Malware Phylogeny Using Maximal πPatterns

Md Enamul Karim, Andrew Walenstein, Arun Lakhotia

Center for Advanced Computer Studies,

Lafayette, LA, U.S.A

and

Laxmi Parida

IBM T. J. Watson Research Center

About Authors

Md. Enamul Karim is a Ph.D. candidate at the Center for Advanced Computer Studies, University of Louisiana at Lafayette.

Contact Details: c/o Center for Advanced Computer Studies, #2 Rex Street, Lafayette, LA, 70504, U.S.A, phone 1-337-482-5791, fax 1-337-482-6766, email mek@cacs.louisiana.edu

Andrew Walenstein is a Research Scientist at the Center for Advanced Computer Studies, University of Louisiana at Lafayette.

Contact Details: c/o Center for Advanced Computer Studies, #2 Rex Street, Lafayette, LA, 70504, U.S.A, phone 1-337-482-5791, fax 1-337-482-6766, email walenste@ieee.org

Arun Lakhotia is an Associate Professor at the Center for Advanced Computer Studies, University of Louisiana at Lafayette.

Contact Details: c/o Center for Advanced Computer Studies, #2 Rex Street, Lafayette, LA, 70504, U.S.A, phone 1-337-482-5791, fax 1-337-482-6766, email arun@cacs.louisiana.edu

Laxmi Parida is a Research Scientist at IBM T. J. Watson Research Laboratory.

Contact Details: c/o IBM T. J. Watson Research Center, P. O. Box 218, Route 134, Yorktown Heights, New York 10598, Phone : +1 (914) 945-1376 Fax : +1 (914) 945-4104, parida@us.ibm.com

Keywords

Malware, computer virus, worm, program comparison, phylogeny, feature extraction, clustering, permutation, πpattern, obfuscation, software evolution

Malware Phylogeny Using Maximal πPatterns

Abstract

Construction of malware phylogeny could help in analyzing new malware samples as they arrive. However, the generated phylogenies must be accurate and be able to contend with the changes and obfuscations the malware writers create in the codes. We present our approach of using maximal πpattern, a PQ tree based feature, as a basis for comparing and classifying malwares. We argue that the πpattern approach is capable of dealing with certain obfuscations imposed in malware evolution process and demonstrate this possibility using examples of known viruses. We also suggest this scheme be used for automated naming of malware variants.

Introduction

Systematic code reuse has been an elusive goal for software engineering practice since the term was first coined, yet in certain respects it is an everyday practice for "malware" authors. The term "malware" here is being used as the generic name for the class of code that is malicious, including viruses, trojans, worms, and spyware. Malware authors use generators, incorporate libraries, and borrow code from others—there exists a robust network for exchange, and some malware authors take time to read and understand prior approaches (Arief & Besnard, 2003). Malware also frequently evolves due to rapid modify-and-release cycles, creating numerous strains of a common form. The result is a tangled network of derivation relationships between malicious programs.

In biology such a relationship network is called a "phylogeny"; significant recent efforts in bioinformatics involve automatically constructing meaningful phylogeny models based on information in nucleotide, protein, or gene sequences. Reconstructing malware phylogenies using similar techniques is expected to help in forensic malware analysis. It could provide clues for the analyst, particularly in terms of understanding how new samples relate to previously seen samples. Useful phylogenies could also serve as a principled basis for naming malware. Despite a 1991 agreement on an overall naming scheme and several papers proposing new schemes, malware naming continues to be a problem in practice (Raiu, 2002).

The question remains, though, as to how useful phylogeny models can be built by studying the bodies of malware samples. The method should be able to account for the types of evolutionary change that occurs in malware. Malware authors may try to hide the derivation relationships by several techniques, including garbage insertion, code reordering, and instruction substitution. Some metamorphic worms are known to modify their own code between generations; some shuffle the order of their code (Szor & Perrie, 2001).

We propose to use patterns of permuted code as comparison features for building malware phylogenies. In particular, we explore the use of so-called "maximal πpatterns". Methods based on maximal πpatterns have already proven to be promising in bioinformatics applications (Eres, Landau & Parida, 2003). These methods may also be well suited to application in malware phylogeny generation since malware can evolve through code rearrangements such as instruction reordering and code block shuffling. Matching potentially permuted sequences relaxes requirements for sequencing that are found in many other approaches for matching strings, including sequence alignment and large-*n* *n*-grams. In addition, the *maximal* πpattern approach formalizes a preference for large matches, a matching approach similar in spirit to the use of longest common subsequence (LCS) for finding minimal edit distances.

The following describes our approach of using maximal πpatterns and provides the results from a case study that suggests the automatically extracted phylogeny model grossly agrees to manual analysis.

A Maximal πPattern Approach

There have been several attempts to build malware phylogeny models. Goldberg, Goldberg, Phillips & Sorkin(1998) used 20-grams to generate directed acyclic graphs whose nodes are the viruses and whose edges map ancestors to descendants. They argued the sensibility of this approach by suggesting that malicious programs that are descendents of another are likely to have long byte sequences in common. Carrera & Erdelyi (2004) took a different approach of using a similarity measure based on static call graphs, and generating an X-tree using a hierarchical clustering algorithm. Wehner (2005) also used hierarchical clustering, but used an information theoretic similarity measure computed using a block-sorting compressor (bzip2).

These technique are expected to be at least occasionally problematic. The method by Carerra et. al. requires accurate call graphs, and these can be expensive and difficult to generate. The other methods rely on sequence information and, in general, we can expect methods that are strongly reliant on strict sequences to be suboptimal in cases where evolutionary steps involve significant permutations. For instance a rearrangement on a 20-byte sequence could hide a derivation relationship from large-*n n*-gram techniques. On the other hand small-*n n*-gram techniques are more likely to generate false positives.

It may be valuable, therefore, to have an alternative technique that could potentially capture longer collections of related words (as in LCS), but without absolutely requiring ordering (as in term-vector techniques like *n*-grams). One possible avenue for improving the matching of evolved code is to match on code that can possibly be permuted, i.e., rearranged. It would be desirable to allow for matching of long permutations, where they exist, yet also match smaller ones where they have been chopped up, moved around, and changed. The approach through maximal πpatterns matching is one possible approach.

Maximal πpatterns

Maximal πpatterns were introduced in (Eres *et. al.,* 2003) as a method for finding motifs in protein sequences. It is defined as a restriction on a collection of permutation patterns called πpatterns. Given an integer *K*, a pattern *p* is πpattern on string *S* if $|p|>1$ and *p* or its permutation appears at some $k \geq K$ distinct locations in *S*. Parameter *K* makes it possible to remove infrequent permutations, a filtering practice reminiscent of infrequent *n*-gram trimming. Nonetheless the number of possible πpatterns is $O(n^2)$, so without significant filtering, using πpatterns can be intractable for long sequences, or for matching within many files.

In response to this hurdle, the concept of a *maximal* πpattern was introduced. It is defined as follows: let *P* be the set of all πpatterns on string *S*. $p(a) \in P$ is non-maximal if there exists $p(b) \in P$ such that (1) each appearance of *p(a)* (or its permutation) is *covered* by *p(b)* (or its permutation), and (2) each appearance of *p(b)*on *S* covers a minimum of one *p(a).* A *p(b)* that is not non-maximal is maximal. An instance of pattern *q* covers an instance of *p* if *p* appears in *q*. For example in the string *S*=*abba* with *K*=2, the permutation of pattern '*ab*' appears twice (positions 0 and 2) and is covered by the permutation of pattern '*abb'*, which also appears twice. Thus '*ab*' is non-maximal and '*abb*' is maximal in *S* given *K=2*.

Maximal πpatterns are known to be equivalent to minimal consensus PQ trees (Eres, Landau & Parida, 2005). A PQ-Tree (Booth & Lueker, 1976) is a rooted, ordered tree with two types of internal nodes: P-nodes and Q-nodes. A P-node is known to be a consecutive block of elements, but with the order of the blocks unknown. A Q-node represents consecutive blocks appearing in a fixed order or exactly reversed. Effectively, the set of πpattern includes both *P* and *Q* nodes. Interestingly, since *P*-nodes are not ordered, code reordering is well captured in *P*-nodes. If inserted or deletion is performed in the code, P or Q nodes will be split; however, there will still be a match on effective codes.

Restricting focus to maximal πpatterns reduces the size of the feature space, yet searching for arbitrarily maximal πpatterns could be expensive. Eres *et. al* (2003) presented a two-phase algorithm to compute the collection of maximal πpatterns. The algorithm takes $O(Ln \log |\Sigma| \log n)$ time in the first phase to generate πpatterns of length $\leq L$ for a sequence of length n. The number of πpatterns ρ is bounded by $O(n^2)$. Assuming the maximum length of a location list is associated with a πpattern is l, phase 2 runs in $O(\rho^2 l)$ time. A newer, more efficient algorithm has also been developed for phase 2 in (Landau, Parida, & Weimann, 2005) which has a time complexity of $O(k\ m)$, given k permutations each of length m. Both of these algorithms are two stages and neither avoids the cost of first generating all of πpatterns.

We used a single stage algorithm based on the following observations: (i) if (P_b ***covers*** P_a) and (P_c ***covers*** P_b) then P_c ***covers*** P_a. and (ii) if P_b ***covers*** P_a then $|P_b| > |P_a|$. Our algorithm for finding all maximal πpatterns till length l proceeds as follows:

1. Construct an empty list, Max, of maximal πpatterns.
2. For w=n-K+1 downto l do :
   Slide a window of length w along the string and check if the pattern in the window is a πpattern and is ***covered*** by any larger pattern in Max , i.e., by any of the larger maximal πpatterns generated so far; if yes continue onto the next window, else add this to list of maximal πpatterns.

This stores only $O(n)$ πpatterns and the m maximal πpatterns at any instance instead of $O(n^2)$ πpatterns and reduces $O(\rho^2 l)$ time complexity to $O(\rho m l)$.

Phylogeny Model Generation Method

One way to use maximal πpatterns for building phylogeny models is as follows. First prepare the raw malware samples, if necessary (e.g., by filtering, disassembling, or abstracting), and then catenate the resulting strings together using unique separation markers. Then extract the collection of maximal πpatterns. This collection can be viewed as a set of non-redundant features contained in the malware samples modulo reordering. These can be used for feature-based model building or, if one defines a distance using the features, for distance-based model building.

Given the maximal πpatterns, the remaining issue is the method for phylogeny extraction. A simple method is to use the maximal πpatterns as binary features, represent the programs as binary vectors, and compute similarity as the number of bits set after exclusive-NOR-ing them. This technique has been used to construct phylogenies from gene sequences. The intuition behind using XNOR instead of, say, AND (which counts only features in common) is that both common features and mismatched features count towards these core. Call this the ``XNOR'' approach.

An alternative method is real-vector similarity measure, such as cosine similarity (e.g., see (Zobel & Moffat, 1998)). We have employed CLUTO (Karypis, 2003) to build phylogeny models using both XNOR and cosine similarity measures. CLUTO implements cosine similarity comparisons directly, and we provided it similarity matrices for clustering using XNOR similarity. We used its agglomerative clustering functionality to build dendograms, meaning the phylogeny models being generated cannot capture multiple inheritance relationships. Although in general this may be a limitation our experiences suggest the trees are a good and simple starting point for exploring the underlying maximal πpattern similarity technique.

Malware Naming

A useful naming scheme would ensure that as new malware samples arrive, the previously assigned names do not change. This requires stability in the tree, and implies that the initial tree should be built with a complete database of known malware. Here we provide a method for attempting to build and maintain a suitable phylogeny, although it falls short of actually generating human-readable names for the malicious programs.

To add an unknown virus *u* to the existing tree we follow the following algorithm,

- Find the best match to the stored list of πpatterns down to the leaf.
- If the match exceeds a threshold *k* at leaf *l*, do the following:
- Split *l* to two new leaves l_1 and l_2.
- Store *l* to l_1 and *u* to l_2.
- Store the consensus πpatterns of l_1 and l_2 at the node joining them.
- Name *u* as a new variation of l_1.

  Else, do the following:

- Add a new root r_n with two children: the old root *r* and a new leaf l_n.
- Store *u* to l_n.
- Store the consensus πpatterns of *r* and l_n at r_n.
- Assign *u* a new name.

Case Study

We performed a small study to investigate the potential in using maximal πpatterns, and to develop some initial feedback as to the relative merits of XNOR and cosine similarity measures.

Materials and Procedure

We selected several Windows-based malicious programs taken from an online database (vx.netlux.org) of malware. . We selected two worms and a virus and chose samples that were not encrypted or packed, and could thus be disassembled. We collected groups of samples known be in the same family so that we could evaluate how well family ties are reconstructed. 9 different samples of Win32.Alcaul (0-8), 3 of Win32.Belial (9-11), and 6 of Win32.Eva (12-17) were used. For three of the variants we also used a second disassembler, yielding 18 different code sequences.

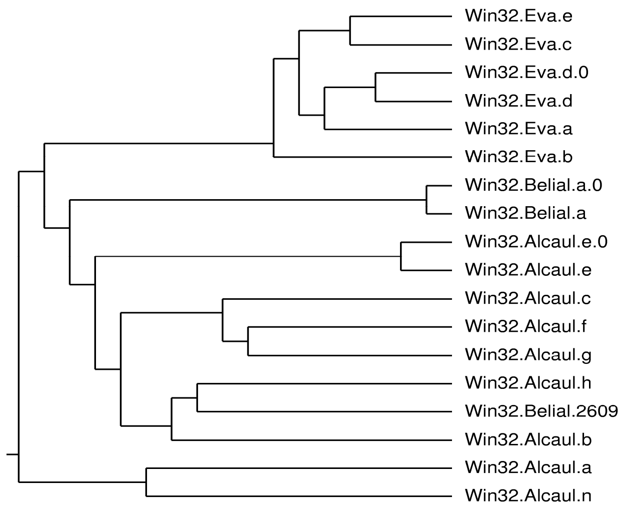

Figure 1a. Results using cosine similarity

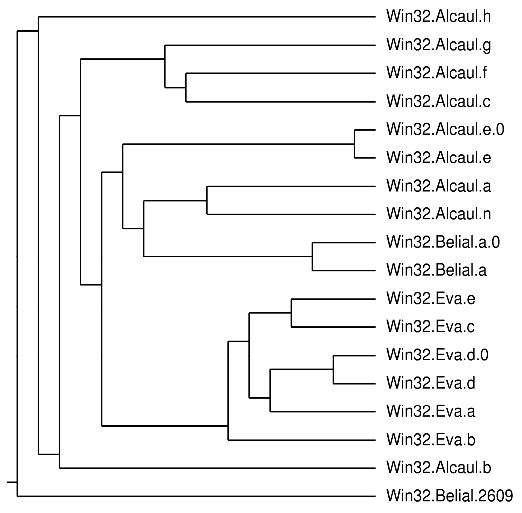

Figure 1b. Results using XNOR similarity

These worms were disassembled and only the opcodes were selected as characters. The character sequences representing the worms were then catenated together, and maximal πpatterns were extracted. The 18 samples generated 1570 maximal πpatterns.

Results

According to the existing naming, the phylogeny structure should look like, ((0-8),((12-17),(9-11))). Phylogenies we generated are shown in Figures 1a and 1b.

Discussion

In the figures, the multiple disassemblies (named with suffix ".0") of the same code are grouped by pairs as they should be. The phylogeny creates new questions to be investigated. It suggests Win32.Alcaul.h and Win32.Belial.2609 may have stronger phylogenic relationship through they are known to be two different species. It also shows that current variant sequencing may not be right, e.g., the phylogeny says Win32.Eva.a and Win32.Eva.d have a closer relationship than Win32.Eva.a and Win32.Eva.b or Win32.Eva.c and Win32.Eva.d. Hand investigation shows that our sample of Win32.Alcaul.h is significantly different from the other Win32.Alcaul variants.

The two different similarity measures do make a difference in the dendograms generated. For instance Win32.Belial.2609 is in an unrelated subtree in the XNOR dendogram (Figure 1(b)), whereas it is considered to be more closely related to Alcaul according to cosine similarity measures. Thus given that Win32.Alcaul.h is known to be substantially different, the XNOR appears to generate a more accurate phylogeny model, at least in this case.

Conclusions

We propose the use of maximal πpatterns for building phylogenies for forensic analysis, and provide efficient algorithms for generating the features for clustering. Our case study is limited and does not allow us to generalize about how well the techniques can be expected to work. Nonetheless the study suggests that permutation-based matching and, in particular, maximal πpattern-based matching may be a viable alternative to other phylogeny model generation methods.

The study also illustrates that the similarity methods used for phylogeny generation can make a difference to the quality of the results, and suggests that the XNOR approach of counting both matches and mismatches may have value for generating phylogenic models of malware evolution.

The study also suggests avenues for further investigation. In the present work we used exact matching of πpatterns, i.e., in order to match two πpatterns should have exactly same set of elements. We like to investigate how approximate matching of πpatterns affects the making of the tree/evolutionary relationship. We also need to study on determining the right value of the threshold to decide whether a virus is a variant or is independently developed.

References

Arief, B. & Besnard, D. (2003). Technical and human issues in computer-based systems security. University of Newcastle upon Tyne. (CS-TR-790).

Booth, S. & Lueker, S. (1976). Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13. pp. 335–379

Carrera, E. & Erdelyi, G. (2004). Digital genome mapping: advanced binary malware analysis. Virus Bulletin Conference, pp 187-197, Sept 2004

Eres, R., Landau, G. M. & Parida, L. (2003). A combinatorial approach to automatic discovery of cluster patterns. In G. Benson & R. Page (Eds.), Algorithms in Bioinformatics: Third International Workshop (pp. 139-150). Berlin: Springer-Verlag.

Goldberg, L. A., Goldberg, P.W., Phillips, C. A. & Sorkin, G. (1998). Constructing computer virus phylogenies. *J. of Algorithms*, 26(1), pp.188-208.

Karipys, G. (November, 2003). CLUTO: A clustering toolkit, Release 2.1.1. University of Minnesota (Report #02-017).

Kolter, J. Z. & Maloof, M. A. (2004). Learning to detect malicious executables in the wild, SIGKDD 2004. Seattle, WA, USA. ACM Press.

Landau, G. M., Laxmi, P. & Oren, W. (2005). Gene proximity analysis across whole genomes via PQ trees. Retrieved 15 March, 2005 from http://cs.haifa.ac.il/LANDAU/gadi/LPW.pdf

Raiu, C. (2002, June 3). A virus by any other name: Virus naming practices. *Security Focus*. Retrieved 5 March, 2005 from http://www.securityfocus.com/infocus/1587

Szor. P. & Ferrie, P. (September 2001). Hunting for metamorphic. In Proceedings of the Virus Bulletin Conference (pp 123-144).

Wehner S.(2005). Analyzing Worms Using Compression. Retrieved 5 March, 2005 from http://homepages.cwi.nl/~wehner/worms

Zobel, J. & Moffat, A. (1998). Exploring the Similarity Space. SIRIG Forum, 32(1), pp. 18-34.