. Cloud provides an innovative and low-cost approach to perform the experiment. The overhead logistics can be modified when examining in the cloud environment.

II. Related Work

Various researchers have been trying to find a solution with hash comparisons such as MD5, SHA-128, and SHA-256 [8] [9]. Different types of packing make it difficult to compare the executables. The development of packing techniques has compelled the researchers to try and develop more innovative solutions. The new technology developed by researchers is PEiD [10], which assembles known patterns or signatures of packing strategies before testing a malware sample with the hash analysis. In a recent paper by Jaenisch et al. [11], a hypothesis was derived that was used to differentiate between malware and benign executables. A method was introduced to convert a disassembled program first into opstrings and then filter these into a recued opcode alphabet. Shoeb et al. proposed a redirected URL-based approach for analyzing and grouping spam email into significant clusters [12]. They showed that out of the 10000 sample spam emails, around 80% of the emails fall into four broad clusters. In a paper by Marnerides et al [13], an approach developed for malware analysis in a virtual environment. The paper mentions the positive impact of doing malware analysis in the cloud framework.

III. Method

A. Dante Algorithm

Increasing difficulties in detection of malware led the researchers to think differently to find a solution for this ongoing problem. We adopted an innovative approach combining mathematics with the clustering algorithm to find a solution. An algorithm named Dante is developed which uses statistical analysis and Euclidean distance to cluster malware. The two main steps used for calculation were 1) statistical analysis 2) distance calculations and heuristic adjustments.

B. Statistical Measurements

The first phase is to obtain a variety of statistical measurements for all of the bytes within the executable code of a binary file. These measurements consists of statistical measurements, higher order moments, fractal dimension measurements, compression, and randomness [11].

The most compelling features, when available, are file size, content, frequency, counts, and other types of consistent metadata commonly associated with binaries. Since in the ASCII character set, characters and numbers occur between 65 and 122, a mean in this region would point to text or string symbols, while a mean below this would indicate more control characters. The standard deviation, or sigma, shows the spread of values expected. A small sigma means most values inside the sample window occur close to the mean with little spread, and vice versa. String data would have a mean value near 70 with a sigma of perhaps 20, while code section data may have a mean near 60 with a sigma of 40 or more.

The skewness (M3) is an indicator of more values in the window being greater than or less than the mean. For instance, if the mean were 70, a positive skewness would indicate that there are more string letter and number characters than punctuation marks in the window. Kurtosis (M4) indicates whether the distribution is peaked or flattened. A significant kurtosis shows the peak of the distribution is much greater than the Gaussian, and more values occur close to the mean value than would occur in normal data.

The hyperflatness (M6) higher order moment indicates how fast the tails slope off of the curve. A small value means that the curve follows an expected Gaussian exponential trail-off converging to zero asymptotically. The higher order moment (M8) shows how multimodal the data is and if there are gaps in the distribution when there are values that occur in discrete bins of a narrow range rather than continuously across the interval.

The Jaenisch fractal dimension examines how the range of the values is related to its spread compared to random data. The closer the value is to 2.0, the more it resembles Gaussian random noise. On the other hand, the closer it is to 1.0, the more it looks like a trending line. The Handley fractal dimension examines how the data changes across the window. The data varies little when the length between the first point and last point is similar to the combined length of the segments.

C. Distance Calculations and Heuristic Adjustments

Euclidean Distance for N features can be expressed as:

$$Dis_{A \leftrightarrow B} = (\sum_{i=1}^{n} (Stat_{A_i} - Stat_{B_i})^2)^{1/2}$$

The distance between all of the samples in the dataset is measured using the Euclidean distance[14] combining with the 14 measurements. This distance calculation is chosen for its simplicity and ability to give a direct projection from any point in N-dimensional space to any other point. The lower the score, the closer the samples are to each other.

In the chance that samples fall close together out of coincidence, a heuristic function to adjust the distance between the two samples is implemented. This is achieved by using the Jaccard similarity coefficient between the strings of the two samples in the distance calculation. This similarity measurement returns a floating-point number between 0 and 1. Dividing the distance score by the similarity score makes the adjustment to the distance. This greatly increases if the similarity score is low.

Algorithm 1 Distance Average

```
n=0 , sum=0
while (dist[n] <CUTOFF) && (n <dist.length) do
    sum +=dist[n], n +=1
end while
Average = sum / n
```

D. Statistical Nearest Neighbor Searching

With all of the adjusted distances calculated for the samples, a nearest neighbor search [15] is employed to extract the clusters. Statistical analysis of the distance from the current node to all other nodes is used to determine the distance to be considered a close neighbor. To determine a suitable distance value, simple average and standard deviation are invoked on each sample distance to all other samples. The calculations are sorted in ascending order, and it allows for the values to be summed until the next value exceeds a cutoff value. The following can be seen in Algorithm 1. The distance to determine if a sample is seen as close to the current node is done by going two standard deviations from the average distance of the current node. If a sample falls within this range, it is considered neighbors. Once all of the samples that are considered neighbors to the current sample have been found, they too are searched for samples close to them. This continues recursively until all the samples in the cluster have been identified. Pseudocode is demonstrated in Algorithm 2.

Algorithm 2 Recursive Search

```
function SetCluster(node, clusterId)
    if node.visited then
        return
    end if
    node.visited = TRUE
    node.clusterId = clusterId
    for i = 0 to node.neighbors.length do
        if node.neighbors[i].visited then
            Continue
        else
            SetCluster(node.neighbors[i],clusterId)
        end if
    end for
end function
function GetCluster(nodes)
    clusterId = 0
    for i = 0 to nodes.length do
        if nodes[1].visited then
            Continue
        else
            GetCluster(nodes[i],clusterId)
            clusterId += 1
        end if
    end for
end function
```

IV. Testing and Results

Dante algorithm was used to analyze a set of 1,728 unknown malware executables. For the testing purposes, packed samples were omitted from this study because of the unavailability of universal unpacker. The algorithm resulted in 1,115 executables grouped in 116 different clusters. Virus total [6] website was used to confirm the samples from top four groups, gathering information about the submitted executable from the fifty-six antivirus vendors. These results were used to give the selected groups appropriate names as can be seen in Table I.

The preliminary analysis indicates that the top two clusters comprised the majority of the proportion of the executables. Upon further investigation, it was found that the first cluster with the count of 335 was exclusively packed with Nullsoft

TABLE I: Top 4 Clusters

Cluster	Name	Count
9	OutBrowse/Crossrider/Solimba	335
5	MultiPlug/BrowseFox/Mikey	174
3	HeurCorrupt	92
8	InstallCore/EoRezo	37

Scriptable Install System (NSIS) [16]. NSIS is a professional open source system to create standard Windows installers (as well as our launchers) packaged as a portable app so you can create installers and launchers anywhere. Using NSIS is a common practice that threat actors perform to avoid detection [17]. The presence of the packer meant that the commonality could be because of the packer. The malware beneath the packer may or may not have the same characteristics that might allow them to be clustered together. Hence, we decided to focus on the second largest cluster (5) that was named as BrowseFox/Mikey/MultiPlug interchangeably by VirusTotal. BrowseFox[18] is an adware that can be linked to your browsers. The main feature of the program is to display pop-up messages that may contain malicious ads and can damage/infect your system by a single click on the link.

The BrowseFox cluster returned a negative result when checked for an exterior packer. The executables in the cluster (5) were tested using the statistical analysis. The primary goal was to define a prototype range for each of the thirteen statistical values that will be helpful in categorizing the executables. The range calculated was for one data set, which was not sufficient. Hence, five different datasets were analyzed with Dante algorithm with a total of 8500 malicious executables.

To summarize, the steps performed were clustering a data set with Dante algorithm, selecting the group that contains executable named as BrowseFox or similar, and analyze them with the Statistical analysis. Once the process was accomplished, we found 687 BrowseFox executables.

A. *Development of Range*

The executables found were analyzed with statistical analysis. An average and standard deviation were calculated for each of the 13 statistics for the executables. One point standard deviation was added and subtracted from the mean to develop a range category for each of the statistics. Hence, a prototype range was developed.

To verify the accuracy of the range, researchers tested the range on the selected samples. While viewing the results, it was found that Skewness measure returned negative scores, which were not taken to consideration, therefore, were removed. The results obtained with the remaining statistics were around 60-61% on the positive side.

An important aspect to keep in mind is that the derived range had zero-tolerance for outliers. The results were strictly of the executable that fell within the range, which meant that even a single outlier out of the twelve forced us to omit the executable. This strict policy made the results impressive as we can plainly say that the following executables are BrowseFox.

The primary goal is to reduce the outliers and develop a better range. Before proceeding, researchers thought to confirm the findings with an unknown data set and provide a litmus test for the range and its effectiveness.

V. Reality Check

After the establishment of the prototype, it became mandatory to perform a litmus test on our findings. Researchers conducted the similar steps that were used to find the BrowseFox executables on a fresh set of samples. The number of samples that were classified as BrowseFox by both the algorithms were selected. Also, found out the overlaps in which both of them would have clustered the executable in the same group. Finally, calculating the percentage of duplicates for the number of samples.

TABLE II: Count of BrowseFox cluster with different algorithms

Dante	Range	Duplicates	% to Dante	% to Range
163	201	130	79.75	64.67
174	160	100	57.47	62.5

The results generated were astonishing. The common executables that were classified as BrowseFox by both Dante and statistical analysis were comparatively a higher percentage than the previous benchmark of 60%. As can be seen in Table II, the number of overlap between Dante and range is a high rate. The figure indicates good results for the prototype range as it can be classified consistent or slightly higher.

Keeping this in mind, the argument of using Dante clustering algorithm instead of statistical analysis is dominant. Researchers understand the consequences and are aware that Dante clustering algorithm is better than the statistical analysis. In Dante clustering algorithm, string similarity comparison is an added feature which provides better results than statistical analysis. On the contrary, string comparison is time-consuming and requires high computational power. Therefore, usability and efficiency will be pertinent if the researchers plan to use Dante in future. Researchers found that statistical analysis is significantly better than Dante when compared to the time and resource management. For example, the consumption time of a dataset of 1700 executables which is 2 GB file size when analyzed by Dante algorithm is close to 210-220 minutes, whereas the statistical analysis is close to 55-60 minutes.

It is visible that there is a vast time difference between the computation time of the results. For small samples sets, Dante clustering is perfect. When you increase the scale to tens of thousands executables per day, it becomes tough for Dante to perform at optimum capacity. On the contrary, statistical analysis may not provide higher percentage but is much faster. The focus of the experiment is in the high-quality results; hence, researchers were inclined towards statistical analysis and developing a better range for future use.

VI. Road towards Cloud

The next objective is to refine the process and move it to the Cloud framework. Cloud has numerous advantages as it supports parallel computing and saves computational power, time and money [19]. Other benefits associated with using the cloud are lack of upfront cost, null infrastructure or data center storage. The overhead cost of data storage are minimal.

A major attracting force in our research is that we are dealing with malicious executable, thus, hosting them locally on servers could be dangerous [13]. Cloud provides a sense of safety that even if the executable is not treated with caution, it will not effect the organization as a whole. Moreover, it would require special skill, knowledge, and proper machinery to find the desired files in the cloud. The attacker might be able to access the data but will be susceptible to self-infection. Thus, it will work as a blessing in disguise, and use of the cloud framework seems to be a step in the right direction for data analysis.

A. Preliminary test

During statistical analysis, researchers took a set of 1,821 executables and divided them into three sub-groups with one containing 1,821, second 611, and third 100. The analysis were performed on the groups while computation time was recorded.

TABLE III: Time taken to compute Statistics

Samples	Time (in minutes and seconds)	Time per Sample
1821	75 minutes 21 seconds (4521s)	2.48s
611	25 minutes 45 seconds (1545s)	2.52s
100	5 minutes 57 seconds (357s)	3.57s

The results in the table III state that computation time is inversely proportional to the number of samples. A basic idea was to analyze the different computational time for each group size. The experiment was done to understand whether it will be beneficial to run multiple instances or single instance corresponding to the number of executables. In researchers' opinion, it will be highly beneficial to run multiple instances on the cloud as division into different groups will prove useful and cost-effective as explained in the next section.

B. Economic Analysis

Economic aspect holds the key to any experiment. Irrespective of the usefulness of the approach, if it fails the economic side, it fails its purpose. Therefore, the proposed approach solves both the purpose of convenience and cost effectiveness. Although these factors may depend on various factors such as inflow of malware, instances required, time taken, feasibility, and the most important, cost.

The current analysis was performed on a Ubuntu 16.04 LTS with a memory of 5.8 GiB, but the study did not utilize much of the resources and can easily be carried out on 2 GiB or 4 GiB. There might not be a significant time difference. Below is an example of various Amazon web services.

TABLE IV: Different cloud instance with specifications and price

AWS	vCPU	Memory (GiB)	Linux/UNIX (per hour)
t2.micro	1	1	$0.013
t2.small	1	2	$0.026
t2.medium	2	4	$0.052
t2.large	2	8	$0.104

As can be seen in Table IV, an increase in specifications of CPU and Memory leads to increase in price, but on the contrary, it guarantees better performance. There is a trade-off for the organizations to decide upon. Assuming that an organization receives 15,000 malware samples per day, analysis of the computational structure for 100, 600 and 1800 samples and cost per hour is mentioned.

TABLE V: Cost per year for various cloud instances [20]

AWS	100 Samples, 150 instances per day for 1 hour	600 Samples, 25 instances per day for 1 hour	1800 Samples, 8.33 or 9 instances per day for 2 hours
t2.micro	$711.75	$118.63	$85.41
t2.small	$1423.50	$237.25	$170.82
t2.medium	$2,847.00	$474.50	$341.64
t2.large	$5,694.00	$949.00	$683.28

In Table V, the cost mentioned is the amount that the organization will be paying per year corresponding to the cloud instance used. As can be seen, the variation of prices for 100 samples is significant when compared to others. But the thing to realize is that the result of 100 samples will be provided within 6 to 10 minutes, depending upon the specifications that are selected. Hence, the key is to maintain a balance between time and cost. It is well-known that no one approach fits all, and will be dependent on the organization. The purpose is to provide an estimate of the economic and resource comparison.

The less computation time and cost-effectiveness of the cloud are the driving forces for researchers to focus on the cloud. Secondly, local data centers do not offer cost-effectiveness and pay-per-use service. Hence, it is a win-win situation from researchers' point of view.

VII. Conclusion

One of the central purposes of this work was to develop a tool that receives malware executables from the database that is maintained in the cloud and then perform statistical analysis on a broader and improved range for various clusters. Our research is still in phase I. The short-term goal is to develop a stable range for the top ten clusters, and the long-term goal is to maximize the detection rate at a large scale and help the anti-virus community to make a better product for malware detection named as Malware Secrets.

VIII. Acknowledgments

The research was supported by the National Science Foundation (NSF) Career Award CNS-1351038 and ACI-1642078. Further, thanks to UAB Malware Lab for providing malware samples and Sentar, Huntsville for the Cybergenome project.

References

[1] M. Santillan, "70software," Feb. 2015. [Online]. Available: https://www.tripwire.com/state-of-security/latest-security-news/70-of-malware-infections-go-undetected-by-antivirus-software-study-says/

[2] D. Dieterle, "Zeus botnet source code," May 2011. [Online]. Available: https://cyberarms.wordpress.com/2011/05/12/zeus-botnet-source-code-leaked-to-internet/

[3] MalwareBytes, "An overview of three zero-days." [Online]. Available: https://www.malwarebytes.com/threezerodays/

[4] T. T. Tens, "Best antivirus software companies." [Online]. Available: https://antivirus.thetoptens.com/

[5] G. Warner, M. Nagy, K. Jones, and K. Mitchem, "Investigative techniques of n-way vendor agreement and network analysis demonstrated with fake antivirus," *Proceedings of the Conference on Digital Forensics, Security and Law*, pp. 231–241, 2014. [Online]. Available: http://fetch.mhsl.uab.edu/login?url=http://search.proquest.com/docview/1626534277?accountid=8240

[6] "Virustotal." [Online]. Available: https://www.virustotal.com/

[7] M. G. Kang, P. Poosankam, and H. Yin, "Renovo: A hidden code extractor for packed executables," in *Proceedings of the 2007 ACM Workshop on Recurring Malcode*, ser. WORM '07. New York, NY, USA: ACM, 2007, pp. 46–53. [Online]. Available: http://doi.acm.org/10.1145/1314389.1314399

[8] K. Timm, "Malware validation techniques," Jul. 2010. [Online]. Available: http://blogs.cisco.com/security/malware_validation_techniques

[9] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.

[10] ALDEID, "Peid," Dec. 2013. [Online]. Available: https://www.aldeid.com/wiki/PEiD

[11] H. M. Jaenisch, A. N. Potter, D. Williams, and J. W. Handley, "Fractals, malware, and data models," pp. 84 080X–84 080X–16, 2012. [Online]. Available: http://dx.doi.org/10.1117/12.941769

[12] A. Awal, M. Shoeb, D. Mukhopadhyay, S. Noor, A. Sprague, and G. Warner, "Spam campaign cluster detection using redirected urls and randomized sub-domains," 2014.

[13] A. K. Marnerides, M. R. Watson, N. Shirazi, A. Mauthe, and D. Hutchison, "Malware analysis in cloud computing: Network and system characteristics," in *2013 IEEE Globecom Workshops (GC Wkshps)*, Dec 2013, pp. 482–487.

[14] WolframMathWorld, "Euclidean distance." [Online]. Available: http://mathworld.wolfram.com/Distance.html

[15] S. Skiena, "The stony brook algorithm repository," Jul. 2008. [Online]. Available: http://www3.cs.stonybrook.edu/~algorith/files/nearest-neighbor.shtml

[16] PortableApps.com, "Nsis portable." [Online]. Available: http://portableapps.com/apps/development/nsis_portable

[17] S. Chimakurthi, "Malware hides in installer to avoid detection," Aug. 2016. [Online]. Available: https://blogs.mcafee.com/mcafee-labs/malware-hides-in-installer-to-avoid-detection/

[18] G. Majauskas, "Browsefox - how to remove?" [Online]. Available: http://www.2-viruses.com/remove-browsefox

[19] Qubole, "Apache hadoop as a service." [Online]. Available: https://www.qubole.com/hadoop-as-a-service/

[20] Amazon, "Amazon ec2 pricing." [Online]. Available: https://aws.amazon.com/ec2/pricing/