Computer Sciences Department

Characterizing Malcode Evolution

Archit Gupta
Pavan Kuppili
Aditya Akella
Paul Barford

Technical Report #1599

August 2007

UNIVERSITY OF WISCONSIN MADISON

Characterizing Malcode Evolution

Archit Gupta, Pavan Kuppili, Aditya Akella and Paul Barford*

ABSTRACT

The diversity, sophistication and availability of malicious software (malcode) pose enormous challenges for securing networks and end hosts from attacks. In this paper, we analyze a large corpus of malcode meta data compiled over a period of *19 years*. Our aim is to understand how malcode has evolved over the years and in particular how different instances of malcode relate to one another. We develop a novel graph pruning technique to establish the underlying relationships between different instances of malcode based on temporal information and key common phrases identified in the malcode descriptions. Our algorithm enables a range of possible inheritance structures, which we investigate through extensive manual validation. The resulting "most likely" malcode family trees show unique structure and diverse characteristics. We present an evaluation of gross characteristics of malcode evolution and also drill down on the details of the most interesting and potentially dangerous malcode families.

Our approach is not definitive and could be improved given better meta data. Nevertheless, it is our hope that this new perspective on malcode evolution will be of great help in the development of more effective defenses in the future.

1. INTRODUCTION

Malicious activity in the Internet is growing at an alarming rate. There are daily reports in the technical and popular press about new vulnerabilities and new types of attacks, and the rapidly increasing economic incentives are sure to catalyze this activity for a long time to come. Well known examples of malicious activity include denial of service, spam, information gathering, and resource gathering. In all cases, this activity is based on the use of software (malcode/malware) that enables attacks to be carried out from one or more hosts distributed throughout the Internet.

Creating effective countermeasures for these threats is fraught with challenges. First, the ever increasing complexity of inter-networked systems and software means that it is extremely difficult to build them to be inherently free from vulnerabilities. Second, it is well know that security is not an intrinsic feature of Internet architecture. The consequence is that security is realized as a patchwork of add-on software, features and capabilities that are unlikely to ever close all opportunities for intelligent and determined attackers. Third, and perhaps most significantly, the authors of malcode are well aware of the details of network and end host security mechanisms, and are developing increasingly sophisticated and effective methods for subverting these defenses. This is an arms race, and it is not difficult to argue that the good guys are losing.

One of the most common methods for defending against attacks is to install anti-virus (AV) software on end hosts. All AV software is based on using a set of signatures that identify attacks and intrusions from malicious software. When these signature sets are kept up to date, AV software provides a high level of protection to end hosts. The process of creating signature sets is central to all AV companies (likewise for signature-based network intrusion detection/prevention systems). It is based in large part on capturing malicious binaries used in Internet attacks using honeypots [1] and by sharing captured binaries with other companies. While AV companies are able to generate signatures quickly after instances of malcode have been captured, it is still a *reactive* process that gives the advantage to the attackers.[1]

The AV companies' measurement activity is almost always coupled with an annotation process that takes place after a signature has been developed to detect a new instance of malcode. This meta-data is used for internal documentation and is typically made available to the community through web sites (*e.g.*, [17, 26, 25]). This meta-data represents a rich source of information on malcode that to date has remained largely unexamined by the research community.

In this paper, we investigate the details of a large corpus of malcode using McAfee's threat library database, which has been compiled over the past 19 years. Our data-set covers very diverse malware which differ widely in their level of sophistication, potency, methods of spreading etc (some of the malware we study are known to have spread via floppy diskettes). To the best of our knowledge, ours is the *first-ever* long-term empirical study of the evolution of malware.

Our specific interest is in understanding the *evolving relationships* between different instances of malcode over time; In other words, we are interested in identifying and studying malcode families. Examples of characteristics that can reflect relationships of interest include targeting a common vulnerability, using a common method for scanning or denial of service and, especially, sharing specific pieces of code – a practice that is widely known to be common in malcode development. Our challenge in establishing these relationships is that we are working with meta-data that *describes* specific instances of malcode and not with the malcode itself (there

*Computer Science Department, the University of Wisconsin - Madison. E-mail: archit,pavan,akella,pb@cs.wisc.edu.

[1]Recent proposals such as vulnerability-based signatures [29] and semantic-aware signatures [31] offer some opportunities to reduce the attackers advantage.

are also well know challenges in working with malcode binaries). Thus, identifying specific instances of, for example code sharing, is impossible unless it is somehow identified in the description. A further challenge is that most of the descriptions are entered in plain English sentences, and hence they are *unstructured*.

We develop a novel method for evaluating malcode meta-data that has two components: text mining and graph pruning. The first component identifies the frequent phrases in the textual descriptions of all instances of malcode in our data set. This process not only enables suppression of common phrases (*e.g.,* "The following entry"), but also exposes the key features of malcode as expressed in specific sequences of text. Thus, it imposes structure on the unstructured textual descriptions. This immediately makes several analyses possible - for example, we can now develop metrics to estimate the similarity between two malware instances.

The second component of our evaluation methodology begins by considering all instances of malcode as a fully connected graph. We then prune the edges of the graph based on phrase/word similarities using two different threshold parameters. The graphs that result after the pruning process reflect both the *temporal and feature-based relationships* between instances of malcode.

In our analysis, we explore the trade-offs of using different pruning threshold values. Fortunately, the analysts who document the malcode also label many instances with names that indicates membership in a particular family (*e.g.,* W32/Bagle.n@MM). These labels provide a means for validating the family trees that result from our analysis. Using this approach, we identify a threshold parameterization that results in what we argue is a "mostly likely" set of families.

Our graph generation mechanism unearths 702 malware families. The details of the families are fascinating. We find instances of families that are very short lived, and others that persist for years. We find families that have a large fanout *i.e.,* many children after the root (frequently the case for well known malcode) and others that are quite narrow. One of the most interesting aspects of our analysis is how new malcode families evolve from old families – we provide an in-depth characterization of this phenomenon as well.

The long-term objective of our study is to contribute to building a foundation for Internet security systems that are *robust and proactive* by expanding the perspective on malcode. It is our hope that analyzing the evolving families of malcode may help to guide efforts at building defense mechanisms in much the same way that virologists attempt to identify the most likely flu strains on an annual basis.

The remainder of this paper is organized as follows. In Section 2 we discuss prior work related to our study. Details of the malcode meta-database and our analysis methodology are described in Section 3. The results of our evaluation are presented in Sections 4 and 5. We summarize our study and discuss future work in Section 6.

2. RELATED WORK

There are a growing number of empirical studies of malicious activity in the Internet. Well know examples of these include [22, 21, 19, 20, 30, 23, 14]. These studies are often focused on a particular segment of malicious activity such as denial of service attacks or worm outbreaks, and the reports frequently coincide with the emergence of new threats. More recently, Freiling *et al.* [12] and Rajab *et al.* [24] provided empirical details on the escalation of botnet activity – one of the most potent threats in the Internet today. In all cases, these studies take advantage of a particular measurement infrastructure such as Dshield.org [28] or distributed honeypots [1] as a means for gathering data, and typically provide statistical characterizations of the data. Similar to these studies, the data analyzed in our study is based on 24x7 malware gathering and annotation conducted by AV companies. However, we only rely on meta-data not actual malcode (source or binaries) themselves.

Evaluating the details of malcode binaries once they have been captured can provide interesting insights. However, it is challenging because the authors are increasingly using techniques to confound this analysis. Disassemblers, debuggers and system monitors are common tools used by AV companies to generate signatures and create the documentation used in our study [6, 4, 3]. An alternative is to evaluate malware source code which can sometimes be found on the Web or in Usenet newsgroups. A recent example of source code analysis of bots can be found in [9].

The notion of considering Internet malware in a biological and immunological context has a relatively long history (*e.g.,* [10, 13, 11]). Excellent, comprehensive reference material on malware can be found in [27]. Ma *et al.* present a study more closely related to our own that infers the phylogeny *i.e.,* behavior characteristics of malware shellcode [16]. Our work is most similar to these studies in that we too aim to establish evolutionary relationships between malware.

While the overall objective of finding relationships in malware is similar to our own, the above studies differ in the data and algorithmic mechanisms they employ. Another key difference is that our study provides a very "long-term" view into the evolution of malware. Our work is unique in that it offers several insights into a multitude of malware families and their evolution patterns.

From an algorithmic stand-point, our study is informed by prior work in data and text mining. Zaki describes an efficient algorithm for identifying frequent sequences in large data sets [32]. While efficiency is less of an issue in our work, methods for finding all frequent sequences such as those described in [8] are important. Temporal text mining studies are particularly relevant. For example, Mai and Zhai present a temporal text mining algorithm used to identify "themes" and then apply it to news articles (thereby creating thematic graphs) [18]. Finally, Li *et al.* use text mining to explore the bugs related to "copy and paste" in Linux and

FreeBSD variants in [15]. Copy and paste is also common in malware variants. However, without source code, it is difficult to apply the same method of analysis.

3. MALWARE FAMILIES

Our dataset contains a wealth of information on a large number of malware instances. Almost all of this information is entered by hand, after capturing and deconstructing the malicious code in a controlled environment. Since the information is entered manually, it is semantically rich and quite detailed. However, not all malicious code is described to the same extent of detail or using the same English constructs!

In this section, we begin by providing details on the malware dataset. We then describe the first component of our analysis, which creates a uniform schema for malware descriptions based on *text mining*. The schema provides a baseline from which we can establish relationships between malware instances. Next, we describe our *graph pruning algorithm* that we use to generate "likely" malware family trees.

3.1 Description of the Dataset

The malware meta data that we evaluate is the McAfee Avert Labs Threat Library [17]. There are 44,504 malware instances in the database, spanning a period of 19 years from 1987 to 2006. The database schema is quite exhaustive and is summarized in Table 1. The schema provides a variety of information - e.g., the malware name, discovery date of the malware, size of the malware (in bytes), malware type, "danger" of payload, prevalence etc. However, from the point of view of inferring relationships between malware, the most useful fields are the ones with rich textual descriptions. We focus on three such fields - malware characteristics, methods of infection, and indications of infection. A total of 8,182 malware instances include detailed text descriptions in these three fields. Therefore, we focus our efforts on these 8,182 instances.

Gleaning relationship information from free text (generated by many different people over time) is quite challenging, and we describe our approach in more detail below.

3.2 Challenges in Mining the Data

The textual descriptions for the malware provide a wealth of "interesting" information. However, these descriptions carry no specific structure or organization. Thus, we are faced with the challenge of systematically separating the wheat from the chaff in these textual descriptions, with as little human input as possible, and uncovering the most informative properties of malware. This is crucial in order for us to be able to infer the relationships between malware with high confidence. Our first insight is that we can use techniques from Information Retrieval to mine the most important properties as well as to establish if two malware are "related" to one another. We elaborate on this in Section 3.3.

A second insight we use is that the information obtained from unstructured data when combined with other structured information can provided interesting views into the evolution of malware. Note that structured data can be exploited much more readily than the unstructured text. As an example, we use the discovery date to "orient" the edges in our malware relationship graph and to indicate the possibility that the newer of the two malware was spawned from the older one. We describe this in Section 3.4.[2]

3.3 Schema Discovery Via Text Mining

In order to make the unstructured textual descriptions more useful toward the construction of malware relationships, we attempt to impose a suitable schema and convert the textual descriptions into a set of tuples that follow the schema. Each piece of malware is then mapped onto a tuple, each field of which describes a key property. With this in place, we can then define a *similarity function* over the tuples which quantifies the relationship between malware.

To obtain the appropriate schema from the textual descriptions, we use *frequent phrase extraction*. We consider phrases which occur frequently throughout all textual descriptions and make each such phrase a column in the schema. Thus, each malware is represented by a tuple in the schema and the columns take boolean values indicating the presence or absence of the corresponding phrases in the description for the malware.

We used a frequent phrase extraction tool from the IR community, Extrphr32 [2], which extracts the *maximal* frequent phrases: a maximal frequent phrase is not a substring of any other frequent phrase. The tool also considers only those phrases which do not start or stop with *stopwords*: these are common English words like "the", "but" etc. Accounting for stopwords allows us to weed out common phrases which do not carry much significance. To determine if a phrase is frequent or not, we use a tunable parameter σ. Phrases which occur fewer than σ times are considered infrequent and hence not useful.

We used the frequent phrase extraction approach on the "virus characteristics", "methods of infection", and "indications of infection" text columns. We tuned the threshold parameter and finally ended up with ≈ 6500 frequent phrases. Not all of the resulting phrases were informative, however. To guard against inferring spurious relationships based on these phrases, we manually weeded out the phrases which did not seem to have much significance or did not clearly express a possible relationship with other malware (these include generic single-word adjectives and verbs, and other short phrases such as "following text", "application is designed", and "description is meant").[3]

After this manual pass, we were left with ≈ 5300 phrases.

[2]Our analysis completely ignores other structured fields, such as the potency and the prevalence of malware. This is because we want to focus primarily on the understanding how the different malware families evolve over time.

[3]Note that this process could be automated.

Fields/Attributes	Type of Data	Example	Comment
virus_k	Structured	133997	Primary identifier in Database
virus_name	Unstructured	W32/Mytob.be@MM	Name of the malware
length	Structured	49,278 bytes	Size of the binary of the malware
discovery_date	Structured	2005-05-26 00:00:00.000	Date of discovery of malware
date_added	Structured	2005-05-26 13:37:35.060	Date malware was added
remove_k	Semi-structured (411 possible values)	"All Users: Use current engine and DAT files for detection and removal ..."	Instructions for removing malware
virus_type_k	Semi-structured (11 possible values)	Virus	Type of malware
virus_subtype_k	Semi-structured (202 possible values)	Email	Sub type of malware
danger_of_payload_k	Structured (Range: [1..5])	0	5 implies the payload is dangerous 0 implies no information
prevalence_k	Structured (Range: [1..4])	0	The prevalence of the malware. 0 implies no information
virus_characteristics	Unstructured	"This detection is for a mass-mailing worm that combines W32/Mydoom@MM functionality with W32/Sdbot.worm functionality..."	Description of malware
method_of_infection	Unstructured	"... Finally the virus sends itself via SMTP constructing messages using its own SMTP engine. The worm guesses the recipient email server, prepending the target domain name with the following strings ..."	Method of infecting a host
indications_of_infection	Unstructured	"The Sdbot functionality in the worm is designed to contact the following IRC server, join a specified channel, and wait for further instructions: irc.blackcarder.net (on TCP port 7000)... "	Indications that a host is infected

Table 1: A summary of the McAfee dataset. A subset of the fields of the database are described along with their structuring and an example.

A few examples of the phrases that were used in our schema follow:

> "This virus constructs messages using its own SMTP engine. Target email addresses are harvested from files"
> "memory resident at the top of system memory but below the 640k dos boundary, hooking interrupt 21."
> "Its spreading activity remains only in the german language version of microsoft word. However, the virus may be able to execute its payload in another language version."

Using the obtained schema, we create and populate a table of all malware.

3.4 Constructing the Graph

Our goal is to derive a graph G', whose vertices are the malware in our dataset, and whose edges accurately reflect the true underlying relationship between pairs of malware. We derive G' using a multi-stage graph pruning approach. As we argue in Section 3.5, the graph G' is actually a collection of multiple different *malware families*.

As mentioned earlier, each malware has an associated "time of discovery" field. This is a crucial piece of information which we use along with the database of frequent phrases, to mine the underlying relationships and to understand malware evolution. We first start with a completely connected graph G with directed, weighted edges: each vertex in this graph is a piece of malware, and the weight of an edge is the similarity between the sets of frequent phrases associated with the malware. Edges point from the older malware to the more recent one. The similarity metric ranges between 0 and 1 and is defined as follows: For an edge $A \rightarrow B$ between two malware A and B, let $S(A \rightarrow B)$ be the set of properties in the above frequent phrase database which are common between A and B. The weight of the edge is $\frac{|S(A \rightarrow B)|}{|B|}$, where $|B|$ is the number of defined properties in B and $|S(x)|$ is the cardinality of Set $S(x)$. Thus, the similarity metric captures how many of A's properties are also shared by B.

Note that if $|B|$ is very small, then we do not have sufficient information to relate it to other malware. To prevent such malware from corrupting our malware relationship graph, we introduce a threshold γ. If $|B| < \gamma$, we do not consider B in creating the graph G'. We set $\gamma = 10$, which we admit is an arbitrary choice, and could still introduce spurious relationships. However, as we argue below, our conservative graph pruning algorithm is designed to further prune such edges.

Overall, we do not consider 2174 malware instances in our analysis because they do not satisfy the γ threshold.

When are two pieces of malware "related"? In our analysis, we use the term "relationship" to mean that that the two malware in question share several important characteristics, such as method of infection, method of spreading, and the symptoms exhibited by infected hosts. Based on anecdotal evidence [5], we speculate that such similarities arise due to one or both of the following two reasons: (1) The authors of the malicious code share some routines with each other. This practice of code sharing is known to be common in the black hat community. Or, (2) The authors of the malware exploit the same set of vulnerabilities in an operating system, use the same vector for spreading, etc, but there is no explicit code sharing. While there could be other reasons, we argue that identifying relationships on the basis of which properties are shared between malware is important in order to design effective defenses and to contain future malware strains in a more timely fashion.

Graph construction. We now describe a simple, multi-stage technique for systematically deleting insignificant edges from the fully-connected graph G to obtain the underlying relationship graph, G'.

We define an edge A→B as "good" if the weight of the edge is above the threshold δ_1. When $|A|$ and $|B|$ are sufficiently high, then a "good" A→B edge indicates exactly what we are looking for: that B shares several key features with A, and it is likely to be derived from, or to be otherwise closely related, to A. In our relationship graph, we refer to malware which have a good edge incident on them as "good nodes". Thus, the first stage of our pruning algorithm is simple: we just remove all non-good edges from G.

Now consider the case when node C has two "good" incoming edges from malware A and B. It might be the case that A and B are themselves very similar to each other, and that C actually evolved from A. Thus, we must delete the incoming edge from B.

To enforce this, we use another threshold δ_2 in the second stage of our graph pruning. If an edge does not *uniquely* contribute more than δ_2, we prune the edge. Specifically, say a node C has a pair of incoming edges $A \rightarrow C$ and $B \rightarrow C$. We check if $\frac{|S(A\rightarrow C)-S(B\rightarrow C)|}{|C|} < \delta_2$ (the numerator in the inequality computes the set difference). If this inequality is satisfied, we delete the incoming edge with the lower weight.

It turns out that this two-stage pruning is overly aggressive; the graph we end up with after the pruning—G''—is a subgraph of our target G'. This is because G'' only has good edges. Thus, it misses out on some key relationships between malware: For example, a certain malware may have no significant parent in G'' (i.e, there is no incoming good edge), but it may still share important properties with multiple other malware—in a sense, it may have "evolved" from multiple parents. Such "bundling of threats" has in fact been observed in the wild (e.g. the creation of the `Mytob` worm from earlier variants `Mydoom` and `SDBot`) [5]. It is important that our target G' capture such relationships.

To accommodate situations of this kind, we revisit the original complete graph G and consider those edges in G whose weights lie between δ_2 and δ_1. We call these "fuzzy" edges. We selectively add a small subset of these fuzzy edges back to the graph G'' to obtain the target G', as described next.

If a piece of malware has a good edge incident on it, then we prune all the fuzzy edges incident on it. Furthermore, we prune out the fuzzy edges whose unique contribution is less than δ_2. From the remaining fuzzy edges incident on a malware, we take a minimal subset whose total contribution is greater than δ_1. We delete all the other incoming edges.

Note that δ_1/δ_2 is the maximum number of incoming fuzzy edges on any malware. We refer to malware which have one or more fuzzy edges incident on them as "fuzzy nodes". If, for a certain malware, the total contribution of all fuzzy edges is less that δ_1, then it is likely that the malware evolved fairly independently.

The careful reader may note that our approach for adding the fuzzy edges is very conservative. This is a deliberate choice we made - while fuzzy edges are useful, we don't want them to create spurious evolutionary relationships in our malware families. After including the fuzzy edges selected in this manner, we finally have the graph G'. For completeness, we provide a formal description of our graph construction algorithm in Figure 3.4.

3.5 From the Relationship Graph to Malware Families

The final graph G' contains several connected *components*. Each component is a directed acyclic graph. Furthermore, each component will contain one or more malware with zero edges incident on them. We refer to such malware as *roots*. Most components will have a single root. We refer to such components as *trees* or *malware families* or *family trees*. Henceforth, we use these three terms interchangeably.

Whenever a component has more than one root, it is because fuzzy nodes and fuzzy edges exist at the lower levels of the component. In such cases, we "split" the component into as many malware families as the number of roots. We assign each fuzzy malware to the family tree in which it appears earliest according to a depth first search starting at the root of the family. We break ties in favor of the family which contributed the heaviest edge incident on the malware.

We perform this decomposition of our relationship graph G' into malware trees only because it allows us to simplify our analysis and speak of "most likely" malware families. With this decomposition, we can now study key properties of the likely families, such as the total number of malware, the life-span of the family, the total number of generations etc.

It is possible that malware instances that are classified into different families by our algorithm are actually related. Indeed, when we examine the resulting families closely (more in Section 5), we noticed that several well-known malware instances (e.g. instances of `Bagle` malware) are spread

Part-I: Good Edges - Obtaining G''

For any threshold δ_1:

1. Mark all edges with weight greater than δ_1. Call these "good edges".

2. For a node, if we have multiple incoming good edges (say k in number)

Look at all the ${}^kC_2$ combination pairs of incoming edges.
For every pair x, y of incoming edges,
if $weightminus(x, y) < \delta_2$
Delete either x or y
(delete the one which has a lesser weight).

Part-II: Adding fuzzy edges back to G''

For each threshold δ_2:

1. Mark all edges with weight greater than δ_2 but less than δ_1. Call these "fuzzy edges"

2. If there exists an incoming good edge (weight greater than δ_1) for a node, remove all the incoming fuzzy edges for that node.

3. For a node C, if there is is no incoming "good edge"

Look at all the ${}^kC_2$ combination pairs of incoming fuzzy edges (there are k incoming fuzzy edges).
For every pair x, y of edges, if $weightminus(x, y) < \delta_2$, delete the edge with the lesser weight.
Sort the remaining fuzzy edges in the descending order of their weights and store the result in a queue Q.
Let P be an empty set.
while(Q is not empty AND $weightunion(P) < \delta_1$)
Dequeue an edge from Q and add to P.
if($weightunion(P) < \delta_1$)
delete all incoming edges of C
else
delete all incoming edge of C which remain in Q.

Figure 1: Constructing the malware Relationship graph. If a node C has a pair of incoming edges x and y, define weightunion(x,y) as $\frac{|S(x) \cup S(y)|}{|C|}$ and weightminus(x, y) as $\frac{|S(x) - S(y)|}{|C|}$.

across multiple family trees in our classification. However, our classification ensures that members within a tree are much more likely to be strongly related to each other than members across trees.

Finally, we stress that the malware in a family may not necessarily have been derived from one another. Rather, all family members share a collection of important properties. Understanding the evolution of the shared properties across the generations of a family may prove instrumental in designing effective malware defenses for the entire family or for future strains emerging from the family.

3.6 Parameter Choice

Our algorithm uses three key parameters δ_1, δ_2 and σ. The eventual quality of the malware families hinges crucially on how these parameters are chosen. A permissive setting for these parameters (low values for all) could cause us to infer relationships where there are none. An overly conservative choice may cause us to miss important relationships.

Since we only have access to malware metadata and not the actual malware, it is hard to tell if a particular choice of parameters is conservative or not. In our study, we err on the side of caution and choose a parameter setting that is as conservative as possible while not causing the malware families to become completely degenerate.

We first study how to select the two δs. In Table 2 we illustrate the effect of 20 different parameter choices on the entire relationship graph. The candidate choices for δ_1 lie on the more conservative side. This allows us to identify several "good" relationships accurately. Our choices for δ_2, on the other hand, are more permissive and this allows us to include edges that we may have otherwise ignored due to our conservative setting for δ_1.

We look at four features of the relationship graph: the overall number of good and fuzzy nodes, the number of roots and the total number of nodes that do not belong in any family (i.e. isolated nodes). First, as expected, the number of good nodes decreases with higher values of δ_1 and is unaffected by δ_2. The number of fuzzy nodes drops drastically with δ_2 and becomes insignificant when $\delta_2 > 0.3$. The total number of roots does not show a clear monotonic trend as it has a complex dependence on both δ_1 and δ_2. On the other hand, the total number of isolated nodes generally increases with δ_1, with the total number at $\delta_1 = 0.9$ almost 50% higher than the number at $\delta_1 = 0.7$.

Since our goal is to have as few isolated nodes as possible (to prevent the families from becoming degenerate), while keeping our parameter choice as conservative as possible (to eliminate spurious edges), we chose $\delta_1 = 0.7$ and $\delta_2 = 0.3$. We note that a few different parameter choices (e.g. (δ_1, δ_2) = (0.6, 0.3), (0.7,0.3)) provide roughly similar trade-offs. We leave a more detailed sensitivity analysis for future work.

We now turn our attention to σ which specifies the minimum number of times a phrase should appear in the text describing the malware in order for the phrase to be considered useful. We tried various values of σ to get a sense of the quality and the number of frequent phrases we obtain. A higher value of σ brings out the most important phrases and characteristics that are common to a lot of malware instances. A low value of σ, on the other hand, brings out more selective and precise information, which is common to a smaller number of malware. At first, a low value of σ alone looks promising. But a higher value of σ may also be useful because it allows us to relate instances of malware in situations where not all malware are described to the same thorough extent. In our approach, we decided to use a bimodal σ setting. We took two sets of phrases extracted with $\sigma = 30$ (low setting) and $\sigma = 100$ (high setting) and merged them (after performing manual pruning on each set), where merging means that two phrases describing the same information were considered only once.

3.7 Validation

$\delta_1 \rightarrow$ $\delta_2 \downarrow$	0.6	0.7	0.8	0.9
0.1	3619	2741	1803	941
0.2	3619	2741	1803	941
0.3	3619	2741	1803	941
0.4	3619	2741	1803	941
0.5	3619	2741	1803	941

(a) Num. good nodes

$\delta_1 \rightarrow$ $\delta_2 \downarrow$	0.6	0.7	0.8	0.9
0.1	1646	1837	1612	931
0.2	404	348	220	100
0.3	61	58	36	18
0.4	3	3	3	2
0.5	0	0	0	0

(b) Num. fuzzy nodes

$\delta_1 \rightarrow$ $\delta_2 \downarrow$	0.6	0.7	0.8	0.9
0.1	530	949	1481	1638
0.2	735	869	824	573
0.3	617	702	631	458
0.4	581	659	599	435
0.5	582	659	598	433

(c) Num. roots

$\delta_1 \rightarrow$ $\delta_2 \downarrow$	0.6	0.7	0.8	0.9
0.1	2387	2655	3286	4672
0.2	3424	4424	5335	6568
0.3	3885	4681	5712	6765
0.4	3979	4779	5777	6804
0.5	3981	4782	5781	6808

(d) Num. isolated nodes

Table 2: The effect of δ_1 and δ_2 on the relationship graph.

We note that a complete validation of all our malware families is not possible. Even the lesser task of quantifying the accuracy of the relationships we infer is not easy. Both of these limitations arise because we only have access to the malware meta-data. We focus on the latter issue here because it is more tractable.

One approach we considered was to check if malware that we identify as being related also share similar names. The experts who populate the malware meta-data tend to name each malware according to the "family" that they believe it belongs to. For example, most variants of W32/Gaobot are named W32/Gaobot.*.*. To check if the malware classification due to our algorithm aligns with the names provided by McAfee, we developed an "entropy metric", described next.

Assume our algorithm generates k family trees $T_1, .., T_k$. Let there be N families of malware according to names in the McAfee database. Say a McAfee family i has n_i members, and these are distributed across the k family trees output by our algorithm. If, for the McAfee family i, the fractions of its members across our k trees are f_1, .., f_k, then the entropy of that family in our classification is $e_i = \Sigma_{j=1}^{k}(-f_j log f_j)$. The mean entropy of all the McAfee families will be $\frac{\Sigma_{i=1}^{N} n_i * e_i}{\Sigma_{i=1}^{N} n_i}$.

The entropy value lies between 0 and $log_2 k$. The entropy value will be smaller if members of the same McAfee family go into a single family generated by our algorithm. Note that this metric is based only on the malware which have been classified into named families by McAfee. Note also that this metric does not quantify the evolutionary patterns. Nonetheless, it provides a good sanity check.

For $(\delta_1, \delta_2) = (0.7, 0.3)$ we obtain $k = 702$ trees. Thus, a random algorithm will have an entropy of $log_2 702 = 9.45$. In comparison, our algorithm results in an entropy of 1.04 (see Table 3). Drawing a simple analogy, the entropy metric can be interpreted as the mean number of binary questions to ask in order to decide which tree a malware goes into. Thus, for the members of a single McAfee-named family, we need to ask only ~ 1 question (equivalent to deciding between 2 trees) instead of the worst case ~ 9 questions.

We note that the entropy metric is less useful in actually choosing the parameters. This is mainly because different choices of the parameter space results in different number of trees. The smaller the number of trees, the smaller the upper bound on entropy, and the smaller the value of the entropy itself. For instance, the setting $(\delta_1, \delta_2) = (0.6, 0.1)$ has low entropy value partly because it has a smaller number of trees overall compared to, for example, $(\delta_1, \delta_2) = (0.7, 0.3)$ (see Table 3).

$\delta_1 \rightarrow$ $\delta_2 \downarrow$	0.6	0.7	0.8	0.9
0.1	0.82	1.36	1.7	1.9
0.2	1.07	1.15	1.27	1.47
0.3	1.02	1.04	1.24	1.41
0.4	1.03	0.95	1.14	1.43
0.5	1.05	0.97	1.17	1.43

Table 3: The entropy in the names assigned by McAfee.

To further verify if we are inferring relationships erroneously, we augmented the above checking of names with a manual check of the accuracy of some of the relationships. Specifically, for each family tree we obtained, we first checked if most of the members share a common prefix (e.g. W32/Gaobot). If this check is not satisfied, we manually check the meta-data to see if we erroneously inferred a relationship. In almost all the cases, we observed strong similarities between a "parent" and its "child". Besides, some well known malware evolutions were also observed in our family trees. For example, our trees expose the evolution of `Mytob` from `Mydoom` and `Sdbot`, which is well documented in the popular and technical press.

Later, in Section 5, we show the entire family trees for some of the largest families we uncovered. A more complete list of the families we inferred may be found at our Project Web site [7].

4. GROSS CHARACTERISTICS OF MALWARE EVOLUTION

As mentioned earlier, only 8182 malware from the entire malware database had a non-trivial amount of textual description associated with them. Our analysis focuses on these malware. After employing the parameters described in Section 3 and applying the tree decomposition process, we ended up with a collection of 702 malware family trees. Overall, 4681 malware were not classified into any family because of our choice of thresholds. In all, we have 2741 good nodes and just 58 fuzzy nodes.

Most of the analysis in this section provides a deeper look into the key properties of the 702 families, focusing mainly on the larger ones. We look at several features, such as the sizes of the families, how deep the families are, how many successors originate from a given malware in a family, how

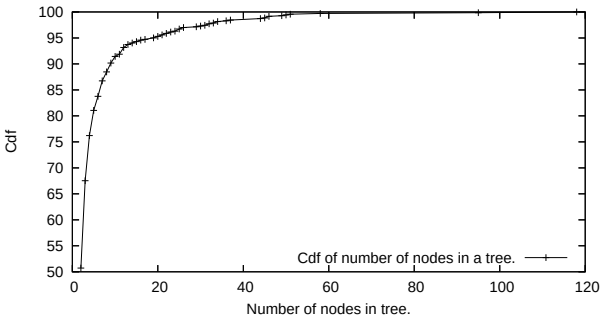

Figure 2: The cdf of the number of nodes in a tree.

long a time must elapse before a new generation of malware is spawned from a predecessor etc. We hope that this gross characterization of malware families will be useful in understanding the evolution and expansion of potent malware families.

We stress that our analysis is preliminary and is aimed at providing a first-cut insight into malware evolution. Our choice of parameters has been very conservative, and we believe that this causes us to infer fewer relationships than what actually exist. With better textual descriptions and more informed approaches for setting the thresholds, it may be possible to infer many more malware relationships with even higher confidence. We do believe, however, that the gross characteristics we derive, such as the relative sizes of malware families etc., will remain qualitatively similar (especially for the largest families we identify).

4.1 Analysis of size and fanout

Figure 2 shows the cumulative distribution of the number of malware in each family tree. We note that a large number of trees are very small: $> 90\%$ of the trees have less than 10 nodes and 50% of the trees have just two nodes (a single edge). We do find it very interesting that a handful of the families among the ones we identify are very large: 5 of the families have more then 50 malware each and the maximum number of malware in a family is 118! Later in this section, we delve deeper into some of the properties of the 5 large trees. In Section 5, we study the key features that are retained across generations of malware in some of these large families.

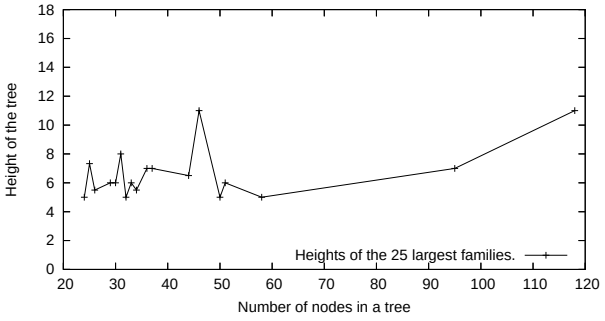

Figure 3: The height for the top 25 families.

We now briefly look at the number of "generations" in each malware family. It is likely that the new generation malware were released in response to specific counter-measures that were developed by AV companies to contain the previous generation. Thus, this analysis provides us a brief glimpse into the arms race between malware authors and AV experts. In Figure 3, we show the distribution of the depths of the 25 largest families. On average, these families span 7 generations! In Section 4.2, we study the time-span of these generations, and find that some of them span a few years.

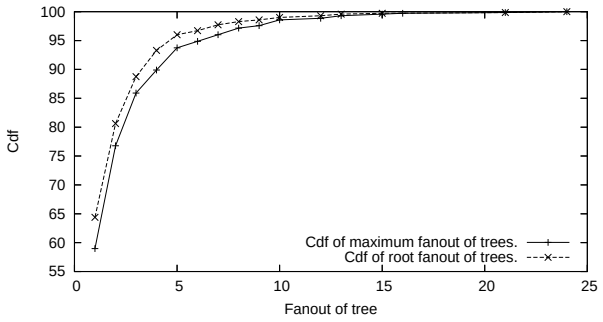

Figure 4: Cdf of maximum and root fan-outs in a tree.

Figure 4 shows the cdf of the maximum fanout in a tree, as well as the fan-out of the root of the tree. In several families, each malware spawns few other malware over time: 89% of the trees have a maximum fanout less than 5; and 93.7% of the roots have a fanout less than 5. However, we do observe two trees with a maximum fan-out > 20.

Since the root and maximum fan-out distributions are different, we immediately infer that the root may not necessarily have the largest fan-out. This happens whenever some intermediate malware bundles together a lot more exploits than any of its predecessors and in turn becomes the source for multiple future strains. In fact, we observed this phenomenon in our in-depth analysis of the `Mytob` family tree (more in Section 5).

Although not shown here, we found that the five largest families have maximum fan-outs of 16, 24, 12, 8 and 10, and root fan-outs of 1, 24, 12, 1 and 5, respectively. These families have 118, 95, 58, 51 and 50 malware respectively. We note that in two of the five cases, the root has a single descendant; and in two other cases, the root has the maximum number descendants among all nodes in a tree. In general, we note that the large families are characterized by malware (root or intermediate) that spawn numerous other strains.

4.2 Time-span of Evolution

Each edge in the graph we obtain has an associated *length*. We define the length of an edge as the difference in the "time of discovery" field in the two malware; we measure the length in days. Edge lengths help us understand the time duration over which the the successors of potent malware instances are developed and released. Since the "time of discovery" is entered when the malware was first analyzed by McAfee, our estimate of the length of an edge may be off from the true time lag between the first appearances of the

two malware. Nevertheless, it provides an interesting view into the evolutionary trends.

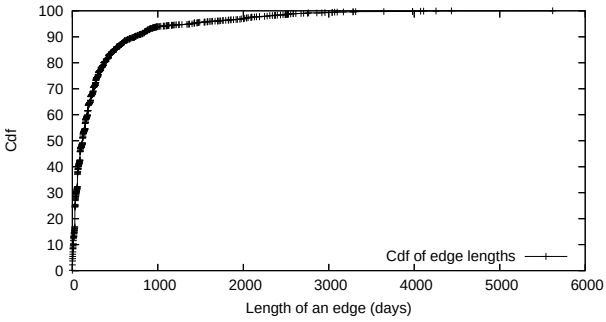

Figure 5: The cdf of the the lengths of all the edges.

Figure 5 shows a cdf of the lengths of all edges in our relationship graph. We see that 90% of the edge lengths are less than 730 days long. Thus, it appears that most malware are spawned from their predecessors in under two years.

We note that 0.5% of the edges – or 35 edges – are longer than 7yrs. These edges are spread across 23 families, of which 20 families have fewer than 10 malware. Thus, the long edges do not affect our observations regarding the largest families. Long edges usually appear when a malware instance that has an early timestamp (say in the early nineties) is also accompanied by a rich text description. Newer malware which have poor textual descriptions end up sharing quite a few features in common with this earlier instance. The newer malware, however, mostly appear as terminal nodes in the family tree.

Note that it is highly unlikely there is any evolution or code copying going on between the two malware connected by a long edge. Nevertheless we include the long edges in our analysis because they bring to focus important characteristics that even the malware which are separated by several years share with each other.[4]

Figures 6 and 7 show scatter-plots of the fanout of a malware versus the mean and the minimum length of its outgoing edges, respectively. These plots helps us understand the correlation between the popularity of a malware - defined in terms of how many immediate successors are spawned from it - and the time to the evolution of its successors. We note an interesting trend: malware with a high fanout do not have any long outgoing edges. In other words, it seems that malware which spawn a lot of children (perhaps because the malware's source code was reused very frequently), do so relatively quickly! Focusing on malware which spawn few successors, we note that a much longer time may elapse before they spawn their first successors. For example, in Fig 7, there are several cases where the minimum edge length is > 1000 days for malware with a fan-out < 5.

4.3 Evolution Dynamics of Malware Families

[4]We believe that it is possible to modify our algorithm to prune such edges (by imposing a threshold on the maximum allowed length of an edge). This is one direction we plan to explore in future work.

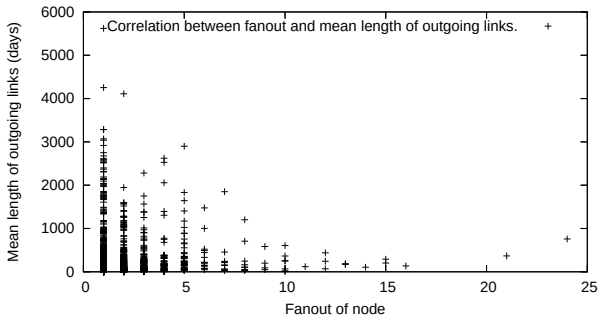

Figure 6: The correlation between the fanout of a node and the mean length of its outgoing edges.

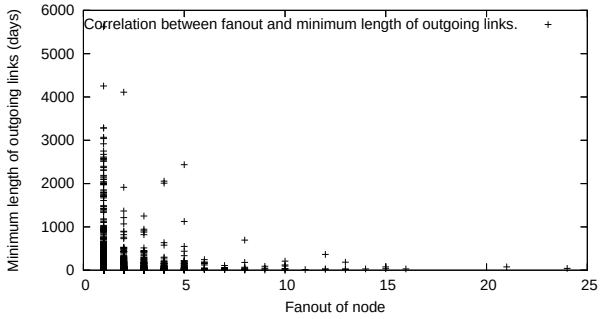

Figure 7: The correlation between the fanout of a node and the minimum length of its outgoing edges.

We now study the temporal patterns in the "birth", lifespan, and "death" of malware families. Our focus is on understanding how these evolutionary dynamics changed over the past decade or so. We note that this analysis sheds more light on the effect that two concurrent phenomena have on the overall prevalence of malware: (1) the ongoing race between malware code writers, and the anti-virus companies, which could cause some families to have a very long life time; and (2) the improvements in operating system and application software security, which could cause early death of some families.

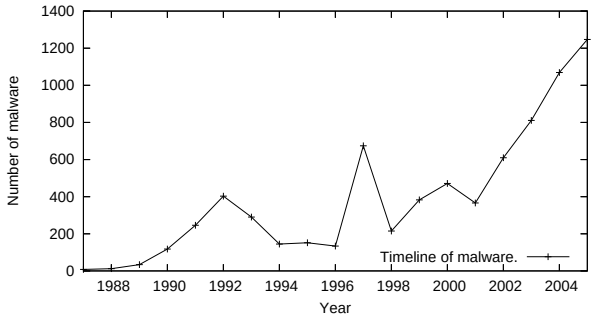

Figure 8: The number of malware spawned each year.

Figure 8 shows the time line of the distribution of "time of discovery" fields in the malware, binned by the year. This graph cover all the 8182 pieces of malware which have text descriptions. Thus, this plot includes even the isolated nodes in our relationship graph. We ignore all other malware in the database. The key point to note is that there is a definite

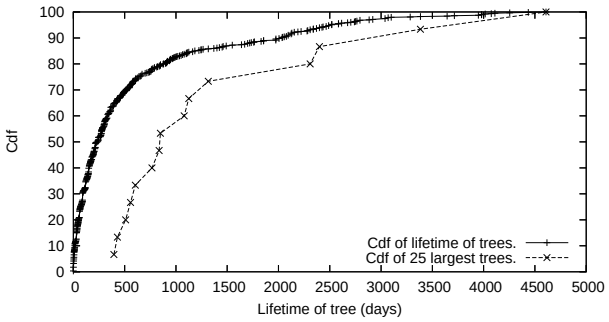

Figure 9: The cdf of the tree lifetimes for all families as well as the top 25 families.

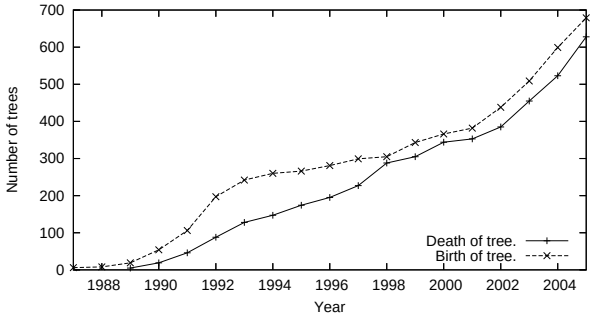

Figure 10: The cumulative count of the number of families born and the number of extinct trees.

slump in the number of new malware between 1994-1996 and also in 1998.

We now look at individual malware families and study their evolutionary trends. Figure 9 shows the cdf of the total lifetime of the 25 largest malware families as well as the cdf of the lifetime of all families. The lifetime of a family is defined as the difference in the timestamp of the root and the timestamp of the most recent member of the family. We note that 80% of all families have a lifetime less than 900 days (around 2.5yrs) years). However, a small fraction of the families, roughly 10%, last more than 2000 days (around 5 years!). Among the top 25 families, we see that $\sim 50\%$ of the families have a lifetime greater than 1000 days (around 2.5 years), and 20% of them have a lifetime more than 2300 days (around 6 years). Thus, it appears that the large families have very long lifetimes.

We consider families which were last seen in 2006 to be "active". These families have not been included in the above distributions. There are 74 "active" families, of which 45 have more than 2 nodes. Furthermore, we found that 10 of the 25 largest families are still active. The 45 active families with > 2 nodes have been around for a mean time of 527 days.

Figure 10 delves deeper into the evolution dynamics. Here, we compare the cumulative counts over time of the total number of trees born and the total number of trees that died since the first family originated in 1987. The gap between the two cumulative counts indicates the number of families alive in a given year. Note that there is a huge gap between the two plots in the early 1990s, indicating that a lot of trees

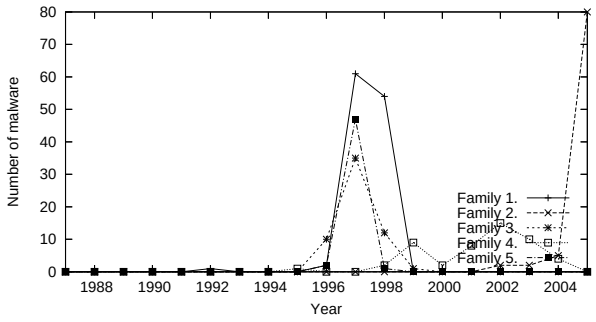

Figure 11: The timeline of the five largest families.

were active. By 1998, the two curves almost meet indicating (perhaps) that most of the vulnerabilities of the early nineties were patched. Along the same lines, we can infer that the early 2000's saw a revival of sorts in malware exploits (the two plots are almost parallel). These observations are supported by the anecdotal evidence on the prevalence of malware in recent years.

The right half of Figure 10 is even more interesting: the slope of the "birth of trees" curve becomes very steep, yet the gap between the two curves remains roughly fixed. This is representative of the ongoing tussle between malware authors and AV companies. For every malware family the AV companies eliminate, malware authors are able to come up with newer families which (possibly) exploit newer vulnerabilities. Thus, AV companies have their task cut out - they need to be extremely pro-active in identifying and eliminating new malware families in a timely fashion.

We now focus on the 5 largest families and dig a bit deeper into their life-spans (a more in-depth analysis is presented in the next section). Figure 11 shows the timelines of the five largest families. The largest family was chiefly active in 1997 and 1998. The members of this family do not have any specific McAfee family name. Most of the members of this family are viruses which infect files. These viruses mostly spread via floppy diskettes and online downloads. The second largest family is an Adware family and was widely active in 2005 (80 malware in 2005). It continues to be active and must be monitored closely. The third and fifth largest families are Word Macro families and were mainly active in 1997. The fourth largest family (a Joke family) has had a very long lifespan. It was first seen in 1995, and was last seen in 2004. It was mainly active in 2002 and 2003.

5. A DEEPER LOOK AT SOME MALWARE FAMILIES

In this section we drill down on the details of some of the largest families we identified in our analysis. We provide further details on these families and also highlight the unexpected characteristics exposed by our graph generation technique. We name each family according to the most commonly appearing McAfee-assigned name across all malware in the family. In some cases where we show graphical representations of the families, we also use the McAfee names

to identify individual malware in the family. As we will see, this allows us to better explain the results of our in-depth analysis.

In the interest of space, we analyze the key properties of four large family trees: the first contains "Mytob" malware instances predominantly, the second contains "Downloader" instances, the third contains instance of "Loveletter", and the fourth contains "Bagle" instances.

5.1 A `Mytob` Family

The "Mytob" malware instances are spread across several families in our classification. We show the largest Mytob family in Fig. 12. This family is quite interesting: it does not just contain the members of the Mytob family, but it also shows the "Sdbot" family evolving into the Mytob family.[5] The tree shown in Figure 12 has 46 nodes, a height of 16 (the maximum among all the Mytob families), and a maximum fanout of 10.

Most of the Mytobs in this tree spread via email. We examined some of the phrases which were common across the different generations of the malware in this tree. We noticed phrases such as "sender address", "mass mailing worm", "mail propagation", "arrives in an email message", "via SMTP", "via SMTP constructing messages using its own SMTP engine the worm guesses the recipient email server prepending the target domain", and "worm contains strings which it uses to randomly generate or guess email addresses these are prepended as user names".

Another interesting aspect of this tree is that it starts out with Sdbots but these eventually spawn Mytobs at a depth of 4. Actually, it is a well known fact that the Mytob family derives from the Sdbot and Mydoom families [5]. It is interesting to note that our classification algorithm is able to unearth such evolutionary trends without the direct aid of specific text describing the evolution.

Another interesting aspect of this tree is its structure. The tree is a linear chain in the initial part (i.e. all nodes have fanout 1), but it starts branching out once the Sdbots evolve into Mytobs. In particular, the `W32/Mytob.cv@MM` malware instance has a maximum fanout of 10. Three of its successors, `W32/Mytob.eu@MM`, `W32/Mytob.do@MM`, and `W32/Mytob.dl@MM` spawn further descendants. When we looked closely at the "time of discovery" of these malware instances, we realized that the family in Figure 12 seems to be evolving along the above three main sub-families. More interestingly, one of these sub-families (rooted at `W32/Mytob.eu@MM`) also has a couple of instances of `W32/Zotob` malware. This is a possible indication that Mytobs may be evolving into Zotobs.

5.2 A Downloader Family

Next, we consider a family tree consisting of "Downloader" instances. This tree is shown in Fig. 13 and has 44 nodes, a depth of 8, and a maximum fanout of 7. This tree started with 2 malware instances in 2004, had 27 instances appear in 2005, and 15 appeared in the first few months of 2006. Thus, it appears that this family is spreading fairly quickly.

The phrases that are most common among malware in this tree include "website hosting a scripted exploit which installs the downloader onto the user s system with no user interaction", "visiting a malicious web page either by clicking on a link or by the website hosting a scripted exploit", "downloaders are not viruses and as such do not themselves contain any method to replicate however they may themselves", "downloaders are designed to pull files from a remote website and execute the files that have been downloaded", " website being communicated is normally controlled by the malware author any files being downloaded can be remotely modified", and "adware is installed via a downloader it may install it cleanly with the relevant uninstaller included for the user". As expected, these phrases given a clear indication of the most common and potentially significant properties (from the point-of-view of developing counter-measures) of the Downloader malware.

This tree also illustrates the advantage of using conservative parameters in our algorithm: Note that this tree seems to have few "spurious" edges; In fact, *all* the 44 members of this particular family are classified as Downloaders by McAfee.

Among other interesting artifacts, the root node of this family tree, `Downloader-QO`, has the maximum fanout of 7. `Downloader-UT`, `Downloader-ABA`, `Downloaded-ABS`, and `Downloader-ASE` seem to be other important members of the tree spawning numerous other descendants. This tree seems to be evolving in several directions. It is also evolving fairly quickly: a major fraction of the members of this family appeared very recently (in 2005 and 2006).

5.3 A `Loveletter` Family

Next, we consider a family tree that captures instances of "VBS/Loveletter" malware. These malware are considered extremely potent as they are known to cause serious damage to infected hosts. We focus on one Loveletter family that our algorithm derived. This family contains 23 malware instances (see Figure 14). All 23 are named Loveletter/* by McAfee. The tree has a depth of 5, and a maximum fanout of 10.

A key aspect of this family is that it is very short-lived: it had a total life-span of 131 days! When we perused the online technical press to learn more about these malware instances, we found that the malware in this family got a lot of media attention at the time. This led to an aggressive response from the AV companies and OS vendors, leading to quick patches. This may explain the short lifespan that we observed for the family.

The phrases which are most common to malware in this tree include "arrive in an email message", "MSKERNEL32.VBS", "VBS Loveletter", "Win32DLL.vbs in order to run the worm

[5]There is also a second small tree which shows "Mydoom" evolving into Mytob. This is not shown here

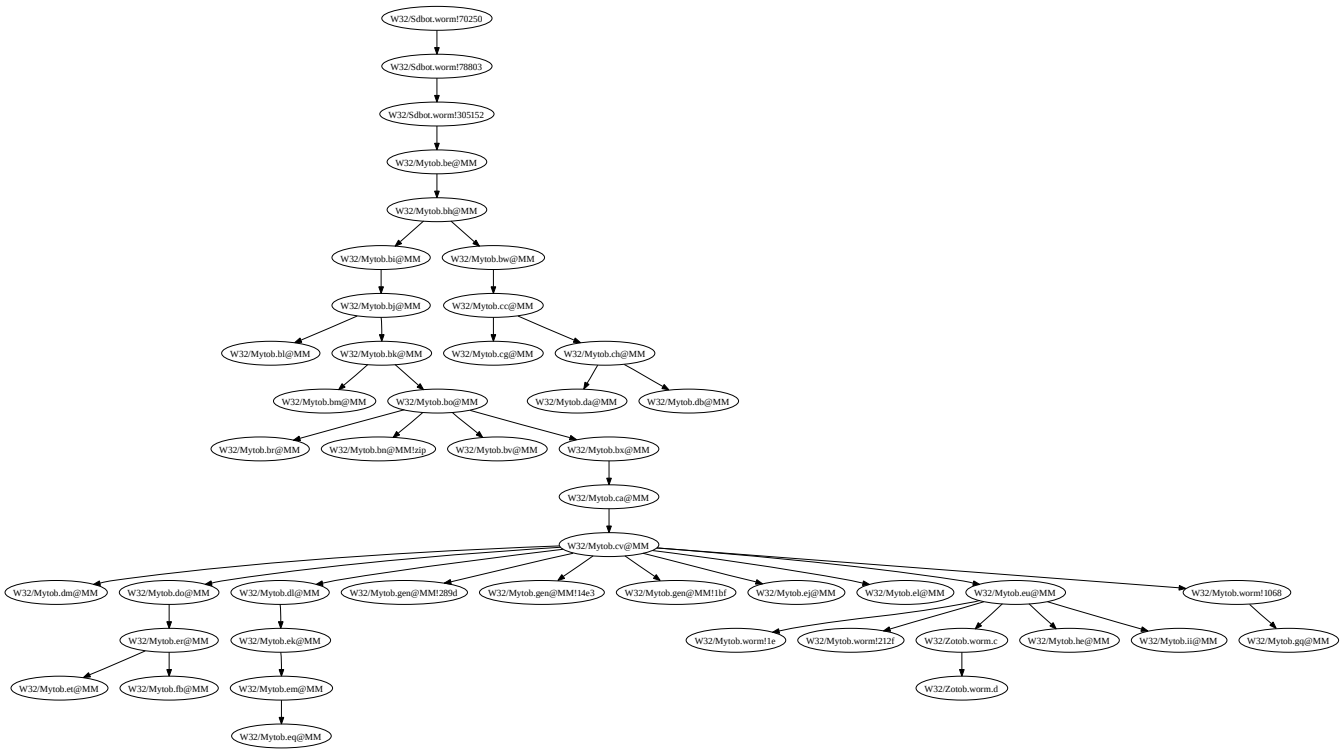

Figure 12: A Mytob family tree.

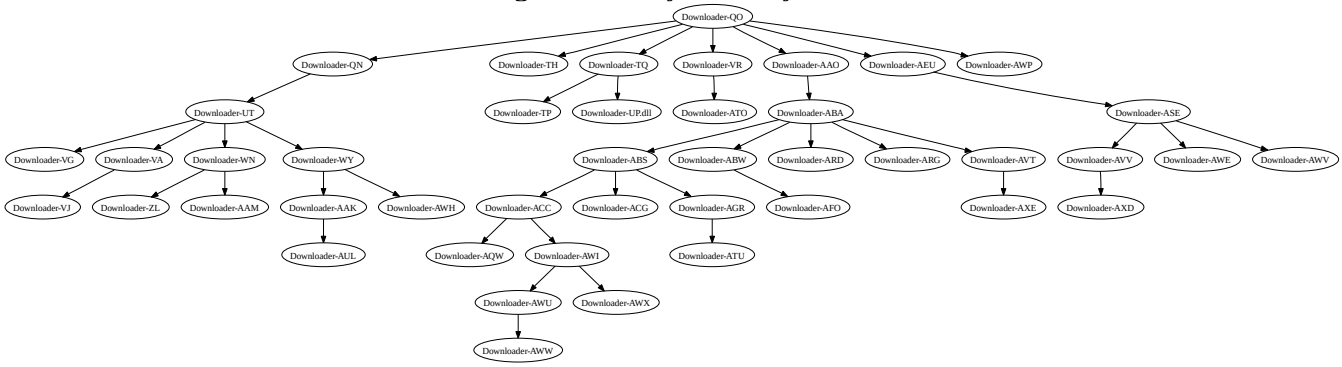

Figure 13: A Downloader malware family tree.

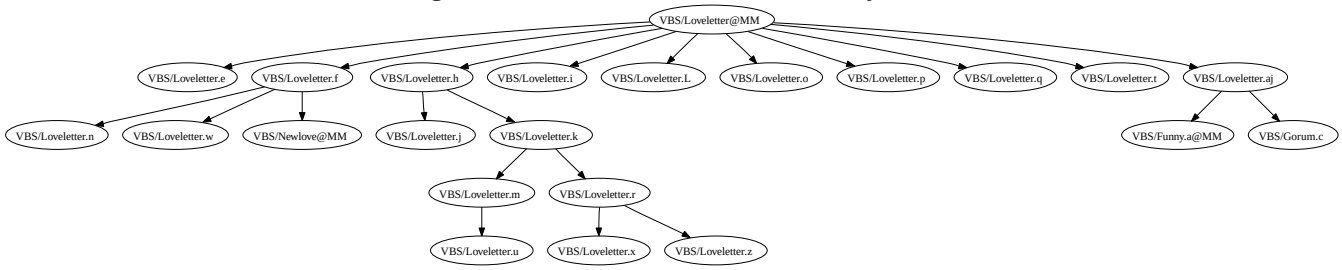

Figure 14: A Loveletter malware family tree.

at system startup". These phrases tell us crucial details about the family: The malware instances arrive in email messages; MSKERNEL32.VBS is one of the files the malware instances all attempt to edit; and Win32DLL.vbs is part of the key added to the Windows registry in order to make the worm run at system startup. Different generations in this family differ mainly in the contents of the particular email that ends up becoming the vector for the malware.

5.4 A `Bagle` Family

The W32/Bagle.* collection of malware is one of the most prolific set of malware in our entire dataset. In our classification, we found that the Bagle malware instances were spread across four large family trees and seven smaller trees. The biggest among the large families is shown in Figure 15. This family has 34 nodes, a depth of 7 and a fanout of 5. Some

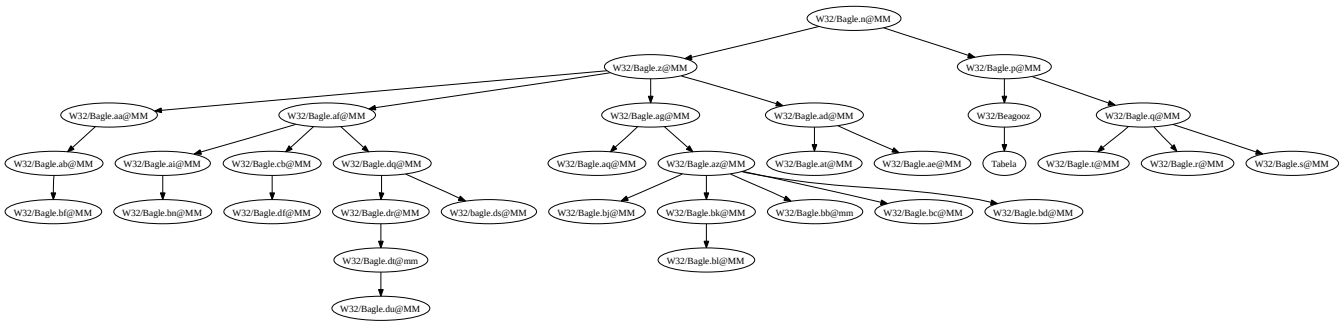

Figure 15: A Bagle family tree.

of the phrases which were common to the malware in this tree include "spoofed", "worm opens", "mail propagation", "copies itself to folders", "contains its own smtp engine", "contains a remote access component", "peer to peer applications", "kazaa bearshare limewire", "email addresses are harvested", "spam", and "mass mailing worm". As is well known (and as the phrases seem to indicate), the Bagle variants copy themselves to the shared folders of popular peer-to-peer applications.

The malware instances belonging to this tree were most active in the year 2004, with 22 variants being discovered in that year. Only six new variants were discovered in 2005 and 2006. This may suggest that the malware may have lost some of its prevalence, but there are a few unpatched vulnerabilities that continue to be exploited by the new variants of Bagle. We speculate that the high number of variants in 2004 may be due to the rise and popularity of peer to peer applications around that time, and that the drop in 2005 and 2006 may have come about due to the community's growing awareness of the security problems associated with popular peer-to-peer applications.[6]

6. CONCLUSIONS

Users throughout the Internet are plagued by malicious attacks on an on-going basis. The task of defending against these attacks is complicated by many factors, including complexity, scale, and the increasing sophistication of malware authors. The premise of our work is that an expanded perspective on malware behavior and in particular the relationships between malware variants will eventually lead to the development of more effective countermeasures.

In this paper we present an analysis of malcode meta-data compiled by McAfee, one of the largest AV companies in the world. The meta-data describes malware that was collected by McAfee and other AV companies over a period of 19 years. The objective of our work is to identify and evaluate relationships between malware instances based on the details of their descriptions. We do this through a process that begins by decomposing the descriptions into frequent phrases, and then pruning the resulting set to eliminate the superfluous phrases. Next, we establish relationships between instances of malware using a tunable graph pruning algorithm that is based on the similarity of frequent phrases between all instances of malware in our data set. In our analysis, we show the trade offs in graph structure using different parameter settings and select a configuration that results in a graph that we validate as being "most likely" using the malware names applied by McAfee.

The resulting family trees have rich structure. We identify 702 distinct malware families. Some of the families are very large, containing in excess of 50 members. We found that some of the families were active for a few years at a stretch, while others last no more than a few days. Detailed examination of the trees reveals many instances where specific traits (as identified by a specific phrase) are inherited after many months and that one instance of malware may spawn many others. Several of the trees are available at our project web-site ([7] - under construction) for general perusal.

We plan to pursue several extensions to this work. First, we hope to expand the corpus of malware meta-data in order to flesh out the evolutionary characteristics of malware in greater detail. Second, we believe that adding the behavioral characteristics such as those identified in [16] and others will further enrich our analysis. Finally, we will work more closely with AV companies and others concerned with malware analysis, to develop methods for anticipating future trends in malware development. We hope that this will enable AV companies to generate malware counter-measures in a more proactive fashion.

[6]However, we have no data to back this up!

7. ACKNOWLEDGEMENTS

The authors wish to sincerely thank Joe Telafici and his colleagues at McAfee's AVERT Labs for granting us access to the VIL database. The authors also thank Jeff Naughton for his input and guidance on this work.

8. REFERENCES

[1] The Honeynet Project. http://project.honeynet.org, 2003.
[2] ExtPhr32 - Frequent phrase extraction tool. http://instruct.uwo.ca/gplis/677/extphr32/extphr32.htm, 2005.
[3] Regmon. http://www.sysinternals.com, 2005.
[4] The SoftICE Driver Suite. http://www.compuware.com, 2005.
[5] Malware dangers grow as e-criminals pool resources. http://www.itweek.co.uk/itweek/news/2160344/malware-dangers-grow-criminals, 2006.
[6] The IDA Pro Disassembler and Debugger. http://www.datarescue.com, 2007.
[7] The Malware Geneology Project, University of Wisconsin-Madison - Malware Families. http://www.cs.wisc.edu/ archit/Malware, 2007.

[8] H. Ahonen-Myka. Mining All Maximal Frequent Word Sequences in a Set of Sentences. In *Proceedings of ACM International conference on Information and Knowledge Management*, New York, NY, October 2005.

[9] P. Barford and V. Yegneswaran. *An Inside Look at Botnets*, volume 27 of *Advances in Information Security, Malware Detection*. Springer, 2007.

[10] M. Eichin and J. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *Proceedings of IEEE Security and Privacy*, Oakland, CA, June 1989.

[11] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A Sense of Self for Unix Processes. In *Proceedings of IEEE Security and Privacy*, Oakland, CA, June 1996.

[12] F. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *Proceedings of The 10th European Symposium on Research in Computer Security (ESORICS '05)*, September 2005.

[13] J. Kephart and S. White. Directed-Graph Empdemiological Models of Computer Viruses. In *Proceedings of IEEE Security and Privacy*, Oakland, CA, June 1991.

[14] A. Kumar, V. Paxson, and N. Weaver. Exploiting Underlying Structure for Detailed Reconstruction of an Internet Scale Event. In *Proceedings of ACM Internet Measurement Conference*, Berkeley, CA, November 2005.

[15] Z. Li, s. Lu, s. Myagmar, and Y. Shou. CP:Miner: Finding Copy-Paste and Related Bugs in Large Scale Software Code. *IEEE Transactions on Software Engineering*, 32(3), March 2006.

[16] J. Ma, J. Dunagan, H. Wang, S. Savage, and G. Voelker. Finding Diversity in Remote Code Injection Exploits. In *Proceedings of ACM Internet Measurement Conference*, Rio de Janeiro, Brazil, October 2006.

[17] McAfee. Avert labs threat library. http://vil.nai.com, 2007.

[18] Q. Mei and C. Zhai. Discovering Evolutionary Theme Patterns from Text - An Exploration of Temporal Text Mining. In *Proceedings of ACM SIGKDD*, Chicago, IL, August 2005.

[19] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Stanifor, and N. Weaver. The Spread of the Sapphire/Slammer Worm. http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html.

[20] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Stanifor, and N. Weaver. Inside the Slammer Worm. In *Proceedings of IEEE Security and Privacy*, Oakland, CA, June 2003.

[21] D. Moore, C. Shannon, and J. Brown. Code Red: A Case Study on the Spread and Victims of an Internet Worm. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[22] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Proceedings of the 2001 USENIX Security Symposium*, Washington D.C., August 2001.

[23] R. Pang, V. Yegneswaran, P. Barford, v. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of ACM Internet Measurement Conference*, Taormina, Italy, October 2004.

[24] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of ACM Internet Measurement Conference*, November 2006.

[25] Sophos. Threat analysis. http://www.sophos.com/security/analysis, 2007.

[26] Symantec. Enterprise support knowledgebase. http://www.symantec.com, 2007.

[27] P. Szor. *The Art of Computer Virus Research and Defense*. Addison Wesley, 2005.

[28] J. Ullrich. Dshield. http://www.dshield.org, 2005.

[29] H. Wang, C. Guo, D. Simon, and A. Zungenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *Proceedings of ACM SIGCOMM*, Portland, OR, August 2004.

[30] V. Yegneswaran, P. Barford, and J. Ullrich. Internet Intrusions: Global Characteristics and Prevalence. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, June 2003.

[31] V. Yegneswaran, J. Giffin, P. Barford, and S. Jha. An Architecture for Generating Semantic-Aware Signatures. In *Proceedings of the 2001 USENIX Security Symposium*, Baltimore, MD, August 2005.

[32] M. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, 42(1), 2001.