Learning and Classification of Malware Behavior

Konrad Rieck[1], Thorsten Holz[2], Carsten Willems[2],
Patrick Düssel[1], and Pavel Laskov[1,3]

[1] Fraunhofer Institute FIRST
Intelligent Data Analysis Department, Berlin, Germany
[2] University of Mannheim
Laboratory for Dependable Distributed Systems, Mannheim, Germany
[3] University of Tübingen
Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany

Abstract. Malicious software in form of Internet worms, computer viruses, and Trojan horses poses a major threat to the security of networked systems. The diversity and amount of its variants severely undermine the effectiveness of classical signature-based detection. Yet variants of malware families share typical *behavioral patterns* reflecting its origin and purpose. We aim to exploit these shared patterns for classification of malware and propose a method for learning and discrimination of malware behavior. Our method proceeds in three stages: (a) behavior of collected malware is monitored in a sandbox environment, (b) based on a corpus of malware labeled by an anti-virus scanner a *malware behavior classifier* is trained using learning techniques and (c) discriminative features of the behavior models are ranked for explanation of classification decisions. Experiments with different heterogeneous test data collected over several months using honeypots demonstrate the effectiveness of our method, especially in detecting *novel* instances of malware families previously not recognized by commercial anti-virus software.

1 Introduction

Proliferation of malware poses a major threat to modern information technology. According to a recent report by Microsoft [1], every third scan for malware results in a positive detection. Security of modern computer systems thus critically depends on the ability to keep anti-malware products up-to-date and abreast of current malware developments. This has proved to be a daunting task. Malware has evolved into a powerful instrument for illegal commercial activity, and a significant effort is made by its authors to thwart detection by anti-malware products. As a result, new malware variants are discovered at an alarmingly high rate, some malware families featuring tens of thousands of currently known variants.

In order to stay alive in the arms race against malware writers, developers of anti-malware software heavily rely on automatic malware analysis tools. Unfortunately, malware analysis is obstructed by hiding techniques such as polymorphism and obfuscation. These techniques are especially effective against byte-level content analysis [17, 19] and static malware analysis methods [8, 10, 11]. In contrast to static techniques, dynamic analysis of binaries during run-time enables monitoring of malware

behavior, which is more difficult to conceal. Hence, a substantial amount of recent work has focused on development of tools for collecting, monitoring and run-time analysis of malware [3, 5, 6, 14, 22, 23, 25, 27, 36, 38].

Yet the means for collection and run-time analysis of malware by itself is not sufficient to alleviate a threat posed by novel malware. What is needed is the ability to *automatically* infer characteristics from observed malware behavior that are essential for detection and categorization of malware. Such characteristics can be used for signature updates or as an input for adjustment of heuristic rules deployed in malware detection tools. The method for automatic classification of malware behavior proposed in this contribution develops such a characterization of previously unknown malware instances by providing answers to the following questions:

1. *Does an unknown malware instance belong to a known malware family or does it constitute a novel malware strain?*
2. *What behavioral features are discriminative for distinguishing instances of one malware family from those of other families?*

We address these questions by proposing a methodology for *learning* the behavior of malware from labeled samples and constructing models capable of classifying unknown variants of known malware families while rejecting behavior of benign binaries and malware families not considered during learning. The key elements of this approach are the following:

(a) Malware binaries are collected via honeypots and spam-traps, and malware family labels are generated by running an anti-virus tool on each binary. To assess *behavioral patterns* shared by instances of the same malware family, the behavior of each binary is monitored in a sandbox environment and behavior-based analysis reports summarizing operations, such as opening an outgoing IRC connection or stopping a network service, are generated. Technical details on the collection of our malware corpus and the monitoring of malware behavior are provided in Sections 3.1–3.2.

(b) The learning algorithm in our methodology embeds the generated analysis reports in a high-dimensional vector space and learns a *discriminative model* for each malware family, i.e., a function that, being applied to behavioral patterns of an unknown malware instance, predicts whether this instance belongs to a known family or not. Combining decisions of individual discriminative models provides an answer to the first question stated above. The embedding and learning procedures are presented in Sections 3.3– 3.4.

(c) To understand the importance of specific features for classification of malware behavior, we exploit the fact that our learning model is defined by weights of behavioral patterns encountered during the learning phase. By sorting these weights and considering the most prominent patterns, we obtain characteristic features for each malware family. Details of this feature ranking are provided in Section 3.5.

We have evaluated our method on a large corpus of recent malware obtained from honeypots and spam-traps. Our results show that 70% of malware instances not identified by an anti-virus software can be correctly classified by our approach. Although such

accuracy may not seem impressive, in practice it means that the proposed method would provide correct detections in two thirds of hard cases *when anti-malware products fail*. We have also performed, as a sanity check, classification of benign executables against known malware families, and observed 100% detection accuracy. This confirms that the features learned from the training corpus are indeed characteristic for malware and not obtained by chance. The manual analysis of most prominent features produced by our discriminative models has produced insights into the relationships between known malware families. Details of experimental evaluation of our method are provided in Section 4.

2 Related work

Extensive literature exists on static analysis of malicious binaries, e.g. [8, 10, 18, 20]. While static analysis offers a significant improvement in malware detection accuracy compared to traditional pattern matching, its main weakness lies in the difficulty to handle obfuscated and self-modifying code [33]. Moreover, recent work of Moser et al. presents obfuscation techniques that are provably NP-hard for static analysis [24].

Dynamic malware analysis techniques have previously focused on obtaining reliable and accurate information on execution of malicious programs [5, 6, 23, 38]. As it was mentioned in the introduction, the main focus of our work lies in *automatic processing* of information collected from dynamic malware analysis. Two techniques for behavior-based malware analysis using clustering of behavior reports have been recently proposed [4, 21]. Both methods transform reports of observed behavior into sequences and use sequential distances (the normalized compression distance and the edit distance, respectively) to group them into clusters which are believed to correspond to malware families. The main difficulty of clustering methods stems from their unsupervised nature, i.e., the lack of any external information provided to guide analysis of data. Let us illustrate some practical problems of clustering-based approaches.

A major issue for any clustering method is to decide how many clusters are present in the data. As it is pointed out by Bailey et al. [4], there is a trade-off between cluster size and the number of clusters controlled by a parameter called *consistency* which measures a ratio between intra-cluster and inter-cluster variation. A good clustering should exhibit high consistency, i.e., uniform behavior should be observed within clusters and heterogeneous behavior between different clusters. Yet in the case of malware behavior – which is heterogeneous by its nature – this seemingly trivial observation implies that a *large* number of *small* classes is observed if consistency is to be kept high. The results in [4] yield a compelling evidence to this phenomenon: given 100% consistency, a clustering algorithm generated from a total of 3,698 malware samples 403 clusters, of which 206 (51%) contain just one single executable. What a practitioner is looking for, however, is exactly the opposite: a *small* number of *large* clusters in which variants belong to the same family. The only way to attain this effect using consistency is to play with different consistency levels, which (a) defeats the purpose of automatic classification and (b) may still be difficult to attain at a single consistency level.

Another recent approach to dynamic malware analysis is based on mining of malicious behavior reports [9]. Its main idea is to identify differences between malware

samples and benign executables, which can be used as specification of malicious behavior (malspecs). In contrast to this work, the aim of our approach is discrimination between families of malware instead of discrimination between specific malware instances and benign executables.

3 Methodology

Current malware is characterized by rich and versatile behavior, although large families of malware, such as all variants of the Allaple worm, share common behavioral patterns, e.g., acquiring and locking of particular mutexes on infected systems. We aim to exploit these shared patterns using *machine learning techniques* and propose a method capable of automatically classifying malware families based on their behavior. An outline of our learning approach is given by the following basic steps:

1. *Data acquisition.* A corpus of malware binaries currently spreading in the wild is collected using a variety of techniques, such as honeypots and spam-traps. An antivirus engine is applied to identify known malware instances and to enable learning and subsequent classification of family-specific behavior.
2. *Behavior Monitoring.* Malware binaries are executed and monitored in a sandbox environment. Based on state changes in the environment – in terms of API function calls – a behavior-based analysis report is generated.
3. *Feature Extraction.* Features reflecting behavioral patterns, such as opening a file, locking a mutex, or setting a registry key, are extracted from the analysis reports and used to embed the malware behavior into a high-dimensional vector space.
4. *Learning and Classification.* Machine learning techniques are applied for identifying the shared behavior of each malware family. Finally, a combined classifier for all families is constructed and applied to different testing data.
5. *Explanation.* The discriminative model for each malware family is analyzed using the weight vector expressing the contribution of behavioral patterns. The most prominent patterns yield insights into the classification model and reveal relations between malware families.

In the following sections we discuss these individual steps and corresponding technical background in more detail – providing examples of analysis reports, describing the vectorial representation, and explaining the applied learning algorithms.

3.1 Malware Corpus for Learning

Our malware collection used for learning and subsequent classification of malware behavior comprises more than 10,000 unique samples obtained using different collection techniques. The majority of these samples was gathered via *nepenthes*, a honeypot solution optimized for malware collection [3]. The basic principle of nepenthes is to emulate only the *vulnerable* parts of an exploitable network service: a piece of self-replicating malware spreading in the wild will be tricked into exploiting the emulated vulnerability. By automatically analyzing the received payload, we can then obtain a binary copy

of the malware itself. This leads to an effective solution for collecting self-propagating malware such as a wide variety of worms and bots. Additionally, our data corpus contains malware samples collected via *spam-traps*. We closely monitor several mailboxes and catch malware propagating via malicious e-mails, e.g., via links embedded in message bodies or attachments of e-mails. With the help of spam-traps, we are able to obtain malware such as Trojan horses and network backdoors.

The capturing procedure based on honeypots and spam-traps ensures that all samples in the corpus are *malicious*, as they were either collected while exploiting a vulnerability in a network service or contained in malicious e-mail content. Moreover, the resulting learning corpus is *current*, as all malware binaries were collected within 5 months (starting from May 2007) and reflect malware families actively spreading in the wild. In the current prototype, we focus on samples collected via honeypots and spam-traps. However, our general methodology on malware classification can be easily extended to include further malware classes, such as rootkits and other forms of non-self-propagating malware, by supplying the corpus with additional collection sources.

After collecting malware samples, we applied the anti-virus (AV) engine *Avira AntiVir* [2] to partition the corpus into common families of malware, such as variants of RBot, SDBot and Gobot. We chose Avira AntiVir as it had one of the best detection rates of 29 products in a recent AV-Test and detected 99.29% of 874,822 unique malware samples [35]. We selected the 14 malware families obtained from the most common labels assigned by Avira AntiVir on our malware corpus. These families listed in Table 1 represent a broad range of malware classes such as Trojan horses, Internet worms and bots. Note that binaries not identified by Avira AntiVir are excluded from the malware corpus. Furthermore, the contribution of each family is restricted to a maximum of 1,500 samples resulting in 10,072 unique binaries of 14 families.

Table 1. Malware families assigned by Avira AntiVir in malware corpus of 10,072 samples. The numbers in brackets indicate occurrences of each malware family in the corpus.

1: Backdoor.VanBot	(91)	8: Worm.Korgo	(244)
2: Trojan.Bancos	(279)	9: Worm.Parite	(1215)
3: Trojan.Banker	(834)	10: Worm.PoeBot	(140)
4: Worm.Allaple	(1500)	11: Worm.RBot	(1399)
5: Worm.Doomber	(426)	12: Worm.Sality	(661)
6: Worm.Gobot	(777)	13: Worm.SdBot	(777)
7: Worm.IRCBot	(229)	14: Worm.Virut	(1500)

Using an AV engine for labeling malware families introduces a problem: AV labels are generated by human analysts and are prone to errors. However, the learning method employed in our approach (Section 3.4) is well-known for its generalization ability in presence of classifcation noise [34]. Moreover, our methodology is not bound to a particular AV engine and our setup can easily be adapted to other AV engines and labels or a combination thereof.

3.2 Monitoring Malware Behavior

The behavior of malware samples in our corpus is monitored using *CWSandbox* – an analysis software generating reports of observed program operations [38]. The samples are executed for a limited time in a native Windows environment and their behavior is logged during run-time. CWSandbox implements this monitoring by using a technique called *API hooking* [13]. Based on the run-time observations, a detailed report is generated comprising, among others, the following information for each analyzed binary:

- Changes to the file system, e.g., creation, modification or deletion of files.
- Changes to the Windows registry, e.g., creation or modification of registry keys.
- Infection of running processes, e.g., to insert malicious code into other processes.
- Creation and acquiring of mutexes, e.g. for exclusive access to system resources.
- Network activity and transfer, e.g., outbound IRC connections or ping scans.
- Starting and stopping of Windows services, e.g., to stop common AV software.

Figure 1 provides examples of observed operations contained in analysis reports, e.g., copying of a file to another location or setting a registry key to a particular value. Note, that the tool provides a high-level summary of the observed events and often more than one related API call is aggregated into a single operation.

```
copy_file (filetype="File" srcfile="c:\1ae8b19ecea1b65705595b245f2971ee.exe",
  dstfile="C:\WINDOWS\system32\urdvxc.exe", flags="SECURITY_ANONYMOUS")

set_value (key="HKEY_CLASSES_ROOT\CLSID\{3534943...2312F5C0&}",
  data="lsslwhxtettntbkr")

create_process (commandline="C:\WINDOWS\system32\urdvxc.exe /start",
  targetpid="1396", showwindow="SW_HIDE", apifunction="CreateProcessA")

create_mutex (name="GhostBOT0.58b", owned="1")

connection (transportprotocol="TCP", remoteaddr="XXX.XXX.XXX.XXX",
  remoteport="27555", protocol="IRC", connectionestablished="1", socket="1780")

irc_data (username="XP-2398", hostname="XP-2398", servername="0",
  realname="ADMINISTRATOR", password="r0flc0mz", nick="[P33-DEU-51371]")
```

Fig. 1. Examples of operations as reported by CWSandbox during run-time analysis of different malware binaries. The IP address in the fifth example is sanitized.

3.3 Feature Extraction and Embedding

The analysis reports provide detailed information about malware behavior, yet raw reports are not suitable for application of learning techniques as these usually operate on vectorial data. To address this issue we derive a generic technique for mapping analysis reports to a high-dimensional feature space.

Our approach builds on the *vector space model* and *bag-of-words model*; two similar techniques previously used in the domains of information retrieval [29] and text processing [15, 16]. A document – in our case an analysis report – is characterized by frequencies of contained strings. We refer to the set of considered strings as feature set $\mathcal{F}$ and denote the set of all possible reports by $\mathcal{X}$. Given a string $s \in \mathcal{F}$ and a report $x \in \mathcal{X}$, we determine the number of occurrences of s in x and obtain the frequency $f(x, s)$. The frequency of a string s acts as a measure of its importance in x, e.g., $f(x, s) = 0$ corresponds to no importance of s, while $f(x, s) > 0.5$ indicates dominance of s in x. We derive an embedding function ϕ which maps analysis reports to an $|\mathcal{F}|$-dimensional vector space by considering the frequencies of all strings in $\mathcal{F}$:

$$\phi : \mathcal{X} \to \mathbb{R}^{|\mathcal{F}|}, \quad \phi(x) \mapsto (f(x, s))_{s \in \mathcal{F}}$$

For example, if $\mathcal{F}$ contains the strings `copy_file` and `create_mutex`, two dimensions in the resulting vector space correspond to the frequencies of these strings in analysis reports. Computation of these high-dimensional vectors seems infeasible at a first glance, as $\mathcal{F}$ may contain arbitrary many strings, yet there exist efficient algorithms that exploit the sparsity of this vector representation to achieve linear run-time complexity in the number of input bytes [28, 31].

In contrast to textual documents we can not define a feature set $\mathcal{F}$ a priori, simply because not all important strings present in reports are known in advance. Instead, we define $\mathcal{F}$ *implicitly* by deriving string features from the observed malware operations. Each monitored operation can be represented by a string containing its name and a list of key-value pairs, e.g., a simplified string s for copying a file is given by

```
"copy_file (srcfile=A, dstfile=B)"
```

Such representation yields a very specific feature set $\mathcal{F}$, so that slightly deviating behavior is reflected in different strings and vector space dimensions. Behavioral patterns of malware, however, often express variability induced by obfuscation techniques, e.g., the destination for copying a file might be a random file name. To address this problem, we represent each operation by *multiple strings* of different specificity. For each operation we obtain these strings by defining subsets of key-value pairs ranging from the full to a coarse representation. E.g. the previous example for copying a file is associated with three strings in the feature set $\mathcal{F}$

$$\texttt{"copy\_file ..."} \quad \longrightarrow \quad \begin{cases} \texttt{"copy\_file\_1 (srcfile=A, dstfile=B)"} \\ \texttt{"copy\_file\_2 (srcfile=A)"} \\ \texttt{"copy\_file\_3 ()"} \end{cases}$$

The resulting implicit feature set $\mathcal{F}$ and the vector space induced by ϕ correspond to various strings of possible operations, values and attributes, thus covering a wide range of potential malware behavior. Note, that the embedding of analysis reports using a feature set $\mathcal{F}$ and function ϕ is generic, so that it can be easily adapted to different report formats of malware analysis software.

3.4 Learning and Classification

The embedding function ϕ introduced in the previous section maps analysis reports into a vector space in which various learning algorithms can be applied. We use the well-established method of *Support Vector Machines* (SVM), which provides strong generalization even in presence of noise in features and labels. Given data of two classes an SVM determines an *optimal hyperplane* that separates points from both classes with maximal margin [e.g. 7, 30, 34].

The optimal hyperplane is represented by a vector w and a scalar b such that the inner product of w with vectors $\phi(x_i)$ of the two classes are separated by an interval between -1 and $+1$ subject to b:

$$\langle w, \phi(x_i)\rangle + b \geq +1, \text{ for } x_i \text{ in class 1,}$$
$$\langle w, \phi(x_i)\rangle + b \leq -1, \text{ for } x_i \text{ in class 2.}$$

The optimization problem to be solved for finding w and b can be solely formulated in terms of inner products $\langle \phi(x_i), \phi(x_j)\rangle$ between data points. In practice these inner products are computed by so called *kernel functions*, which lead to non-linear classification surfaces. For example, the kernel function k for polynomials of degree d used in our experiments is given by

$$k(x_i, x_j) = (\langle \phi(x_i), \phi(x_j)\rangle + 1)^d.$$

Once trained, an SVM classifies a new report x by computing its distance $h(x)$ from the separating hyperplane as

$$h(x) = \langle w, \phi(x)\rangle + b = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b,$$

where α_i are parameters obtained during training and y_i labels ($+1$ or -1) of training data points. The distance $h(x)$ can then be used for multi-class classification among malware families in one of the following ways:

1. *Maximum distance.* A label is assigned to a new behavior report by choosing the classifier with the highest positive score, reflecting the distance to the most discriminative hyperplane.
2. *Maximum probability estimate.* Additional calibration of the outputs of SVM classifiers allows to interpret them as probability estimates. Under some mild probabilistic assumptions, the conditional posterior probability of the class $+1$ can be expressed as:
$$P(y = +1 \,|\, h(x)) = \frac{1}{1 + \exp(Ah(x) + B)},$$
where the parameters A and B are estimated by a logistic regression fit on an independent training data set [26]. Using these probability estimates, we choose the malware family with the highest estimate as our classification result.

In the following experiments we will use the maximum distance approach for combining the output of individual SVM classifiers. The probabilistic approach is applicable to prediction as well as detection of novel malware behavior and will be considered in Section 4.3.

3.5 Explanation of Classification

A security practitioner is not only interested in how accurate a learning system performs, but also needs to understand how such performance is achieved – a requirement not satisfied by many "black-box" applications of machine learning. In this section we supplement our proposed methodology and provide a procedure for explaining classification results obtained using our method.

The discriminative model for classification of a malware family is the hyperplane w in the vector space $\mathbb{R}^{|\mathcal{F}|}$ learned by an SVM. As the underlying feature set $\mathcal{F}$ corresponds to strings $s_i \in \mathcal{F}$ reflecting observed malware operations, each dimension w_i of w expresses the contribution of an operation to the decision function h. Dimensions w_i with high values indicate strong discriminative influence, while dimensions with low values express few impact on the decision function. By sorting the components w_i of w one obtains a *feature ranking*, such that $w_i > w_j$ implies higher relevance of s_i over s_j. The most prominent strings associated with the highest components of w can be used to gain insights into the trained decision function and represent typical behavioral patterns of the corresponding malware family.

Please note that an explicit representation of w is required for computing a feature ranking, so that in the following we provide explanations of learned models only for polynomial kernel functions of degree 1.

4 Experiments

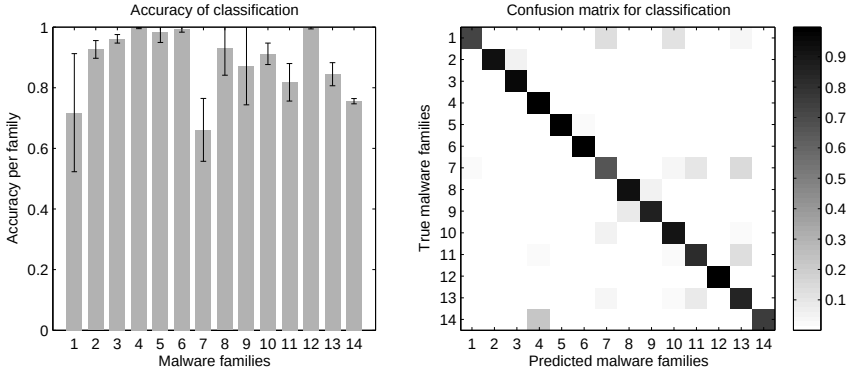
We now proceed to evaluate the performance and effectiveness of our methodology in different setups. For all experiments we pursue the following experimental procedure: The malware corpus of 10,072 samples introduced in Section 3.1 is randomly split into three partitions, a *training*, *validation* and *testing* partition. For each partition behavior-based reports are generated and transformed into a vectorial representation as discussed in Section 3. The training partition is used to learn individual SVM classifiers for each of the 14 malware families using different parameters for regularization and kernel functions. The best classifier for each malware family is then selected using the classification accuracy obtained on the validation partition. Finally, the overall performance is measured using the combined classifier on the testing partition.

This procedure, including randomly partitioning the malware corpus, is repeated over five experimental runs and corresponding results are averaged. For experiments involving data not contained in the malware corpus (Section 4.2 and 4.3), the testing partition is replaced with malware binaries from a different source. The machine learning toolbox *Shogun* [32] has been chosen as an implementation of the SVM. The toolbox has been designed for large-scale experiments and enables learning and classification of 1,700 samples per minute and malware family.

4.1 Classification of Malware Behavior

In the first experiment we examine the general classification performance of our malware behavior classifier. Testing data is taken from the malware corpus introduced in

Section 3.1. In Figure 2 the per-family accuracy and a confusion matrix for this experiment is shown. The plot in Figure 2 (a) depicts the percentage of correctly assigned labels for each of the 14 selected malware families. Error bars indicate the variance measured during the experimental runs. The matrix in Figure 2 (b) illustrates confusions made by the malware behavior classifier. The density of each cell gives the percentage of a true malware family assigned to a predicted family by the classifier. The matrix diagonal corresponds to correct classification assignments.



(a) Accuracy per malware family (b) Confusion of malware families

Fig. 2. Performance of malware behavior classifier using operation features on testing partition of malware corpus. Results are averaged over five experimental runs.

On average 88% of the provided testing binaries are correctly assigned to malware families. In particular, the malware families Worm.Allaple (4), Worm.Doomber (5), Worm.Gobot (6) and Worm.Sality (12) are identified almost perfectly. The precise classification of Worm.Allaple demonstrates the potential of our methodology, as this type of malware is hard to detect using static methods: Allaple is polymorphically encrypted, i.e., every copy of the worm is different from each other. This means that static analysis can only rely on small parts of the malware samples, e.g., try to detect the decryptor. However, when the binary is started, it goes through the polymorphic decryptor, unpacks itself, and then proceeds to the static part of the code, which we observe with our methodology. All samples express a set of shared behavioral patterns sufficient for classification using our behavior-based learning approach.

The accuracy for Backdoor.VanBot (1) and Worm.IRCBot (7) reaches around 60% and expresses larger variance – an indication for a generic AV label characterizing multiple malware strains. In fact, the samples of Worm.IRCBot (7) in our corpus comprise over 80 different mutex names, such as `SyMMeC`, `itcrew` or `h1dd3n`, giving evidence of the heterogeneous labeling.

4.2 Prediction of Malware Families

In order to evaluate how good we can even *predict* malware families which are not detected by anti-virus products, we extended our first experiment. As outlined in Section 3.1, our malware corpus is generated by collecting malware samples with the help of honeypots and spam-traps. The anti-virus engine Avira AntiVir, used to assign labels to the 10,072 binaries in our malware corpus, failed to identify additional 8,082 collected malware binaries. At this point, however, we can not immediately assess the performance of our malware behavior classifier as the *ground truth*, the true malware families of these 8,082 binaries, is unknown.

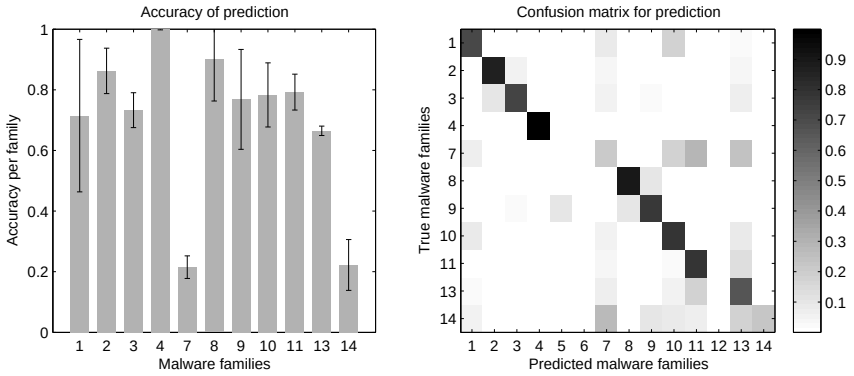
We resolve this problem by re-scanning the undetected binaries with the Avira AntiVir engine after a period of four weeks. The rationale behind this approach is that the AV vendor had time to generate and add missing signatures for the malware binaries and thus several previously undetected samples could be identified. From the total of 8,082 undetected binaries, we now obtain labels for 3,139 samples belonging to the 14 selected malware families. Table 2 lists the number of binaries for each of the 14 families. Samples for Worm.Doomber, Worm.Gobot and Worm.Sality were not present, probably because these malware families did not evolve and current signatures were sufficient for accurate detection.

Table 2. Undetected malware families of 3,139 samples, labeled by Avira AntiVir four weeks after learning phase. Numbers in brackets indicate occurrences of each Malware family.

1: Backdoor.VanBot	(169)	8: Worm.Korgo	(4)
2: Trojan.Bancos	(208)	9: Worm.Parite	(19)
3: Trojan.Banker	(185)	10: Worm.PoeBot	(188)
4: Worm.Allaple	(614)	11: Worm.RBot	(904)
5: Worm.Doomber	(0)	12: Worm.Sality	(0)
6: Worm.Gobot	(0)	13: Worm.SdBot	(597)
7: Worm.IRCBot	(107)	14: Worm.Virut	(144)

Based on the experimental procedure used in the first experiment, we replace the original testing data with the embedded behavior-based reports of the new 3,139 labeled samples and again perform five experimental runs.

Figure 3 provides the per-family accuracy and the confusion matrix achieved on the 3,139 malware samples. The overall result of this experiment is twofold. On average, 69% of the malware behavior is classified correctly. Some malware, most notably Worm.Allaple (4), is detected with high accuracy, while on the other hand malware families such as Worm.IRCBot (7) and Worm.Virut (14) are poorly recognized. Still, the performance of our malware behavior classifier is promising, provided that during the learning phase *none* of these malware samples was detected by the Avira AntiVir engine. Moreover, the fact that AV signatures present during learning did not suffice for detecting these binaries might also indicate truly novel behavior of malware, which is impossible to predict using behavioral patterns contained in our malware corpus.



(a) Accuracy per malware family (b) Confusion of malware families

Fig. 3. Performance of malware behavior classifier on undetected data using operation features. Malware families 5, 6 and 12 are not present in the testing data.

4.3 Identification of Unknown Behavior

In the previous experiments we considered the performance of our malware behavior classifier on 14 fixed malware families. In a general setting, however, a classifier might also be exposed to malware binaries that do *not* belong to one of these 14 families. Even if the majority of current malware families would be included in a large learning system, future malware families could express activity not matching any patterns of previously monitored behavior. Moreover, a malware behavior classifier might also be exposed to benign binaries either by accident or in terms of a denial-of-service attack. Hence, it is crucial for such a classifier to not only identify particular malware families with high accuracy, but also to verify the confidence of its decision and report unknown behavior.

We extend our behavior classifier to identify and reject *unknown behavior* by changing the way individual SVM classifiers are combined. Instead of using the maximum distance to determine the current family, we consider probability estimates for each family as discussed in Section 3.4. Given a malware sample, we now require *exactly one* SVM classifier to yield a probability estimate larger 50% and *reject* all other cases as unknown behavior.

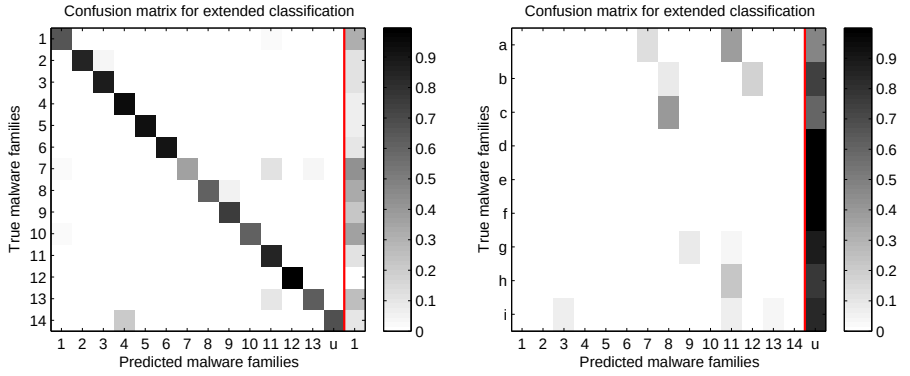
For evaluation of this extended behavior classifier we consider additional malware families not part of our malware corpus and benign binaries randomly chosen from several desktop workstations running Windows XP SP2. Table 3 provides an overview of the additional malware families. We perform three experiments: first, we repeat the experiment of Section 4.1 with the extended classifier capable of rejecting unknown behavior, second we consider 530 samples of the unknown malware families given in Table 3 and third we provide 498 benign binaries to the extended classifier.

Figure 4 shows results of the first two experiments averaged over five individual runs. The confusion matrices in both sub-figures are extended by a column labeled *u* which contains the percentage of predicted unknown behavior. Figure 4 (a) depicts the confusion matrix for the extended behavior classifier on testing data used in Sec-

Table 3. Malware families of 530 samples not contained in malware learning corpus. The numbers in brackets indicate occurrences of each malware family.

a: Worm.Spybot	(63)	f: Trojan.Proxy.Cimuz	(73)
b: Worm.Sasser	(23)	g: Backdoor.Zapchast	(25)
c: Worm.Padobot	(62)	h: Backdoor.Prorat	(77)
d: Worm.Bagle	(20)	i: Backdoor.Hupigon	(96)
e: Trojan.Proxy.Horst	(29)		

tion 4.1. In comparison to Section 4.1 the overall accuracy decreases from 88% to 76%, as some malware behavior is classified as unknown, e.g., for the generic AV labels of Worm.IRCBot (7). Yet this increase in false-positives coincides with decreasing confusions among malware families, so that the confusion matrix in Figure 4 (a) yields fewer off-diagonal elements in comparison to Figure 2 (b). Hence, the result of using a probabilistic combination of SVM classifiers is twofold: on the one hand behavior of some malware samples is indicated as unknown, while on the other hand the amount of confusions is reduced leading to classification results supported by strong confidence.



(a) Confusion on testing data (b) Confusion on unknown malware

Fig. 4. Performance of extended behavior classifier on (a) original testing data and (b) malware families not contained in learning corpus. The column labeled "u" corresponds to malware binaries classified as *unknown behavior*.

Figure 4 (b) now provides the confusion matrix for the unknown malware families given in Table 3. For several of these families no confusion occurs at all, e.g., for Worm.Bagle (d), Trojan.Proxy.Horst (e) and Trojan.Proxy.Cimuz (f). The malware behavior classifier precisely recognizes that these binaries do not belong to one of the 14 malware families used in our previous experiments. The other tested unknown malware families show little confusion with one of the learned families, yet the majority of these

confusions can be explained and emphasizes the capability of our methodology to not discriminate AV labels of malware but its behavior.

- Worm.Spybot (a) is similar to other IRC-bots in that it uses IRC as command infrastructure. Moreover, it exploits vulnerabilities in network services and creates auto-start keys to enable automatic start-up after system reboot. This behavior leads to confusion with Worm.IRCBot (7) and Worm.RBot (11), which behave in exactly the same way.
- Worm.Padobot (c) is a synonym for Worm.Korgo (8): several AV engines name this malware family Worm.Padobot, whereas others denote it by Worm.Korgo. The corresponding confusion in Figure 4 (b) thus results from the ability of our learning method to generalize beyond the restricted set of provided labels.
- Backdoor.Zapchast (g) is a network backdoor controlled via IRC. Some binaries contained in variants of this malware are infected with Worm.Parite (9). This coupling of two different malware families, whether intentional by the malware author or accidental, is precisely reflected in a small amount of confusion shown in Figure 4 (b).

In the third experiment focusing on benign binaries, all reports of benign behavior are correctly assigned to the unknown class and rejected by the extended classifier. This result shows that the proposed learning method captures typical behavioral patterns of malware, which leads to few confusions with other malware families but enables accurate discrimination of normal program behavior if provided as input to a classifier.

4.4 Explaining Malware Behavior Classification

The experiments in the previous sections demonstrate the ability of machine learning techniques to effectively discriminate malware behavior. In this section we examine the discriminative models learned by the SVM classifiers and show that relations of malware beyond the provided AV labels can be deduced from the learned classifiers. For each of the 14 considered malware families we learn an SVM classifier, such that there exist 14 hyperplanes separating the behavior of one malware family from all others. We present the learned decision functions for the Sality and Doomber classifiers as outlined in Section 3.5 by considering the most prominent patterns in their weight vectors.

Sality Classifier Figure 5 depicts the top five discriminating operation features for the family Worm.Sality learned by our classifier. Based on this example, we see that operation features can be used by a human analyst to understand the actual behavior of the malware family, e.g., the first two features show that Sality creates a file within the Windows system directory. Since both variants created during the preprocessing step (see Section 3.3 for details) are included, this indicates that Sality commonly uses the source filename `vcmgcd32.dl_`. Moreover, this malware family also deletes at least one file within the Windows system directory. Furthermore, this family creates a mutex containing the string `kuku_joker` (e.g., `kuku_joker_v3.09` as shown in Figure 5 and

```
0.0142: create_file_2 (srcpath="C:\windows\...")
0.0073: create_file_1 (srcpath="C:\windows\...", srcfile="vcmgcd32.dl_")
0.0068: delete_file_2 (srcpath="C:\windows\...")
0.0051: create_mutex_1 (name="kuku_joker_v3.09")
0.0035: enum_processes_1 (apifunction="Process32First")
```

Fig. 5. Discriminative operation features extracted from the SVM classifier of the the malware family *Sality*. The numbers to the left are the sorted components of the hyperplane vector *w*.

```
0.0084: create_mutex_1 (name="GhostBOT0.58c")
0.0073: create_mutex_1 (name="GhostBOT0.58b")
0.0052: create_mutex_1 (name="GhostBOT0.58a")
0.0014: enum_processes_1 (apifunction="Process32First")
0.0011: query_value_2 (key="HKEY_LOCAL...\run", subkey_or_value="GUARD")
```

Fig. 6. Discriminative operation features extracted from the SVM classifier of the the malware family *Doomber*. The numbers to the left are the sorted components of the hyperplane vector *w*.

kuku_joker_v3.04 as sixth most significant feature) such that only one instance of the binary is executed at a time. Last, Sality commonly enumerates the running processes.

Based on these operation features, we get an overview of what specific behavior is characteristic for a given malware family; we can *understand* what the behavioral patterns for one family are and how a learned classifier operates.

Doomber Classifier In Figure 6, we depict the top five discriminating operation features for Worm.Doomber. Different features are significant for Doomber compared to Sality: the three most significant components for this family are similar mutex names, indicating different versions contained in our malware corpus. Furthermore, we can see that Doomber enumerates the running processes and queries certain registry keys.

In addition, we make another interesting observation: our learning-based system identified the mutex names GhostBOT-0.57a, GhostBOT-0.57 and GhostBOT to be among the top five operation features for Worm.Gobot. The increased version number reveals that Gobot and Doomber are closely related. Furthermore, our system identified several characteristic, additional features contained in reports from both malware families, e.g., registry keys accessed and modified by both of them. We manually verified that both families are closely related and that Doomber is indeed an enhanced version of Gobot. This illustrates that our system may also help to identify *relations* between different malware families based on observed run-time behavior.

5 Limitations

In this section, we examine the limitations of our learning and classification methodology. In particular, we discuss the drawbacks of our analysis setup and examine evasion techniques.

One drawback of our current approach is that we rely on one single program execution of a malware binary: we start the binary within the sandbox environment and observe one execution path of the sample, which is stopped either if a timeout is reached or if the malware exits from the run by itself. We thus do not get a full overview of what the binary intends to do, e.g., we could miss certain actions that are only executed on a particular date. However, this deficit can be addressed using a technique called *multi-path execution*, recently introduced by Moser et al. [23], which essentially tracks input to a running binary and selects a feasible subset of possible execution paths. Moreover, our results indicate that a single program execution often contains enough information for accurate classification of malware behavior, as malware commonly tries to aggressively propagate further or quickly contacts a Command & Control servers.

Another drawback of our methodology is potential evasion by a malware, either by detecting the existence of a sandbox environment or via mimicry of different behavior. However, detecting of the analysis environment is no general limitation of our approach: to mitigate this risk, we can easily substitute our analysis platform with a more resilient platform or even use several different analysis platforms to generate the behavior-based report. Second, a malware binary might try to mimic the behavior of a different malware family or even benign binaries, e.g. using methods proposed in [12, 37]. The considered analysis reports, however, differ from sequential representations such as system call traces in that multiple occurrences of identical activities are discarded. Thus, mimicry attacks can not arbitrarily blend the frequencies or order of operation features, so that only very little activity may be covered in a single mimicry attack.

A further weakness of the proposed supervised classification approach is its inability to find structure in new malware families not present in a training corpus. The presence of unknown malware families can be detected by the rejection mechanism used in our classifiers, yet no further distinction among rejected instances is possible. Whether this is a serious disadvantage in comparison to clustering methods is to be seen in practice.

6 Conclusions

The main contribution of this paper is a learning-based approach to automatic classification of malware behavior. The key ideas of our approach are: (a) the incorporation of labels assigned by anti-virus software to define classes for building discriminative models; (b) the use of string features describing specific behavioral patterns of malware; (c) automatic construction of discriminative models using learning algrithms and (d) identification of explanatory features of learned models by ranking behavioral patterns according to their weights. To apply our method in practice, it suffices to collect a large number of malware samples, analyse its behavior using a sandbox environment, identify typical malware families to be classified by running a standard anti-virus software and construct a malware behavior classifier by learning single-family models using a machine learning toolbox.

As a proof of concept, we have evaluated our method by analyzing a training corpus collected from honeypots and spam-traps. The set of known families consisted of 14 common malware families; 9 additional families were used to test the ability of our method to identify behavior of unknown families. In an experiment with over

3,000 previously *undetected* malware binaries, our system correctly predicted almost 70% of labels assigned by an anti-virus scanner *four weeks later*. Our method also detects unknown behavior, so that malware families not present in the learning corpus are correctly identified as unknown. The analysis of prominent features inferred by our discriminative models has shown interesting similarities between malware families; in particular, we have discovered that Doomber and Gobot worms derive from the same origin, with Doomber being an extension of Gobot.

Despite certain limitations of our current method, such as single-path execution in a sandbox and the use of imperfect labels from an anti-virus software, the proposed learning-based approach offers the possibility for accurate automatic analysis of malware behavior, which should help developers of anti-malware software to keep apace with the rapid evolution of malware.

Bibliography

[1] Microsoft Security Intelligence Report, October 2007. http://www.microsoft.com/downloads/details.aspx?FamilyID=4EDE2572-1D39-46EA-94C6-4851750A2CB0.

[2] Avira. AntiVir PersonalEdition Classic, 2007. http://www.avira.de/en/products/personal.html.

[3] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of the 9th Symposium on Recent Advances in Intrusion Detection (RAID'06)*, pages 165–184, 2006.

[4] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th Symposium on Recent Advances in Intrusion Detection (RAID'07)*, pages 178–197, 2007.

[5] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A tool for analyzing malware. In *Proceedings of EICAR 2006*, April 2006.

[6] U. Bayer, A. Moser, C. Kruegel, and E. Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2:67–77, 2006.

[7] C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

[8] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium*, pages 12–12, 2003.

[9] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2007.

[10] M. Christodorescu, S. Jha, S. A. Seshia, D. X. Song, and R. E. Bryant. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*, pages 32–46, 2005.

[11] H. Flake. Structural comparison of executable objects. In *Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'04)*, 2004.

[12] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the 15th USENIX Security Symposium*, pages 241–256, 2006.

[13] G. C. Hunt and D. Brubacker. Detours: Binary interception of Win32 functions. In *Proceedings of the 3rd USENIX Windows NT Symposium*, pages 135–143, 1999.

[14] X. Jiang and D. Xu. Collapsar: A VM-based architecture for network attack detention center. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[15] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137 – 142. Springer, 1998.

[16] T. Joachims. *Learning to Classify Text using Support Vector Machines*. Kluwer Academic Publishers, 2002.

[17] M. Karim, A. Walenstein, A. Lakhotia, and P. Laxmi. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1–2):13–23, 2005.

[18] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer. Behavior-based spyware detection. In *Proceedings of the 15th USENIX Security Symposium*, pages 19–19, 2006.

[19] J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7(Dec):2721 – 2744, 2006.

[20] C. Kruegel, W. Robertson, and G. Vigna. Detecting kernel-level rootkits through binary analysis. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC)*, 2004.

[21] T. Lee and J. J. Mody. Behavioral classification. In *Proceedings of EICAR 2006*, April 2006.

[22] C. Leita, M. Dacier, and F. Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *Proceedings of the 9th Symposium on Recent Advances in Intrusion Detection (RAID'06)*, Sep 2006.

[23] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of 2007 IEEE Symposium on Security and Privacy*, 2007.

[24] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*, 2007. to appear.

[25] Norman. Norman sandbox information center. Internet: `http://sandbox.norman.no/`, Accessed: 2007.

[26] J. Platt. Probabilistic outputs for Support Vector Machines and comparison to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 2001.

[27] F. Pouget, M. Dacier, and V. H. Pham. Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In *ECCE'05, E-Crime and Computer Conference, 29-30th March 2005, Monaco*, Mar 2005.

[28] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9(Jan):23–48, 2008.

[29] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[30] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[31] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[32] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Maching Learning Research*, 7:1531–1565, 2006.

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

[34] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[35] Virus Bulletin. AVK tops latest AV-Test charts, August 2007. http://www.virusbtn.com/news/2007/08_22a.xml.

[36] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *SIGOPS Oper. Syst. Rev.*, 39(5):148–162, 2005.

[37] D. Wagner and P. Soto. Mimicry attacks on host based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, pages 255–264, 2002.

[38] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2), 2007.