Towards Automating Malware Classification and Characterization

Alexander K. Seewald
Seewald Solutions, 1180 Vienna, Austria
alex@seewald.at

Abstract:
Spam has become a problem of global impact. Most spam messages are currently sent out by captured machines organized in bot networks, which are infected with malicious software and are therefore under direct control of spammers. The connected explosion of automatically generated new malware variants has manual analysis at a great disadvantage, while classical automated analysis systems have problems keeping up with the diversity of new variants.

Here, we propose using machine learning approaches to learn global (i.e. malware intent) and local (i.e. specific functionality) malware properties based on behavioral traces of malware recorded in virtual environments, and test them on a small corpus. Initial results are somewhat promising, so we also discuss areas for improvement as well as current and future challenges.

1 Introduction

Undoubtedly Spam is a problem of global impact that keeps getting bigger. Already in 2001, according to a study undertaken for the European Commisson [GD01], Internet subscribers worldwide wasted an estimated 10 billion Euro per year just in connection costs due to Spam. Large as the economic impact may be, it is only part of the problem – waste of time, resources and the gradual erosion of trust in EMail communications must also be considered.

Spam filters are reasonably effective now, but are slowly becoming less effective with new forms of image spam and audio-based spam which will in time prove to be a challenge for today's most widely used bayesian filters. Spammers have adapted again and again to effective adversaries and countermeasures. This is likely to continue in the future.

Most if not all of today's Spam is sent via large networks of captured machines which are under direct control of the Spammers. These large networks have been named *botnets*, from software ro*bot net*works. One of the largest spamfilter companies, SpamHaus, has estimated that already in 2004, 70% of spam was sent out via such networks [NI05]. Any unsecured computer attached to the internet may be infected rapidly – via known security holes, by having users clicking or only viewing infected mails or via surfing past an appropriately prepared webpage. Even secure computers may be prone to zero-day attacks, which rely on the delay between the publication of a weakness and the availability of a

patch to correct it.

By distributing their workload among several hundreds or thousands of machines, spammers are able to utilize combined bandwidth of gigabits per second to send out spam rapidly. Typically, such botnets will stay active only for a short time period – half of them last less than a day. This bandwidth is also used for Denial-of-Service attacks which can bring down whole internet backbones and have been used to attack the internet infrastructure of small countries.

The ease with which it is possible to create new malware variants to infect machines for botnets has human analysts at a great disadvantage. While spammers are able to automatically generate new malware variants in unprecedented numbers, out of necessity the anti-virus community has become focussed on malware detection. It clearly has become infeasible to analyze manually what each of the thousands of new variants appearing each and every hour are capable of in detail. Botnet software nowadays is capable to automatically download updates from the internet to stay ahead of anti-virus detection, so this analysis would have to be done in near real-time as well. However, with only detection by antivirus software, it becomes almost impossible to tell what a given malware will do *before* it spreads in the wild.

One approach to deal with this is automated analysis of malware program behaviour in virtual environments, so-called sandboxes. While these are not complete solutions[1], they remain the current best approach. However, the analysis of recorded malware behaviour still utilizes hand-crafted software and a lot of human expertise, and thus is not expected to scale as well or as far as the spammers' automated approaches.

Here, we propose and test a different approach to analyze recorded malware behaviour. Based on a small corpus of malware with known high-level behaviour patterns taken from detailed virus descriptions, which have been established by human domain experts via thorough manual analysis of malware behavior, we formulate the task of malware characterization as a machine learning problem. Thus, human expertise can be leveraged to create scalable automated systems at some cost in form of reduced accuracy. Our results are promising and show that this approach is generally feasible.

2 Related Research

Because of the wealth of related research in this area, we group it into two distinct area: those dealing with static code analysis of unpacked, decrypted executables; and those dealing with dynamic code analysis in a simulated environment (similar to our approach). Both approaches have their merits and disadvantages: the former scales extremely well, but automated unpacked and decryption is not always possible and will become harder in the future. The latter does not scale as well, but is superior in characterizing new, unknown malware regardless of its packaging.

[1] Malware is potentially able to detect whether or not it runs in a specific sandbox, and this is practically impossible to prevent. This is also true for hardware sandboxes.

2.1 Dynamic Code Analysis

[Wi05] proposes a behavior-based approach to distinguish between different malware variants and non-malware programs. He uses high-level behavioral features such as *survives_reboot* or *stealth* (i.e. running in a hidden window or hiding from the task manager) which are based on automatic data collection of process traces. This approach works remarkably well even with only ten specific feature detectors and a Naive Bayes classifier, which is either due to the choice of feature detectors or due to the integration of non-malware executables – something which we intend to look into in the future.

[LM06] propose to apply machine learning algorithms to learn virus family classification from event data collected by a distributed sandbox system. This is similar to the first experiment on Malware classification within this paper. As we did, they also found that instance-based learning – albeit with a costly edit-distance based distance measure between event sequences – performs quite well for this task. Our results are most similar to Experiment B, 44 clusters, 100 events, and a bit worse at 26.6% versus their 21.06% error rate. Their event-type features seem promising especially for a high number of events, where the error rate could be reduced to 7.8%, but at greatly increased computational cost. Note however that our classification was on the level of global behavior patterns and malware intent, where we would expect far more variance than on the level of individual virus families. Thus our learning task is much harder and biases this comparison.

[Ke97] from IBM's Digital Immune Systems research have prophesized the rise of botnets a decade ago. They describe an integrated system for automated malware detection, identification, behavioral characterization and removal developed by IBM in the 1990ies, which still reads amazingly current. For example, their description of an artificial immune system for computers strongly resembles the approach used by anti-virus companies all over the world. However, they were far too optimistic concerning the applicability of fully automated methods. The paper does not give technical details concerning their methods nor sufficiently detailed results, so we cannot compare it to our approach.

2.2 Static Code Analysis

[Gh05] presents an evaluation of instance-based learning with three different distance measures, on a set of binary malware samples. They propose using a bloom filter plus edit distance to speed up searches for nearest neighbors, which seems a reasonable approach as it is much more scalable than edit distance on its own. While they show that the evolutionary relationsship of one small, well-defined virus family with 10 members can be reconstructed appropriately with this approach, it remains to be seen whether the reconstruction of large-scale malware evolution is as feasible. As they note, executable packing and encryption of malware defeats this system and must be compensated before applying it.

[We07] proposes to use instance-based learning using normalized compression distance (NCD) as distance measure, both for malware executable and malicious traffic charac-

terization. They also visualize a clustering based on the obtained distance matrix and show that it similar variants usually cluster closely together. UPX-style executable packing impairs the system's performance, so more complex packing and encryption is likely to defeat it. The effect of a good encryption, after all, is to create a binary stream with maximum entropy and thus minimum compressibility, so all encrypted malware samples are expected to seem very similar by NCD, regardless of the underlying code.

[DR05] present a graph-based system to determine differences between malware variants with a static code analysis, which has been applied to analyze changes between malware variants. This approach can be considered complementary to the dynamic run-time code analysis which we use, and could easily be combined. Again, executable packing and malware encryption must be undone before applying this method.

[KM07] also describe a system using machine learning techniques and static code analysis with 4-gram overlapping byte sequences as features, which distinguishes malware from non-malware (similar to [Wi05] – arguably a much easier task than classifying malware variants), and has also been applied with less success to the determination of malware subtypes mass-mailer, backdoor and executable virus. This approach is again complementary to ours, and may be combined. They also cite as main impediment for further analysis the lack of availability of high-quality high-level behavioral classification data for malware, which is what we also found. Although at first glance the performance of this system seems impressive for a purely static analysis, it fails to live up to expectations: when applying a similar approach to our malware samples, we have found that its performance by error rate and Area-under-ROC (AUC) is always significantly worse than our approach. This might be explained by their use of old (2003) malware samples, where code obfuscation techniques such as executable packing and encryption were far less widespread. Again, these should be compensated before applying this method.

[Ov04] noted that bayesian filters are useful for spam and malware filtering – in fact, for classifying all types of content – and demonstrated their usefulness on his own mailbox. His conclusions are still valid, and in the meantime new variants of bayesian and SVM learning using n-gram features yield state-of-the-art performance without any email preprocessing. Incidentially, the author's own spamfilter system uses a simple bayesian filter for both spam and malware without utilizing a separate virus scanner.

3 Experimental Setup

As mentioned, we propose an approach to analyze malware behaviour based on recorded executation traces in a sandbox virtual environment. We focus both on classifying new, unknown malware into a set of existing groups based on their overall behaviour, as well as learning more specific high-level behaviour descriptions – i.e. *characterizing* malware behaviour. Both are usually considered to be tasks for human domain experts, and in fact the manual analysis of malware yields the highest number of useful hints for countermeasures, as domain experts continue learning to recognize and undo the spammers obfuscation techniques. However, this approach is generally not scalable, as the number of newly

Name	Description
fsize	size of malware executable
status	status at end of execution session (=dead or alive)
exit_code	exit code (if status==dead)
regRead	number of registry entries read
regModified	number of registry entries modified
regCreated	number of new registry entries created
propPingSuccess	prop. of outgoing pings that were successful
numPing	number of pings send out
port139	outgoing TCP traffic conversations to port 139
port445	outgoing TCP traffic conversations to port 445
port25	outgoing TCP traffic conversations to port 25 (SMTP)
dead:0	no. of subprocesses dead with exit-code 0 at end of session
dead:0S	no. of services dead with exit-code 0 at end of session
dead:1	no. of subprocesses dead with non-zero exit-code
dead:1S	no. of services dead with non-zero exit-code
alive:0	no. of processes alive at end of session
alive:0S	no. of services alive at end of session
num2nd	no. of subprocesses started
numService	no. of services started

Table 1: Full list of features derived from a trace of malware activities.

appearing malware variants is on the order of thousands per hour and still growing.

As a representative set of malware software, we have chosen to use the December 2006 wildlist[2] from WildList Organization International, which maintains a list of computer viruses that are most prevalent in the wild (i.e. circulating widely on the internet). Their wide distribution makes them dangerous, and as such they are a better sample for our approach than e.g. recent samples collected by honeypots[3], which may or may not survive on the internet. We collected malware samples for all but three samples on the wildlist, and this formed the malware corpus for all remaining experiments.

As virtual sandbox environment we chose Anubis.[4] The rationale for choosing a newly developed, relatively unknown sandbox environment was that it was quite unlikely that currently circulating malware had already adapted to this new environment. For this analysis, the malware was allowed to *call home* (i.e. had full access to the internet) from an anonymous IP address. We automatically pushed all malware executables through the publicly available webform, and recorded behavioral traces for a two minutes execution of each executable. Note that our approach could be used with different sandbox environments – even data from multiple sandboxes – and would also be able to utilize data from several runs of the same executable.

[2]Data collection for this paper dates back to 04/2007, and we were intentionally selecting a few months old wildlist to allow virus researchers at least some time to do detailed analysis on newly appearing variants. Wildlist Dec. 2006 see `http://www.wildlist.org/WildList/200612.htm`

[3]Machines which simulate known vulnerabilities to attract attacks to collect currently circulating malware.

[4]`http://analysis.seclab.tuwien.ac.at`, previously known as *TTAnalyze*.

Samples	Group ID	Content
164	0	Mass Mailing Wurm
119	1	Öffnet ein Backdoor
2	1	Öffnet ein Backdoor auf dem infi zierten System
1	1	Öffnet ein Backdoor auf TCP Port 8181
90	2	Besitzt eigene SMTP Engine
3	2	Benutzt seine eigene SMTP Engine
1	2	Sendet Nachrichten in Deutsch und Englisch
1	2	Sendet mehrsprachige Email Nachrichten
1	2	Sendet Emails in verschiedenen Sprachen
53	3	Nutzt bekannte Schwachstellen aus
10	3	Nutzt bekannte Sicherheitslücken aus
20	4	Verringert die System Sicherheit
1	4	Verringert Sicherheits- Einstellungen
1	4	Verringert die Systemsicherheit
1	4	Verringert die Sicherheit des Systems
3	4	Verändert die System Einstellungen
20	5	Verbreitet sich mit Hilfe von Instant Messengern
6	5	Benutzt den MSN Messenger um sich zu verbreiten
1	5	Verbreitet sich über Netzwerkfreigaben und den AOL Messenger
20	6	Beendet vordefi nierte Prozesse
8	6	Beendet Prozesse
1	6	Deaktiviert Programme
6	6	Beendet Prozesse und Services
14	7	Downloader Trojaner
3	7	Lädt Dateien aus dem Internet herunter
3	7	Lädt Dateien herunter und führt diese aus
8	8	Installiert sich in die Registry
4	8	Ändert Registry Einträge:
4	9	Sammelt Email Adressen
4	9	Durchsucht das System nach Email Adressen
3	9	Sucht nach Email Adressen auf dem infi zierten System
3	9	Durchsucht das System nach Emailadressen
7	10	Verbreitet sich über File Sharing Netzwerke
1	10	Versucht File Sharing Netzwerke auszunutzen
1	10	Nutzt File Sharing Netzwerke aus um sich zu verbreiten
1	10	Benutzt File Sharing Netzwerke um sich zu verbreiten
1	11	Verbreitet sich über Netzwerkfreigaben
1	11	Verbreitet sich über Netzwerkfreigaben und den AOL Messenger
1	12	Überschreibt .exe Dateien
1	12	Infi zierte .exe Dateien
3	13	Verbreitet sich über Netzwerkverbindungen
1	13	Verbreitet sich über Netzwerke
3	14	Infi ziert und löscht Dateien
3	14	Infi ziert Prozesse

Table 2: Grouped short descriptions (only available in German)

In preprocessing, we mapped the very comprehensive XML-based behaviour descriptions to a table-based representation containing features such as the malware executable size (*fsize*), the number of registry entries read (*regRead*) and the number of TCP conversations corresponding to accessing ports 139, 445 and 25 on foreign machines (*port139,445,25* – corresponding to access points for one well-known vulnerability and the SMTP protocol port for sending email). Also, *propPingSuccess* tells us very roughly if the pings are sent to random addresses on the network, or whether the malware uses a more intelligent approach. Table 1 shows the full feature list. We intentionally kept the feature list short to obtain a baseline of what can already be achieved with relatively simple features.

For behavioral classification and characterization, we chose to use publicly available data from a local anti-virus company. Consistency dictates that data from a single company, preferably from a single malware analyst, should be used. The chosen company had a small analytics department consisting of four people, and was in this sense better suited than larger companies. On the downside, the number of available detailed virus descriptions was therefore relatively small.

For behavioral classification, we chose to use the first part of the official virus name as given by their most recent virus scanner. This yields a rough grouping into global be-

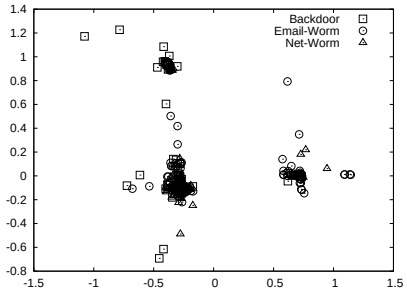

Figure 1: 2D visualization of three classes via Sammon Mapping.

havioral categories based on main malware intent such as *Backdoor*, *Net-Worm* or *Trojan-Dropper*.

For behavioral characterization, we chose to use their web-based virus lexicon, which gave detailed descriptions for roughly 10% of the malware from our corpus. In each case, the descriptions were written by human domain experts and were based on a thorough analysis of the corresponding malware executable. We focussed on the short descriptions, which mentioned major characteristics of each virus, such as *Opens a Backdoor*, *Has its own SMTP engine* or *Mass Mailing Worm* – the latter corresponds to a worm used to send spam – and grouped them into 15 categories. Table 2 shows the established grouping of German sentence fragments from this short description. We only used groups 0 to 3 for the experiments, because the number of training samples was deemed to be too small for the rest of them.

In preliminary experiments, we tested many different learning algorithms, including Naive Bayes, Logistic Regression, Instance-based learning, and the rule learner JRip [Co95]. Instance-based learning [AK91] with one nearest neighbor – which classifies each sample by the known classification of its nearest neighbor from the training set by computing euclidean distance between feature vectors – offered the best performance. As the model of instance-based learning is not easily understandable, we also show more comprehensible but less accurate results of the JRip rule learner. All machine learning experiments were done within WEKA [FW05] using ten-fold cross-validation. XML data and text preprocessing was done in Perl.

4 Results

Here, we present the results of our experiments. First, we present the malware classification results based on the first part of the virus name, which relies on malware identification by virus scanner – in most cases without a detailed analysis of the malware having been involved. Afterwards, we present more precise behaviour characterization based on grouped short descriptions of malware which *has* been analyzed in detail by domain experts.

IM-Worm	Worm	Email-Worm	Generic	Trojan-Downloader	Net-Worm	Backdoor	Trojan-Dropper	True Class
10	0	5	0	0	12	1	1	**IM-Worm**
0	39	6	0	0	2	0	1	**Worm**
2	9	242	3	7	15	12	0	**Email-Worm**
0	0	3	7	0	0	0	0	**Generic**
0	0	10	0	31	6	2	0	**Trojan-Downloader**
12	5	12	0	5	132	7	4	**Net-Worm**
3	0	16	0	2	14	73	2	**Backdoor**
2	1	3	0	0	9	1	4	**Trojan-Dropper**

Table 3: Confusion matrix of IBk on malware classification dataset via ten-fold cross-validation.

4.1 Malware Classification

For Malware classification, we have used the first part of each virus name, which corresponds to a global category. We removed all categories with less than 10 entries, leaving 733 samples of the original 789, and eight categories.

As we mentioned, the best learning algorithm on this dataset was instance-based learning with $K = 1$ nearest neighbors. This classifier achieves an error rate of 26.6% on eight classes. Baseline error rate, i.e. always predicting the most common class, was 60.4%. Table 3 shows the full confusion matrix based on a ten-fold crossvalidation. The rows and row label in the last column correspond to true classes, and the columns correspond to predicted classes in the test fold of the cross-validation. A perfect classification would have all off-diagonal elements equal to zero. Here, we can see that this is not the case: there is confusion between several classes (e.g. *Net-Worm*, *Email-Worm* and *IM-Worm*[5]). This might have been expected as most current malware has multiple functionalities, and these labels are assigned only after a cursory analysis, mostly based on malware code signatures – and all worms show similar behaviour to some extent. *Trojan-Dropper*, for example, is most often classified as *Net-Worm*, as both show similar behaviours: propagation through known security flaws, and installation of malware which may download additional functionality from the internet. It may also be that the definition of principal names is not as clear-cut as we would like it to be.

Figure 1 demonstrates this by showing a 2D visualization of three classes via Sammon mapping [Sa69]: there are three distinct clusters, but they do not always correspond to classes: the top left cluster is mainly *Backdoor*, the bottom right cluster is mainly *Email-Worm*, but the bottom left cluster is an almost equal mix of all three classes. Clearly the short names do not fully explain the behaviour of a given malware.

4.2 Malware Behaviour Characterization

Our previous results in malware classification indicated that the principal name of a malware might not be the best way to characterize its behavior. As we mentioned, this name is mostly given after a cursory analysis of the malware executable for purposes of signature generation, is based on similarity to other signatures already present in the virus database, and there can be only one principal name even if the malware shows multiple

[5]Instant Messenger Worm – viruses propagating via MSN or AOL instant messaging systems.

functionalities (such as Trojan-Downloader, IM-Worm, and Backdoor for one typical botnet malware). All of this makes the principal name less suitable as indicator for overall behaviour patterns.

This motivated us to aim for a more precise behaviour characterization. To achieve this, we selected only those malware executables for which a detailed entry in the virus database existed – 257 of the initial 789. In some cases, more than one malware executable was referenced by one virus database entry, in which case we assumed that the entry data applied to all of them. Since the main text proved to be too variable for text analysis without the major effort of creating an appropriate textual corpus, we focussed on analyzing only the short description. This description consisted of several lines with short sentence fragments indicating specific behaviours and functionalities of the malware, such as *Has its own SMTP engine* or *Spreads via known security holes*. We hoped that this more precise characterization might be more useful for our task.

We used groups 0-3 from Table 2 for our experiments. This created four distinct learning tasks with different baseline error rates. Except for group 3 (*Spreads via known security holes*), IBk proved to give the smallest error rate as estimated by ten-fold cross-validation.

Group 0 (*Mass Mailing Worm*) had a baseline error of 36.2%. The error of IBk was better at 26.1%. In retrieving the malware samples with this behaviour among all samples, a precision of 0.777 and a recall of 0.829 could be obtained. JRip was only slightly worse at 28.0% (p=0.735, r=0.878), and obtained the following rules:

```
(fsize<=61440) && (fsize>=31358) && (regCreated==1) => No.
(fsize>=210432) => No.
=> Yes.
```

Group 1 (*Opens a Backdoor*) had a baseline error of 47.5%. IBk was much better at 19.5% (p=0.790, r=0.803). JRip performed worse at 27.6% (p=0.701, r=0.730) and obtained the following rules:

```
(fsize>=63488)    && (port25>=2) => Yes.
(regCreated>=2)   && (fsize>=102400) => Yes.
(port445>=1) => Yes.
(fsize<=27403)    && (regModified>=16) => Yes.
(regModified<=11) && (status==dead) && (regRead=>11) => Yes.
=> No.
```

Group 2 (*Has its own SMTP engine*) had a baseline error of 37.0%. IBk was better at 24.5% (but with lower recall and precision of p=0.670, r=0.663); JRip performed only slightly worse than IBk at 26.8% (p=0.648, r=0.600) and obtained the following rules:

```
(regCreated>=2) && (regRead>=92) => Yes.
(regCreated>=4) => Yes.
(fsize<=28160)  && (fsize>=19052) => Yes.
=> No.
```

Finally, Group 3 (*Spreads via known security holes*) had a baseline error of 24.5%. IBk only performed slightly better at 20.2% (p=0.582, r=0.619). JRip was the best learning algorithm here at 18.7% error rate (p=0.612, r=0.651) with the following rules:

```
(port445>=1) => Yes.
(regModified<=11) && (fsize>=83456) && (regRead>=7) => Yes.
=> No.
```

While these results are not as impressive as we had hoped, they show that the basic principle is sound. Our features only include basic port traffic features for two ports known to be involved in one vulnerability, which probably explains the bad performance (vs. baseline) for *Spreads via known security holes*. More detailed features, which could also analyze the traffic going through the respective ports, would clearly be able to uncover more attack attempts. Note that the file size, *fsize*, features prominently, which indicates that some features from static code analysis may be useful.

By its construction IBk does not return continuous class probabilities to construct recall/precision curves, so we cannot analyze these to determine the trade-off between recall and precision to test how much work the proposed system could save. The same applies to a lesser but still sufficient extent to JRip.

5 Discussion & Future Work

The fact that IBk works best in our experiments indicates that behavioural clusters are still mainly found by referring to feature similarity with known malware variants. This is probably due to our relatively simple feature set, which is directly based on the recorded behavioral traces and does not utilize more complex pattern detectors such as passive traffic analysis systems, packet timing data and known traffic meta-patterns such as those used in [Wi05]. The sparsity of Fig. 1 also indicates that the feature variance is too low. One main way to improve the current system would therefore be to enrich the feature representation which these and other pattern detectors.

We believe that with a richer feature representation, JRip would become more useful in finding non-trivial patterns. While its performance is currently slightly worse than that of IBk, its small set of easily understandable rules make it more acceptable to domain experts, and allow its obtained knowledge to be integrated into classical malware analysis systems. This might prove to be more useful for the near future, until it is feasible to build systems which are competitive to domain experts.

Another related issue which we believe needs to be addressed is that virus database entries, while including a wealth of information, might not be very well suited for behavioual analysis at all. The meaning of keyphrases is not explained and in some cases not unambiguously clear, the description focusses on actionable knowledge (such how to remove the malware from an infected system) which is irrelevant for purposes of the proposed approach, and the behaviour of the malware is generally not specified in sufficient detail. A better approach might be to work together with state-of-the-art anti-virus companies and

– in cooperation with their domain experts – define a way to specify malware behaviour at the desired granularity level in sufficient detail. This would give useful hints as to which patterns are utilized by humans in classifying malware, as well as yielding a high-quality corpus of training data for research purposes. Here, it would be nice if anti-virus companies were more forthcoming in terms of white papers with detailed descriptions of their current approaches to analyze malware, although such data should as a rule probably not be generally available – spammers could utilize it as well.

While two minutes runtime is rather short for tracking malware behaviour, and indeed we did not find a specific example of any malware sending spam, it seems that longer periods do not improve upon this: we ran five suspicious variants for varying times up to eight hours, but again failed to find a single sample of spam sent out. However, it might be that we were simply unlucky in that no botnet operator was trying to send spam during this period. On the other hand it remains to be investigated which specific malware variants are used to send spams via botnets, and whether these are still catched by current honeypots.

Last but not least we also need to address the ongoing problem of sandbox detection code, which allows malware to determine whether or not it runs in a sandbox and take appropriate steps to obfuscate its purpose or otherwise mislead and confuse the analytics system. Learning systems might be able to push the frontier out a bit farther, as can physical hardware sandboxes, but the main problem does need to be addressed. An alternative way to analyse malware is in the wild, where out of necessity it will generate traffic. A passive analysis of this traffic on a sufficiently large scale might be as useful in determining malware behaviour as the local sandbox approach, would scale reasonably well, and cannot be circumvented that easily by spammers. We are currently working on just such an approach within a funded one-year research project. Complementary to this, a static analysis of malware code might be useful as well, provided that encryption and executable packing of malware executable can be automatically and reliably reversed.

6 Conclusion

We have proposed a new approach to classify and characterize malware behaviour patterns based on analyzing behavioral traces recorded during execution in a virtual environment (sandbox) with machine learning methods. We have tested our approach on publicly available data, and have found the basic principle to be sound.

Both global behavioral patterns (i.e. malware intent) and local behavioral patterns (i.e. specific malware functionality, such as *Has its own SMTP server*) were analyzed. Performance was much better than the baseline in most cases.

We have identified major areas to improve the system, which might be of interest to other researchers working in this field, and noted in what way they could be approached, as well as pointing out major present and future challenges in this field, which will also need to be addressed in the near feature. We are looking forward to continuing work on these and other issues within the spam and botnet research community.

6.1 Acknowledgments

This work was funded by the Internet Privatstiftung Austria (IPA) within the *Frühwarnsystem für Botnetze* (early warning system for botnets) project. We would also like to thank Ulrich Bayer for his support in using the Anubis sandbox system, and for contributing the execution traces in its native XML format.

References

[AK91] D. Aha, D. Kibler (1991). Instance-based learning algorithms. Machine Learning. 6:37-66.

[Co95] W.W. Cohen: Fast effective rule induction. In Proceedins of the Twelfth International Conference on Machine Learning, pp.115–123, San Francisco, CA, 1995. Morgan Kaufmann.

[DR05] T. Dullien and R. Rolles, *Graph-Based Comparison of Executable Objects*, Symposium sur la Securite des Technologies de l'Information et des Communications, 2005.

[FW05] Frank E., Witten I.H. (2005) *Data Mining - Practical Machine Learning Tools and Techniques (2nd ed)*. Morgan Kaufmann / Elsevier, 2005.

[GD01] Serge Gauthronet and Etienne Drouard. *Unsolicited commercial communications and data protection*. Technical report, Commission of the European Communities, 2001.

[Gh05] Gheorghescu, M. (2005): An automated virus classification system, Proceedings of the Virus Bulletin (VB) Conference 2005, Dublin, Ireland.

[Ke97] Kephart, J.O., Sorkin, G.B., Swimmer, M., White, S.R. (1997): Blueprint for a Computer Immune System, Proceedings of the Virus Bulletin (VB) Conference 1997, San Francisco, California.

[KM07] Kolter J.Z. and Maloof M.A., Learning to Detect and Classify Malicious Executables in the Wild. Journal of Machine Learning Research, 2007.

[LM06] Tony Lee and Jigar J. Mody, Behavioural Classification. Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference.

[NI05] National Infrastructure Security Co-ordingation Centre (NISCC). Botnets - the threat to the Critical National Infrastructure, NISCC Monthly Bulletin, London, October 17 2005.

[Ov04] Overton, M. (2004): Canning more than spam with Bayesian Filtering, Proceedings of the Virus Bulletin (VB) Conference 2004, Chicago, USA.

[Sa69] Sammon, J.W. (1969): A nonlinear mapping for data structure analysis. IEEE Transactions on Computers, C-18:401-409, 1969.

[We07] Wehner, S. (2007): Analyzing worms and network traffic using compression, Journal of Computer Security, 15(3), 2007.

[Wi05] Williamson, M. (2005): Using behaviour to detect and classify information-stealing malware, Proceedings of the Virus Bulletin (VB) Conference 2005, Dublin, Ireland.