Using an Information Retrieval Technique to Discover Malicious Software

Travis ATKISON
Department of Computer Science and Engineering, Mississippi State University
Starkville, MS 39762, United States

ABSTRACT

This paper describes a research effort to detect unknown, known or variances of known malicious software using an information retrieval technique known as cosine similarity. Document similarity techniques, such as cosine similarity, have been used with great success in several document retrieval applications. By following the standard information retrieval methodology, software, in machine readable format, is regarded as documents in the corpus. These "documents" may or may not have a known malicious intent. The query is a piece of software, again in machine readable format, which contains a certain type of malicious software. This methodology provides an ability to search the corpus with a query and retrieve/identify potentially malicious software as well as other instances of the same type of vulnerability. This retrieval is based on the similarity of the query to a given document in the corpus. The subsequent use of an information visualization technique will allow for quickly and clearly finding the malicious software and will provide the ability for finding similar, potentially new types or variances of malicious behavior.

Keywords: malicious software detection, information retrieval, n-gram analysis, cosine similarity

1. INTRODUCTION

In general, consumers depend on software development firms to deliver their software needs. This includes, but is not limited to, the operating systems on each machine, virus detection engines, and firewall systems. Even specialized applications for corporate business needs are often purchased rather than developed "in-house." The outsourcing of application development coupled with globalization of the software development market means that it is becoming more and more abstract as to where software is being developed and by whom. This presents a difficult security problem for the consumer in that developers of computing application may not have the same philosophies or views as the user of the application. An example scenario could be as follows, a virus detection developer provides a consumer with a detection engine that ignores certain viruses or reports that the machine is free of viruses. Another example could involve a firewall software developer manipulating the consumer's system to report back to an outside entity, without the consumer's consent or knowledge, information regarding network traffic data that passes through the firewall.

No one is immune from these malicious attacks; from the corporation to the unsuspecting home user, everyone is at risk. The motivation behind this research is to develop a more effective malicious software detection tool through the use of well known information retrieval techniques combined with a standard information visualization technique. The goal is to use information retrieval to detect malicious software, either as standalone applications or embedded within other applications.

The next section provides a short background description of information retrieval and discusses malicious software vulnerabilities. In Section 3, the experimental design is discussed including the software and data used. In Section 4, the results are described. Finally, in Section 5 the conclusion and future directions are presented.

2. BACKGROUND

Evaluating the effectiveness of using an information retrieval technique as a solution to part of the malicious software detection problem is an important direction in host security research. Below, the information retrieval technique and malicious software vulnerabilities used in our experiments are described.

2.1. Information Retrieval

Information retrieval traditionally is the "part of computer science which studies the retrieval of information (not data) from a collection of written documents." [3] These retrieved documents' aim is to "satisfy a user's information need." [3] The process can be thought of as combing through a set of documents, called the corpus, to find a certain piece of information that has a relationship to a given entity, called the query. That piece of information can either be an entire document, set of documents or a subset of a document. Within the information retrieval community several methods exist for finding these pieces of relevant information. These methods include vector space models, latent semantic indexing models and statistical confidence models as well as others. "Vector space models are the first approach to represent a document as a set of terms." [6] As their name implies vector space models represent their data as a vector with each dimension being defined as a term which may or may not have a weight associated with it. [9] One of the most common vector space models is cosine similarity. Cosine similarity determines the similarity between two data vectors by measuring the angular distance between them. "Cosine has the nice property that it is 1.0 for identical vectors and 0.0 for orthogonal vectors." [10] The following is the formula used in our work for computing cosine similarity;

$$\textit{Cosine Similarity}\ (Q, D) = \frac{\sum_i w_{Q,i} w_{D,i}}{\sqrt{\sum_i w_{Q,i}^2}\sqrt{\sum_i w_{D,i}^2}} \quad (1)$$

This formula computes the similarity between a query *Q* and a document *D*. It does so by summing the individual components of the two entities represented in the formula as *w*. The individual components for this research are defined as n-grams. An n-gram is "any substring of length *n*." [3] Here the gram (which will be the composite of the substring) is a byte in hexadecimal form. Therefore, w_{Qi} is the weight of the i^{th} n-gram in the query and w_{Di} is the weight of the i^{th} n-gram in the document.

There have been other efforts [1, 2, 4, 5, 7, 8] to use the information retrieval concept of n-grams as a potential for features. Henchiri et. al. [4] and Abou-Assaleh et. al. [1, 2] both use the Common N-Gram (CNG) analysis method, which uses the most frequent n-grams to represent a class, to detect rogue/malicious applications. Henchiri further limits the number of features by imposing a "hierarchical feature selection process". [4] Marceau [7] puts an interesting twist on the problem of using n-grams as features by having "multiple-length" grams instead of the tradition single n-length gram. Marceau does this by first creating and then compacting a suffix tree to a DAG. [7] Reddy et. al. [8] develop their own unique n-gram feature selection measure called, 'class-wise document frequency'

2.2. Malicious Software Vulnerabilities

Today malicious software vulnerabilities come in all "shapes and sizes," from buffer overflows to injection attacks to information leakage attacks. As noted above there are several instances of these vulnerabilities. Our concentration is on information leakage vulnerability attacks.

Information leakage can be defined as when "non-public" information is released (or leaked) without the information owner's knowledge or consent. An information leakage vulnerability can be introduced within an application at design time through malice or through poor programming practices (intentional versus accidental). It can also be introduced by a malicious attacker after deployment by being bundled with, or concealed within, a seemingly non-threatening application. Symantec reported in their bi-annual threat report for the first half of 2005 that "six of the top ten spyware (information leakage) programs were delivered to their victim by being bundled with some other program." [11]

There are several methods by which a malicious attack can induce a victim's computer to leak information without the knowledge or consent of the user. A notable example of this is the introduction of key stroke loggers into an application. A key stoke logger is software that will record every key that is typed on the user's computer. This research will not only look at key stroke loggers but also CD key stealers and password stealers. CD key stealers browse through the victim's computer registry looking for serial numbers for any CD's that the victim may have installed and registered. Password stealers work in a similar way but are geared specifically for detecting and extracting account passwords, such as AOL, Yahoo and MSN. Each of these stealers leaks its illicitly gathered information by packaging and sending it to the attacker through email or directly to an FTP server.

This research will concentrate on detecting malicious applications before execution while still packaged in their transporter. This transporter is often called a Trojan horse and the malicious package is referred to as a Trojan. A Trojan horse, similar to the myth, may provide a useful service (for example, a calculator or Notepad) but once executed performs harmful actions. We investigate a specific kind of Trojan horse known as a *binder* or *dropper*. Binders are applications that have the ability to combine (or bind) two or more applications together, yet allow them to run autonomously when executed. This autonomous nature allows the attacker to place a non-threatening, useful service together with one or more malicious applications. The unsuspecting user then executes the application excepting only the useful application; however, unbeknown to them the malicious application(s) are also executed.

3. EXPERIMENT

The following provides a description of the components of the experimental methodology we used. Described below are details of the software application that we developed as well as a description of the data set that was used in the experiments. This section concludes with an overall experimental design description that provides a description of how the experiments were conducted.

3.1 Similarity Software

The software created for this experiment provides functionality to ingest Windows formatted binary executables and then creates an *m*-dimensional data space that contains vectors representing those applications. In these experiments, *m* is the number of total possible n-grams that can be extracted from the ingested applications, one dimension for each possible n-gram. The information stored in each of the dimensions can take on one of several possible values; the absolute total number of occurrences of the particular n-gram in the application, the normalized value of the total number of occurrences of the particular n-gram in the application or finally a 1 if the applications contained the particular n-gram or a 0 if it did not. Once the *m*-dimensional vectors have been created the cosine similarity algorithm is applied to the query application's vector and the corpus applications' vectors. The cosine similarity algorithm followed is the same as shown in the Eq. (1) above. A special feature of this software is that it has the ability to shift the n-gram window not only by the more traditional byte offsets but also by bit offsets. This allows for a more fine grain tuning of the vector values, e.g., if the malicious adversary performs bit shifting on the malicious applications. It also provides for more accurate similarity result calculations.

3.2 Data

The data used for this experiment consisted of 267 Windows formatted binary executable files that were randomly chosen from a Windows XP operating system. These files ranged in size from 50KB to 500KB. Integrated within the corpus were 24 files that have been infected with malicious code using the F.B.I. (Finding, Binding and Infecting) binder and six standalone malicious applications for a total of 30 malicious applications. The Windows applications infected for this experiment were Microsoft Calculator, MS-DOS Command Prompt, Microsoft Notepad and Microsoft 3D Pinball for Windows. The malicious applications used were the CDKey Harvester v0.9, Fearless KeySpy v2.0, LttLogger v2.0, HermanAgent v1.0, ProAgent v2.0 and Recon v2.0. Each docile application was infected with each of the malicious applications using the F.B.I. binder to create 24 infected files. The binder and all malicious applications are freely available for download from the following website, http://www.trojanfrance.com. Table 3.1 contains short descriptions of the malicious applications used in this experiment.

Table 3.1. Descriptions of malicious applications

CDKey Harvester v0.9	searches victim's registry for Online Game CD Keys and sends them to the attacker through email
Fearless KeySpy v2.0	keystroke logger
LttLogger v2.0	keystroke logger that can completely remove itself at a specified time or after a specific amount of collection
HermanAgent v1.0	password stealer where information is passed back to the attacker through email
ProAgent v2.0	monitoring and surveillance tool that captures data from webcams, screenshots and microphone usage
Recon v2.0	keystroke logger that can disable anti-virus and firewall software

3.3 Design

This section describes the overall design of our experiment. In total six runs were accomplished, one for each of the malicious applications. A run is defined as having a malicious application labeled as the query and cosine similarity computed against the entire corpus of 267 applications. The results of these experiments are presented below.

4. RESULTS

As expected, if an attack occurred in the corpus the research software had no difficulty finding it when used as a query. For example, when using the Fearless KeySpy application as a query the software was able to find it. The most useful results achieved were that the software was able to find applications of similar malicious intent.

Through experimentation it was discovered that the software was able to detect malicious applications that are of the same malicious family as a given query and in some cases of a different but similar family. Using the Fearless KeySpy application as a query the software was also able to detect the ProAgent Application. The software was also able to detect certain other applications of keystroke loggers, specifically the Herman application and the Recon Agent application, using the ProAgent application and Fearless KeySpy application as queries. This appears to be promising as such software should be able to detect variants of specific types of malicious applications.

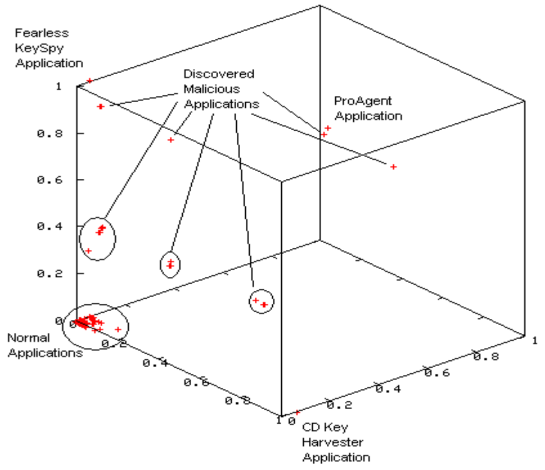

Figure 4.1. Discovery of malicious applications with CD Key and Password Stealers

Figure 4.1 shows the results of visualizing three CD Key and Password Stealer applications as queries against the entire data corpus. Notice the large cluster of glyphs, each one representing an application in the corpus, grouped around the origin. These applications have similarity scores that are near zero when compared to the query, meaning that these applications are not similar to the query. At the extremes of the axes we notice the three query applications themselves. These are not in the edges as one might expect. This is because there is a "pull" from the other query applications that moves them away from their corner. Other important items to note with this figure are the number of malicious applications discovered in the query. After investigation we confirmed that each one of these was indeed a malicious application, either standalone or embedded within another application using the F.B.I. binder. Several applications were discovered at or around the 0.4 similarity value with the Fearless KeySpy application query. The ProAgent application and the CD Key Harvester application were both able to discover themselves embedded within other applications.

Table 4.1 defines the performance values for the Key Stealer queries. These values include true positive rate (TPR), false positive rate (FPR), accuracy and precision. TPR, as known as recall, is the ratio of positive instances that were correctly identified. FPR is the ratio of negative instances that were incorrectly identified. Accuracy is the ratio of the number of positive instances, either true positive or false positive, that were correct. Precision is the ratio of predicted true positive instances that were identified correctly. These results are significant suggesting that we can maintain a high precision, at least 90%, without sacrificing accuracy or TPR. The methods used in previous research, mentioned in Section 2.1, report accuracy ratings ranging from 93% to 98%, so the results presented here are very comparable and in some cases outperform those other methods.

Table 4.1. Performance Values for Key Stealer Queries

Performance Metric	Threshold Values		
	0.5	**0.3**	**0.1**
TPR	0.5	0.9	0.93
FPR	0	0.004	0.01
Accuracy	0.94	0.99	0.98
Precision	1	0.96	0.9

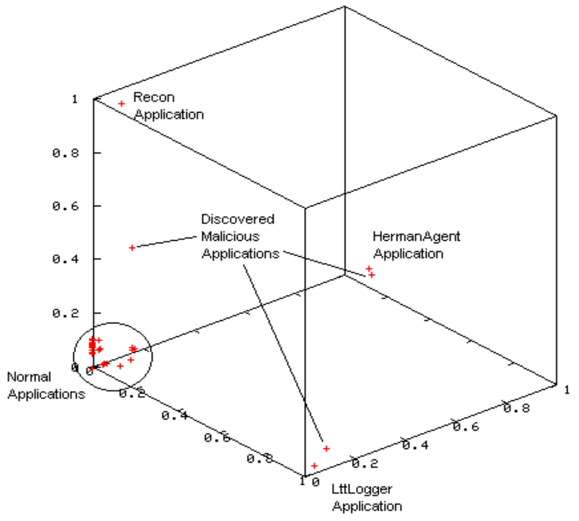

Figure 4.2. Discovery of malicious applications with Key Stroke Loggers

Figure 4.2 is not as impressive as the previous figure but still demonstrates the potential of this software in detecting malicious applications. This figure visualizes three key stroke loggers as queries against the entire data corpus. Notice again the large number of glyphs clustered around the origin. At the extremes of the axes, are the three query applications themselves. The "pull" induced by the other queries is seen as each glyph is not in the extreme corner of its representative query axis. For the discovered malicious applications, after investigation, it was determined that in all cases the query was discovering itself embedded within another application. The "pull" away from the true extremes is again explained as significant similarity value with respect to the other queries displayed in the figure.

Table 4.2 defines the performance values for the Key Logger queries. Again as can be seen on Figure 4.2 our accuracy and precision are high but our TPR is low for comparative threshold values to the previous experiment. This defines the point

Table 4.2. Performance Values for Key Logger Queries

Performance Metric	Threshold Values			
	0.5	**0.3**	**0.1**	**0.08**
TPR	0.37	0.5	0.5	0.97
FPR	0	0	0.008	0.06
Accuracy	0.93	0.94	0.94	0.94
Precision	1	1	0.9	0.67

of how threshold selection is important. By reducing our threshold by a small amount we were able to significantly increase our TPR at a negative cost to our precision. It is interesting to note that at the 0.08 threshold using only the LttLogger application we gained the following significant results; TPR = 0.97, FPR = 0, Accuracy = 0.99 and Precision = 1.

5. CONCLUSION

These results support our hypothesis that the cosine similarity metric has merit in determining if an application may contain a malicious application. It is easy to see the correlations between the given applications using the higher-dimensional space representation.

There is no claim that this is a complete solution, rather a tool designed to fit into the security administrator's toolbox as a data point or first pass to help reduce the number of applications needing review. This potential reduction in number of applications to sort through can provide an administrator or analyst with valuable time saving by not having to analyze applications that clearly do not contain malicious software. With more and more applications not being developed "in-house" this is a positive result for those responsible for providing secure solutions.

Future efforts with this research effort are to expand by the addition of prediction algorithms from the data mining realm, for example decision trees. Also the author plans to investigate additional dimensionality reduction methods and techniques in order to further expand and enhance the analysis capability. Additional research is also planned into determining the threshold values for the similarity algorithm. Determining the key factors in choosing an optimal threshold value is crucial, as can be seen above, to gaining high confidence and to the success rate of the algorithm.

6. ACKNOWLEDGMENTS

The author would like to thank Dr. Ray Vaughn and Dr. Ed Allen for their insight and thoughtful review. The author would also like to thank Chad Steed for his technical input with developing the three-dimensional figures in the document. Lastly, the author thanks Rebekah Atkison for reviewing this document for its grammatical content.

7. REFERENCES

[1] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "Detection of New Malicious Code Using N-grams Signatures," in *Proceedings of the Second Annual Conference on Privacy, Security and Trust*, New Brunswick, Canada, 2004, pp. 193 - 196.

[2] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "N-gram-based Detection of New Malicious Code," in *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.* vol. 2, 2004.

[3] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Harlow, England: Addison Wesley, 1999.

[4] O. Henchiri and N. Japkowicz, "A Feature Selection and Evaluation Scheme for Computer Virus Detection," *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pp. 891-895, 2006.

[5] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, and S. R. White, "Biologically inspired defenses against computer viruses," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intellgence*, San Francisco, CA, 1995, pp. 985 - 996.

[6] N. Liu, B. Zhang, J. Yan, Q. Yang, S. Yan, Z. Chen, F. Bai, and W.-Y. Ma, "Learning Similarity Measures in Non-Orthogonal Space," in *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management* Washington, D.C., USA: ACM Press, 2004, pp. 334 - 341.

[7] C. Marceau, "Characterizing the Behavior of a Program Using Multiple-Length N-grams," in *Proceedings of the 2000 workshop on New security paradigms* Ballycotton, County Cork, Ireland: ACM, 2000.

[8] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal in Computer Virology*, vol. 2, pp. 231 - 239, 2006.

[9] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, pp. 613-620, 1975.

[10] A. Singhal, "Modern Information Retrieval: A Brief Overview," *Bulletin of the Technical Committee on Data Engineering*, vol. 24, pp. 35 - 43, 2001.

[11] Symantec, "Symantec Internet Security Threat Report: Trends for January 05 - June 05," September 2005.