Chatter: Exploring Classification of Malware based on the Order of Events*

Aziz Mohaisen, Andrew G. West and Allison Mankin
Verisign Labs, VA, USA

Omar Alrawi
Qatar Foundation, Doha, Qatar

ABSTRACT

Using runtime execution artifacts to identify malware and its families is an established technique in the security domain. Several papers in the literature relied on explicit features derived from network, file system, or registry interaction. While effective, the collection and analysis of these fine-granularity data points makes the technique quite computationally expensive. Moreover, the signatures and heuristics this analysis produces are often easily circumvented by subsequent malware authors. To this end, we propose CHATTER, a system that is concerned only with the *order* in which high-level system events take place. Individual events are mapped onto an alphabet and execution traces are captured via terse concatenations of those letters. Then, leveraging an analyst labeled corpus of malware, n-gram document classification techniques are applied to produce a classifier predicting malware family. This paper describes that technique and its proof-of-concept evaluation. This proof-of-concept concentrates only on network ordering and three malware families are highlighted. We show the technique achieves roughly 80% accuracy in isolation and makes non-trivial performance improvements when integrated with a baseline classifier of non-ordered features (of roughly 95%).

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General – *Security and Protection*; C.4 [**Performance of Systems**]: Measurement studies

Keywords

Malware, Classification, Sandboxing, n-gram.

1. INTRODUCTION

Malware analysis and classification is an important problem, with applications of wide variety that impact our daily use of computing and "the way we trust machines and codes" [26]. The rapid growth of malware and their attacks does not excluded any types of users: home users, enterprises, and governments [18]. Although malware detection has a more direct impact on home users, malware classification and identification by a family name is equally important for many reasons although in certain contexts and settings that are more operational and industrial [33]. For example, identifying a malware family enables orchestrating mitigation, assessing damages after detection, and reinforcing disinfection mechanisms. Additionally, understudying groups and classes of malware would enable malware researchers to concentrate their efforts on specific sets of families to understand their intrinsic characteristics, and to develop better detection and mitigation tools for them [30].

Techniques for characterizing malware samples fall under two schools of thoughts: signature-based [11,51] and behavior-based [7, 28, 39]. In the signature-based techniques for malware detection and identification, certain patterns in the contents are searched to determine whether a binary is benign or malicious, and if it is malicious, to tell to which group or class of malicious codes it belongs. Signature-based techniques rely on known set of patterns obtained by reverse engineering and manually inspecting malware samples. Accordingly, those techniques require many (specialist) staff-hours of analysis, although computationally cheap by only considering the static binary. On the down side, they are easy to circumvent by malware that uses polymorphic obfuscation, packing, and code rearranging. On the other hand, behavior-based techniques use runtime execution artifacts to extract features for identifying malware samples. While they are expensive, because they require execution of the malware sample to generate behavior artifacts, they are more effective than the statistic signature-based techniques: they are agnostic to the underlying code and can easily bypass code obfuscation, packing, and polymorphism [28].

Without being able to identify the broader classes of malware and their features, the capability of classifying malware would be impossible to implement and to enforce in operational settings. Furthermore, manually vetting malware samples against those features to identify their classes—while certainly possible for small datasets—does not scale for the real-world of malware populations [30]. To that end, machine learning techniques have been an enabler for the automation of malware classification [44]. When a piece of malware infects a system, the behavioral artifacts generated by the malware contain a wealth of features that can be used to "footprint" that malware with the help of machine learning algorithms. Those footprints characterize trails of behavior associated with the malware and its use of memory, file system, networking, and registry, among others. The outcomes and accuracy of machine learning algorithms utilized in the literature rely heavily on three factors: ground truth, algorithms, and features [24]. While algorithms are a venue for constant development and improvements in the machine learning community, the way they are used for classifying malware samples is a straightforward implementation of such results. Many algorithms are well-understood for their pros and cons. On the other hand, both ground truth and features selection are specific to the problem at hand [28].

Finding out a solid ground truth is a hard problem. Relying on blackbox-like sources—including antivirus scans—for obtaining a ground truth for detection and labels of malware samples and using them in machine learning techniques for malware classification is shown to be insufficient [6, 30]. With the ever-increasing complexity of malware, the limitations of signature-based techniques utilized by major antivirus vendors, and the inconsistency in using the signatures across multiple vendors makes them an unreliable source for accurate ground truth. To the best of our knowledge, no system or study in the literature relied on labels other than the ones provided by antivirus vendors for ground truth, despite the common belief that those labels are unreliable.

*An poster on this work appeared in proceeding of IEEE CNS [31]

On the other hand, determining the correct abstraction level of the artifacts to derive the features is an equally important part in the classification process [24, 50]. Features that accurately represent the malware family are an important and defining criterion for high-fidelity malware classification. Furthermore, the level of complexity associated with engineering and deploying systems for obtaining those features in operational settings determines the potential of adopting and accepting such systems and classification algorithms at scale. For example, while in some environments one can force running a malware sample in an arbitrary environment to obtain sufficient and representative behavior artifacts and features and use them in the machine learning algorithms, this privilege is not at no cost in online detection systems [41]. This is, when a piece of malware runs on a host, collecting those deep features becomes invasive to users using those hosts. To this end, while obtaining the correct type of artifacts and features, with the correct level of abstraction is important, it is always preferred that the system used for obtaining such features is less invasive to host operations.

In this paper we address both issues by introducing CHATTER, a "less-invasive" behavior-based system for collecting run-time artifacts and features of malware samples. CHATTER relies on the order in which malware samples generate behavioral artifacts, and use the frequency of the ordered artifacts as features. CHATTER can be built as a stand-alone system to derive those features and to classify malware samples, or as part of other virtualized execution-based systems. For its ground truth, CHATTER relies on highly accurate labels for manually-vetted malware samples. The vetting process requires many man-hours by experts and engineers, and we do that as a natural step in our system. We rely on those manually vetted labels to establish a baseline for operating and boasting CHATTER, and future learning and operation in CHATTER is less expensive by not requiring constant feed of those labels. To this end, the contribution of this paper is as follows:

- We introduce CHATTER, a system for malware analysis and classification based on cheap-to-obtain order-based behavior artifacts and features. The operation of CHATTER is less invasive than existing systems, and it is capable of creating accurate representation of malware families of different types by relying on cheap features. We argue for various scenarios of deployment of CHATTER, addressing operational needs.

- We demonstrate the operation of CHATTER with three malware families. For the evaluation, we rely manually-vetted malware. We demonstrate that CHATTER is capable of identifying malware samples by their groups and classes in a binary-classification context with reasonable operational accuracy, even when limiting the number of features used in its operational to a smaller set from that used in the literature.

The use of n-grams in the context of malware classification and analysis is an old story, as is the behavioral analysis as a tool. However, *a major non-trivial and novel component in our work is that it uses n-gram techniques to encode the order of subsequences of network communication events*. Our hypothesis is that each malware type has a unique communication pattern characterized by a certain order of events[1]. Building on this assumption we attempt to classify malware using only the order of their network communication, and establish the connection between the behavior-based profiling and the n-gram analysis as a tool for characterizing the order of events[2]. This study looks at the behavior of the network traffic and abstracts the features making the semantics of the underlying network communication irrelevant to the classification process. For our study we looked at three malware families that belong to different malware types; DDoS, Trojan, and targeted.

The organization of this paper is as follows. In section 2 we review the design of CHATTER. In section 3 we evaluate CHATTER on three real-world datasets of different malware families. In section 4 we discuss several aspects of CHATTER and its operation. In section 5 we review the related work. In section 6, we draw several concluding remarks.

2. SYSTEM AND DESIGN

Like most related work in the literature using behavior artifacts for characterizing malware, CHATTER relies on the execution of malware samples. For its intended operation, CHATTER does not require the execution of malware samples in a virtualized environment, and can perhaps be used for online characterization of malicious activities.

2.1 Design Goals and Requirements

We start our design by outlining goals that CHATTER tries to achieve. These goals are intended as a guideline. In section 4.1 we show how these requirements are met in our system.

- *Cost-effective:* a nice property that a system used for malware characterization should achieve is a balanced and low cost associated with the extraction of the features. For that, we emphasize in CHATTER that we exclude solutions that would capture the most of the features by requiring deep analysis of large number of artifacts generated by the malware sample as it is the case in the prior literature (AMAL [28] is an example—among many other systems in the literature [6, 7], where the whole image of 10GB hard drive and 256MB of RAM are analyzed to extract features.)
- *Less-invasive:* while it might possible to collect behavioral artifacts about the execution of malware samples in instrumented and virtualized environment, it is always desirable to be able to collect such artifacts about malware in their natural habitat while running on their hosting operating system. A system that can be deployed externally to observe the behavior and anomaly of a malware is ideal. We keep in mind that while this is a design objective that the system has to meet in principle, validating the system by identifying malware samples does not need to meet this feature. In fact, we use an off-the-shelf sandboxed system under full control for collecting malware artifacts as a proof-of-concept.
- *Generalizable and multi-purpose:* while the end goal of CHATTER is to characterize malware samples by their behavior, our end goal is to have a system that is generalizable. This is, we intend to build a system that can characterize any malicious activity or entity based on the behavior, but not limited to the malware. In section 4.3 we show two of such applications that benefit from such generalization.
- *Evolvable to address behavior changes:* given that many malware families evolve over time to circumvent behavior-based techniques for malware detection and classification, one ideal goal of our system is to resist this evolution by providing an evolvable techniques to address changes in malware behavior. Our system should rely on the best set of features representative to the malware samples of interest.
- *Accurate:* the system should make use of a solid ground-truth and match this ground truth by generating as small as possible of false alarms. While it might be impossible to match the

[1]This hypothesis is backed by the prior work of Forrest et al. [13], which gives "a sense of self for Unix processes" by showing that the order of system calls can be used to further tell whether a process is benign or malicious. See the related work in §5 for differences

[2]Our choice of network features is not arbitrary: as discussed in the rest of the paper, network features are cheaper than file system features to obtain. For example, they can be captured without the need of residing on the same host as the malware being executed.

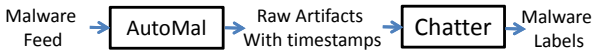


Figure 1: Flow diagram of CHATTER and its use of AUTOMAL and its behavioral artifacts

ground truth 100% of the time, our guideline for the acceptable accuracy is operational: oftentimes, it is only required to weed out the majority of irrelevant malware samples, and a small percent of false alarms might be tolerated.

While these requirements are idealized, and achieving them all at once might not be easily possible, we try to achieve them to a great degree in CHATTER. We show how they are achieved as we walk through the design and in its evaluation.

2.2 System Workflow

We begin by summarizing the CHATTER workflow and its interaction with a sandboxed execution environment capable of producing a list of events that take place during the malware execution, as visualized in Fig. 1.

2.2.1 *Sandboxed Execution and Artifacts Collection*

While any sandboxed execution environment (or bare metal execution) can be used for extracting the features utilized in CHATTER, we use an operational system named AUTOMAL [28] for that purpose. AUTOMAL is a windows-based sandboxed execution system capable of collecting low-granularity artifacts to represent the way malware samples interact with memory, file system, registry, and the network.

AUTOMAL enables researchers and customers to feed into it binaries that are likely to be malware samples. For details, refer to [28] on the design and operation of AUTOMAL. However, in short, AUTOMAL consists of 4 unites: malware samples submitter, controller, workers, and back-end storage. The malware sample submitter enables users to submit binary codes to the systems, which are then queued until some resources in the system are available for analyzing the provided malware sample. The controller makes sure that resources are fully utilized by fetching tasks from the samples' queues, initiating virtual machines (VMs) as workers, loading the proper configurations and settings, and running the malware sample. The worker runs the malware sample and collects run-time artifacts about the malware sample. As of its current design, AUTOMAL enables the collection and profiling of memory, network, file system, and registry artifacts and features. Finally, once the malware sample is executed in the virtual environment, the collected artifacts are passed to the controller, which logs them in the backend storage unit.

While all of those artifacts can be used in CHATTER to achieve its end goal, some of the features are more expensive than others, and obtaining them in a slightly different context than virtualized or tightly controlled environment would be almost impossible without interfering with the normal operations on hosts (further details are in section 4). For that, we focus on network-related features generated by AUTOMAL and use them for the operation of CHATTER in identifying and classifying malware samples. Notice that any other system that is capable of producing the limited number of features used in our system can be utilized. To this end, CHATTER is treated as a blackbox that takes the raw-artifacts with timestamps for malware samples from AUTOMAL and generates labels for those malware samples based on a highly-accurate set of manually inspected training labels. The detailed diagram of the internals of CHATTER is shown in Fig. 2. In the following, we walk through the steps of its operation.

2.2.2 *Behavioral Documents and Their Extraction*

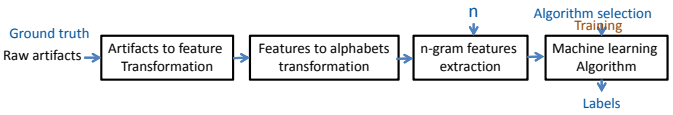


Figure 2: Flow diagram of CHATTER

CHATTER takes the raw artifacts generated by AUTOMAL and highly-accurate labels of malware samples associated with those artifacts as an input. The raw artifacts generated by AUTOMAL are named a *behavioral profile* of the malware sample. The behavioral profile maintains the notion of order to it: a profile would list all events and execution actions as viewed at the behavioral level with their respective order. CHATTER then transforms the artifacts into features. For example, a file system artifact representing the event of writing a file on a certain path is transformed into the features of *file_written_on_a_path*. A DNS query of type MX on port 53 would generate features like *dns_query*, *port_53*, and *mx_query*, all of which can be combined in the order they happen to represent the use of network resources by the malware. Once all artifacts are transformed into features, those features are transformed again into unique alphabets. In this phase, CHATTER computes the unique set of features regardless to their order (details in section 3.2) and assigns a character from a pre-determined list of alphabets to each of those unique features. After generating the unique characters, CHATTER transforms the behavioral profile of the malware sample being analyzed into a new profile (document) in terms of the characters and alphabets it used as a representation. Abstracting the behavioral profile into a document according to the way described above simplifies applying a wide range of literature on "text mining" for observing interesting and relevant patterns for identifying such document and classifying it into a broader class.

2.2.3 *The n-gram Features Extraction*

As mentioned earlier, CHATTER relies on the order in which artifacts are generated to identify a malware sample by its broad family. To this end, CHATTER utilizes n-gram features obtained from the behavioral document. Given a value for the parameter n and the behavioral document, CHATTER computes all n-grams in the document. CHATTER then uses the count of the n-grams it generates as the seed for the feature vector it will use in the machine learning algorithm for identifying the malware sample. After computing the n-grams and their counts for all malware samples, CHATTER takes the union of the unique features of each malware sample, where features are computed as the counts of the individual n-grams, and fill in the counts of the individual malware samples. This step is done to unify the feature vector used for representing the malware samples. Notice that, in theory, one can avoid this step by considering the features vector using all combinations of the alphabets used in representing the behavioral document with the given length n. However, this approach would have two disadvantages: 1) it will result in a sparse representation that is, unlike in text-related n-representation, would be under utilized, and 2) it will explode the number of features greatly to the extent that one cannot easily process them in a reasonable time with the algorithms we use.

2.2.4 *The Machine Learning Component*

Once the n-gram features are computed for the different malware samples, they are fed into the machine learning algorithm of choice to label them. Using a training set, the machine learning model is built, and then using a testing set it is validated for its accuracy, and tuned against the feature it uses. Then, once the model is created, malware samples are passed to the system to identify their family. Further details on the use scenarios of CHATTER are in section 4.

3. EVALUATION

Table 1: Malware families used in the evaluation of CHATTER, including their size and the average number of events per execution trace (further details are in §3.1).

Family	Quantity	Characters Avg.
Zeus	1025	50.74
Darkness	544	61.47
Shady RAT	1130	52.74

In this section we walk through the evaluation of CHATTER, with details on the dataset, and the method used for establishing a baseline for ground truth, the set of features we used for evaluating the different malware families, the machine learning algorithms, evaluation metrics, and findings and results.

3.1 Datasets and Ground Truth

For the evaluation of CHATTER, we use three malware families: Zeus, Darkness, and Shady RAT (SRAT). The combination of all three malware families covers a wide range of network behavior and provides good evaluation points to conduct this study to determine the effectiveness of event order on the network for classification purposes[3]. Each malware sample in these families is obtained from an operational product, where sources of the malware samples include customers (banks, energy companies, etc), partnering antivirus vendors, and researchers private research on popular malware sample. Once the malware samples are fed into the system, AUTOMAL, they are executed for a certain amount of time to generate behavioral artifacts that are then used in CHATTER as described in section 2.2. We provide details on these families and their contexts in the following section[4], and show their population and the average length of their behavior document in characters in Table 1.

3.1.1 Datasets

Zeus: Zeus [29] is the first malware family that we use in evaluating CHATTER. Zeus is a famous banking Trojan that is used by cyber criminals to run a botnet to steal money, credentials, and system resources from the infected victims and their machines. When the Zeus malware infects a system several artifacts are created like file system, registry, network, and memory. Upon infection, the malware communicates with the command and control (C&C) server by sending information pertaining to the infected host and requesting a configuration file to instruct the Trojan for further actions. Zeus also periodically sends out stolen data to "drop site" that is noted in the configuration file sent from the C&C server. These network artifacts can help networks administrators identify possibly Zeus infected host in their environment. These network artifacts can also be used to classify Zeus network traffic thus labeling malicious code by their network artifacts. Accordingly, Zeus is one potential good example to demonstrate CHATTER. For the evaluation of CHATTER, we use 1025 Zeus malware samples. To understand the capabilities of CHATTER, we need another *class* of samples that are not Zeus. For that, we randomly select a sample set of the same size (1025 samples) from a large repository of malware. The samples of this set consist of multiple families.

Darkness: The second malware family we use for evaluating CHATTER is the distributed denial of service (DDoS) bot known by the names *Darkness* and *Optima* (we will refer to it as Darkness throughout the rest of the paper). Darkness infects machines for the sole purpose of using the resource to carry out DDoS attacks. The bot infects the system by installing itself as a service and begins to communicate with the C&C server to receive commands. Darkness is capable of carrying out hypertext transfer protocol (HTTP) flood, Internet control message protocol (ICMP) flood, and transmission control protocol (TCP) or user datagram protocol (UDP) flood attacks. The HTTP flood attack uses different user-agent for each HTTP request making it hard to identify DDoS traffic. The network artifacts generated by the Darkness are noticeable on a network due to their high volume, which makes it a good feature to use for classifying the DDoS bot using machine learning algorithms.

Shady RAT: (SRAT) The final malware family considered in evaluating CHATTER is a target malware that McAfee reported on by the name Shady RAT [3] (we will refer to it as SRAT throughout the rest of the paper). The malware targeted several high profile organization and government entities to steal intellectual property and sensitive data. The malware employs a covert communication channel that makes it hard to detect on the network. SRAT comes in usually as spear phishing email that social engineers the victims into running the malware on their system. The malware infects the system and starts its communication with the C&C server. The communication is done by the malware downloading an HTML page and parsing out HTML comments which contain encrypted commands from the C&C server. Another method the malware would fetch a page and use steganography to decrypt commands from images embedded in a web page. This communication channel is very hard to detect by a network administrator because it would blend in with regular traffic flowing through the network.

3.1.2 Establishing A Ground Truth

Labeling malware to establish a ground truth is perhaps the most important step needed for obtaining reliable and meaningful results in any supervised classification task. The prior literature relies on certain methods for labeling malware samples, but mostly using labels and names provided by anti-virus scanners. However, we noticed by experimenting with several malware families that antivirus scanners are not a reliable source for labeling. This observation is confirmed in several recent works [6, 7, 41][5]. The limitations of AV-scanners in the aspect are understandable, given that the priority and chief goal of those scanners is to give an accurate detection, but not a label for what is detected through them. Furthermore, the wide variety of techniques deployed for labeling makes a consistent label among multiple vendors almost impossible.

To this end, we use a different approach to labeling malware samples to assist the reliability of the evaluation of this study. The malware samples used in this study are collected over a long period of time, and that enabled analysts in our organization to manually identify and label them. This process can be time-consuming: at average a previously unseen malware sample can take more than 10 hours to manually characterize and give the proper label and name by an expert. However, often time this effort comes natural in our organization and does not require dedication of such valuable man-hours for this research initiative only: often time we are contracted by our customers to analyze and report back to them on a given set of rogue binaries. Those binaries accumulated over time are used as our dataset with fine labels created for them by our analysts.

Beside customers who are interested in analyzing and understand the behavior of binaries used in their enterprises, we have two additional sources of malware samples: 1) samples provided to us by partnering antivirus vendors interested in intelligence sharing,

[3]Our system and technique are usable for a wide variety of malware families, and the three families used in this study are only for the demonstration of the systems' operation based on variety of behaviors. We ran our system on 13 other families, including Ramnit, Bredolab, Zero Access, SillyFDC, and Virut, among others, and confirmed the systems' operation and accuracy on them—in all of the aforementioned families, an accuracy of over 90% is achieved.

[4]To foster transparency and reproducibility of results, we intend to release the dataset used in this work to the larger community.

[5]Except in [30], these studies do not particularly go in depth to analyze the limitations of the AV-based labeling but agree with us in total in this conclusion based on undisclosed experiments—yet, admitting it's a challenging problem with no other way to obtain labels, The works in [7, 41] use AV-labels as a ground truth.

Table 2: Features used in composing the behavioral documents of malware samples studied in CHATTER. Each class of features consist of multiple features, and each is represented as an alphabet in the behavioral document.

Feature class	Features listing
IP and port	unique dest IP, certain ports
Connections	TCP, UDP, RAW
Request type	POST, GET, HEAD
Response type	response codes (200s through 500s)
Size	request (quartiles), reply (quartiles)
DNS	MX, NS, A records, PTR, SOA, CNAME

and 2) malware samples collected by our analysts through their own investigations and explorations. The latter samples are usually analyzed by analysts to understand trends and patterns in the malware eco-system. For the first feed provided by antivirus vendors, malware samples are delivered to us from partner antivirus vendors without signatures. For that, and to weed out irrelevant malware samples using Yara signatures [1] created by our analysts based on their knowledge of the malware family of interest (more details on those signatures are provided in [28]). Notice that Yara signatures are only used for pre-processing, and are not used for giving a label to to a malware sample—they are used rather as a secondary channel for labeling. After that, we feed those malware samples to an automated malware analysis system, called AMAL [28], which uses behavioral patterns, and analyst-fed labels associated with those patterns, to extrapolate labels to previously unseen samples. The same technique is utilized for analyst-collected samples, although the latter ones heavily depend on manual inspection by those analysts in giving labels, same as with customers-related samples[6].

While one may see this way of obtaining labels as a limiting factor in the operation of CHATTER and other systems that use analyst-vetted labels, we argue otherwise in section 4.2. We capitalize on the continuous need of manual inspection in certain businesses, thus providing a by-product for the operation of CHATTER.

3.2 Features and Feature Selection

Running a malware sample in a sandboxed environment results in a lot of artifacts, however not all of them are relevant nor meaningful in identifying a malware sample. Accordingly, we limit our attention to only representative and meaningful artifacts that may result in representative features. For that, we rely on the *domain knowledge* of an expert of the studied malware samples and their families, and encode that knowledge in the artifacts selected for characterizing the malware samples of the various families.

To this end, we select 26 features all of which are network-related (for the rationale we mentioned in section 1) as shown in Table 2. Those features are selected according to the description in section 2.2.2, and are not to be confused with the n-gram features. Furthermore, 26 features are generated as a representation for the 3 malware families collectively. However, we notice that some of the features do not exist for a given malware family, as shown in Table 3. As we generate behavioral documents of each malware sample upon its execution, we obtain a set of characters transformed according to the method described in section 2.2.2 with the average length of document (in characters) as shown in Table 1. Following the procedure in section 2.2.3, we obtain the unique set of n-grams (in a condensed representation) for each malware sample. We try that for various values of the parameter n (1 to 8 are shown in Table 3 and used in the rest of the experiments later on).

[6]We note that the majority of samples used in this study come from customers, where man-hours are used for labeling samples, while the two other sources represent a minority of the samples.

Table 3: The number of unique n-grams actually observed in each of the studied families.

n value	1	2	3	4	5	6	7	8
Zeus	24	102	250	481	943	1690	2638	3794
Darkness	24	103	243	461	875	1503	2266	3149
SRAT	25	105	247	460	877	1536	2337	3300

n-gram features selection. While features selection algorithms exist to reduce the number of features used in classifying malware samples with reasonable accuracy, we avoid using any of those algorithms for the following reasons. First, the goal of CHATTER is to enable the use of new features rather than testing which subset of them performs best. Second, reducing the number of features and trying off-the-shelf algorithms, like recursive features selection (RFS), principle component analysis (PCA), and others, is an orthogonal work that brings little merits to this work. However, for the completeness of this work we experimented with an off-the-shelf feature selection algorithm (namely, the RFS) and found that one can achieve the same accuracy as with the whole set of n-gram features but using a smaller set of them. This result, while relevant to some extent, is straightforward, and is in line with a large body of literature on the problem [7, 9, 28, 29, 50].

3.3 Evaluation Metrics and Procedures

To evaluate CHATTER, we use evaluation metrics widely used in the related literature [7, 28]: the accuracy, precision, recall, and F1 score. For a binary classification problem, in which it is required to determine if a given malware sample belong to the class of interest S, we define the following possibilities: 1) true positives (T_p) is the number of samples correctly identified by the machine learning algorithm to belong to the class S. 2) false positive (F_p) is the number of samples marked by the machine learning algorithm falsely to belong to S. 3) true negative (T_n) is the number of samples marked by the machine learning algorithm correctly not to belong to S. 4) false negative (F_n) is the number of samples marked by the machine learning algorithm falsely not to belong to S (but they are actually in S). Using the four outcomes and counts above, the precision, recall, accuracy, and F1 score are defined as follows:

$$\text{Precision} = \frac{T_p}{T_p + F_p}, \text{Recall} = \frac{T_p}{T_p + F_n},$$
$$\text{Accuracy} = \frac{T_p + T_n}{T_p + T_n + F_p + F_n},$$
$$\text{F1 score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

For all experiments we run to evaluate CHATTER, we use the k-fold-cross-validation method with $k = 10$. In this method, the input dataset is divided into k-folds, where $k-1$ folds are used for training the machine learning algorithm and the remaining fold is used for testing. The process is repeated k-times by changing the testing dataset among among the k possible folds. At the end, the result—in terms of the true and false (positive and negative)—is computed as the average over the k runs. We set k to 10 because this is the widely used setting in related contexts.

3.4 Machine Learning Algorithms

In the evaluation of CHATTER, we use three machine learning algorithms: the k-nearest neighbor (k-NN), support vector machines (SVM), and decision tree classifier. All of the three algorithms are intended for binary supervised learning, and are capable of identifying the membership of a malware sample into one of two classes. In the following, we formally and briefly review those algorithms

Support Vector Machines (SVM): Given a training set of labeled pairs $(\mathbf{x}_i, y_i)$ for $0 < i \leq \ell$, $\mathbf{x}_i \in R^n$, and $y_i \in \{1, -1\}$, the

(L2-regularized primal) SVM solves:

$$\min_{\mathbf{w},b,\xi} \ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{\ell}\xi_i$$
$$\text{subject to} \ \ y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0$$

where the training vectors $\mathbf{x}_i$ are mapped into a higher dimensional space using the function ϕ, and the SVM finds a linear separating hyperplane with the maximal margin in this space. $C > 0$ is the penalty parameter of the error term (set to 0.01 in our work). $\xi(\mathbf{w}, \mathbf{x}, y_i)$ is called the *loss* function, where we use the L2-loss defined as

$$\xi(\mathbf{w}, \mathbf{x}, y_i) = \max(1 - y_i\mathbf{w}^T\mathbf{x}_i, 0)^2.$$

Decision Trees Classifier: We utilize a single split tree for two classes classification using all of the features provided by CHATTER. For the target class label $Y = y_1, \ldots, y_n$ and a set of feature vectors $\mathbf{x}_1 \ldots, \mathbf{x}_n$, at each internal node of the tree—and for the training set—we apply a test to one of the inputs, namely $\mathbf{x}_i$, determining to go either left or right in the tree branches based on the outcome of the test. When running over all of the training feature vectors, we mark the leaf nodes as the aggregate (mean) of all the training samples (to one of the class labels in Y.) For testing, we do the same and assign the label of the leaf to that of the sample feature vector used to reach to the leaf. We omit further details and refer the reader to [2] for an extended description. Notice that other variations of the decision trees are shown in the literature to provide better results, including random forests. We did not try any of those techniques, since the technique we utilized already provided reasonable results. We leave trying such techniques as a future work.

The k-Nearest-Neighbor The k-NN is a non-linear classification algorithm. In the training phase, we provide the algorithm of two labels and a set of training samples. In the testing phase, for each sample vector $\mathbf{a}$, we give it the label of the most frequent among the training samples nearest to it. For the lack of space, we refer the reader to a textbook explanation of the technique in [2].

3.5 Results

Computing n-grams: For a selected n it is straightforward to produce trace substrings in a sliding window fashion. Table 3 shows the number of unique n-grams actually observed. While there are 26^6 possible 6-$grams$, only 1690 (0.0005%) are actually observed for the Zeus malware family. A bag-of-words representation is extremely sparse and this hints at the underlying relationship/dependency between certain network actions.

Results in isolation: Table 4 shows performance for selected values (n=1,4,8) across all families and algorithms. To make sweeping generalizations across all families, Decision Tree classifiers tended to perform best with an average accuracy of ≈80%. Although the transition from $n = 1$ to $n = 4$ tended to produce noticeable performance increases, the performance ramifications of the next increase was more mixed. Between families we observe that "Darkness" malware performed most poorly overall, while "SRAT" performance fluctuated wildly based on the algorithm applied.

Atop a baseline classifier: While it is clear that CHATTER is able to independently predict malware family with reasonable accuracy based on the order in which network artifacts happen, it is also desirable to see if it can contribute when paired with a baseline classifier. Fig. 3 shows this result, where the baseline classifier consists primarily of filesystem features. Compared to results presented in Table 4 and compared to the results and findings in [29] performed on the same dataset, we observe that ordered features are capturing independent portions of the problem space and making non-trivial improvements to overall precision, recall, accuracy and F1 score.

4. DISCUSSION

In this section we highlight interesting findings and observations from our study of CHATTER. In our study we limited the value for n to 8 and used only the features that had a vector value to classify the malware families. For example, if using all possible feature combination for n=8 the number of features would be 26^8 features. Instead, we only used ≈ 3800 features for the largest n value. These numbers make the classification much more feasible by only considering feature vectors with values. To this end, we note that the malware families with more features had a better accuracy score around the value of n=4 or 5 than for different values of n.

For example, Zeus Banking Trojan had on average more features than the other families, as n increased, and it performed better in the classification. We do see a performance drop-off as n grows beyond the value 5. The optimal number for n-gram seems to lie between the values 4 and 5. This observation is also validated with the classification results from combining file system and ordered network features. The accuracy graphs in figure 3, 4, and 5 show downward trend as n takes on values larger than 5. We conclude that order does improve classification by about 10% to 12%, see table 4, with an optimal value of 4 or 5 for n.

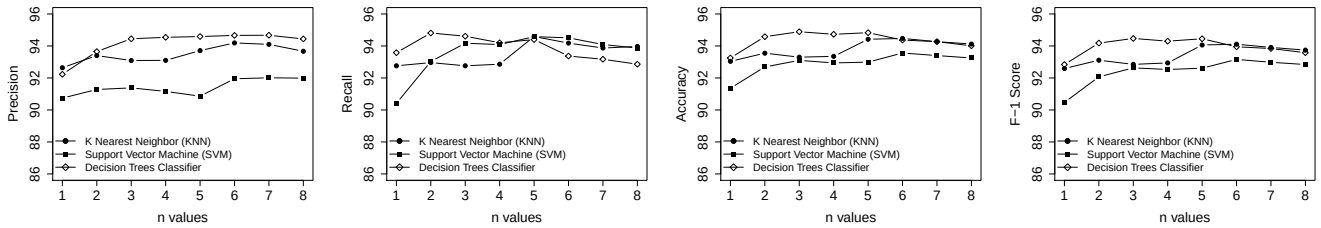
We note that the classification for the DDoS malware family was less accurate than the targeted malware or Zeus even when combining file system features; see figures 3, 4, and 5. This can be explained by the way Darkness infects a host machine and uses it as a bot. In the case of the targeted malware and Zeus, our file system feature selection captures their artifacts pretty accurately by considering several predefined installation locations like APPDATA directory and quartile of file sized generated during infection. The Darkness malware does not create any files instead, it moves itself to system directory and creates a service, which are several registry key value pair, to persist and infect the system.

This technique is not captured by the file system artifacts except for the initial infection file. Adding the file system feature to the ordered network feature did not improve the results by much as noted in the graphs above. The file system artifacts and registry artifacts are more expensive to capture since they require an capturing mechanism on the virtual machine. Whereas network artifacts can be captured externally by simply monitoring the network interface. This trade-off gives us the option for cheaper classification with a degradation in accuracy, about 5% to 7%, or more accurate but expensive classification.

In the following, we extend our discussion to consider how the CHATTER meets our design requirements, outline some limitations of the technique, and some potential application that one can build using the same idea of CHATTER.

4.1 Meeting Design Requirements

In the following we highlight how CHATTER meets the design requirements outlined in section 2.1. First of all, our system is *cost-effective* when deployed for characterization of malware samples. This effectiveness comes from two directions: its use of a single class of artifacts and features, and abstraction of features into order rather than raw features. It has been shown in the literature that a system used for characterizing a malware sample based on the network artifacts generated by the malware sample can run an order of magnitude faster than a system that looks at a large spectrum of features. For example, AMAL [28], which is similar in its operation to a large body of work in literature, operates ideally on 128 virtual machines, and is capable of processing 23,000 malware samples daily. CHATTER, on the other hand, is capable of processing 370,000 malware samples per day using the same infrastructure. This number of malware samples is way more than what we receive and analyze daily. Indeed, the number is even more than the 250,000 samples a popular antivirus provider like Sophos detects



(a) Precision (b) Recall (c) Accuracy (d) F1 Score

Figure 3: Performance measures for the Zeus malware with network artifact classification using CHATTER.

Table 4: Precision, recall, accuracy and F1-score for selected n-gram values.

	n-grams	1				4				8			
	Algorithms	P	R	A	F1	P	R	A	F1	P	R	A	F1
Zeus	k-NN	80.79	79.68	81.48	79.97	79.07	83.90	82.25	81.35	78.29	78.17	79.64	78.09
	SVM	67.41	82.67	72.69	73.92	75.96	80.47	78.67	77.84	80.41	82.87	82.45	81.50
	Decision Trees	80.14	80.90	81.74	80.42	81.13	81.82	82.67	81.35	80.82	82.82	83.02	81.77
Dark.	k-NN	76.22	73.13	76.08	74.56	80.40	71.52	77.70	75.57	71.38	69.58	71.65	70.20
	SVM	76.82	32.38	62.24	45.05	78.18	71.32	76.45	74.35	76.62	76.36	77.22	76.27
	Decision Trees	80.45	72.56	78.20	76.07	81.75	72.89	79.04	76.93	80.50	68.37	76.39	73.59
SRAT	k-NN	81.38	76.78	82.78	78.45	83.87	81.83	85.51	81.95	83.99	74.28	82.93	78.16
	SVM	76.88	65.43	75.88	69.55	83.70	82.94	86.23	83.03	85.68	80.86	86.33	82.71
	Decision Trees	85.16	81.11	86.44	82.60	88.28	81.65	88.01	84.45	86.13	78.92	85.54	81.85

and analyzes daily which is way more than our daily[7].

The same property CHATTER has of relying on network features only makes it also less-invasive: those features can be observed on the network without having to reside on the host on which the malware is executed. However, we notice that the host has to run no processes beside the malware in order for CHATTER to collect relevant features. This latter condition can be fulfilled by running the malware and the host in a limited (monitored) mode of operation. This mode of operation can be further triggered in CHATTER by observing some suspected hosts for harboring malware and this approach allows that host to be studied directly.

As we discuss in the following section (§4.2), our system requires maintenance to address evolution in behavior of malware samples. However, this maintenance is made easy given the operational context of CHATTER: researchers keep feeding the system with new ground truth labels over its operation life-time. Accordingly, any change in behavior will be capture in this ground truth, and the system can be easily retrained to capture such changes.

Finally, while the accuracy provided by CHATTER when running in isolation is less than that provided by other systems; e.g., AMAL [28], the results are operationally acceptable: oftentimes it is required to only weed out likely irrelevant malware samples in large repositories. Other false alarms can be further captured and addressed using more expensive techniques (e.g., by combining other features as we have shown in section 3.5). We emphasize that while many academic studies on malware classification and analysis strive to provide a 99.9% accuracy, our goal in this work is to provide an operationally acceptable accuracy by trading complexity and cost of operating the classification system.

For our classifiers we used k-folds to test the accuracy of our system, where k was set to 10. The accuracy we achieved more than 80% for all cases, which in an operational environment is acceptable. Although academic studies strive for 99% accuracy, in operational environment this is not possible, even for 90% accuracy. Thwarting the system's accuracy using obfuscation and re-ordering is not possible because we are looking at the network behavior artifacts.

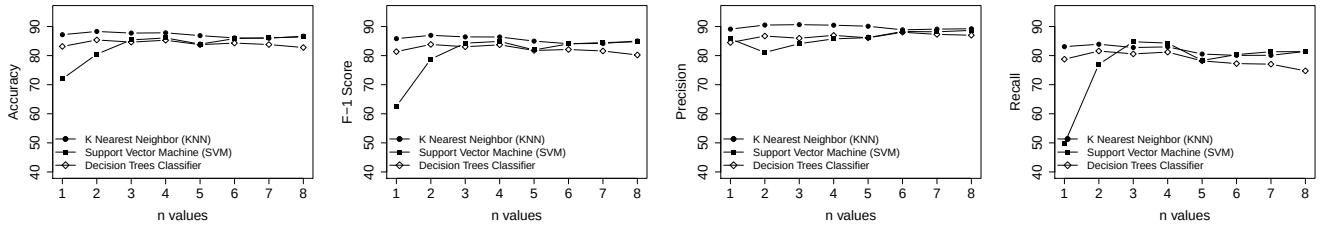
[7] http://bit.ly/17UVQ19

4.2 Limitations

There are several limitations of CHATTER that would impact its performance, which we address in this section.

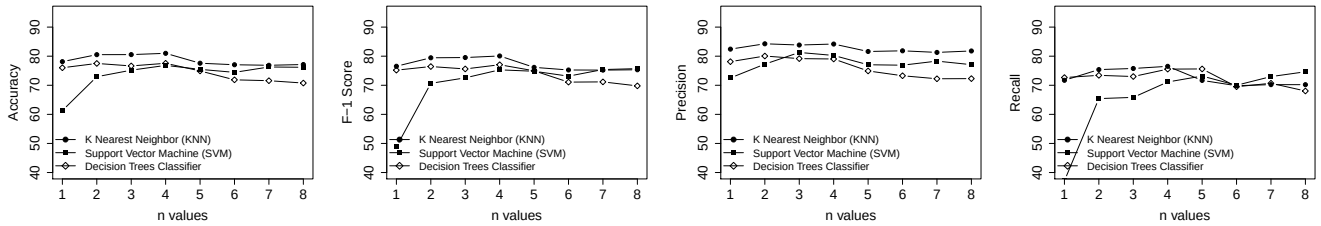
Noised features: Like most behavior-based systems for malware classification CHATTER performs best when malware samples do not produce extra information to disguise their behavior and fool the used machine learning algorithm. Or even worse, often times malware samples evolve overtime, and a real-world system for malware characterization and classification needs to address this evolvability. However, unlike systems that make use of exact matching of behavior profiles, CHATTER provides some flexibly in how matching of malware samples are grouped together using the n-gram features. A potential scenario for fooling CHATTER is to generate a lot of irrelevant behavioral artifacts and plug them in the behavioral profile in the hope of hiding what is relevant and used for characterizing malware samples. To that end, CHATTER will generate potentially different features for malware samples that potentially belong to the same family. While the problem is generic and not limited to the operation of CHATTER, but rather any system that relies on behavioral patterns in the execution of malware, we address this issue in two ways:

- We emphasize that not all the features generated by a malware sample need to be used to operate the machine learning algorithm: a feature selection algorithm can be used to marginalize the impact of the noised features on the operation of CHATTER.
- We note that certain events in the operation of a malware sample that belongs to the malware family have to happen in the same partial order, regardless to the noise put in between them. Our future work to address this limitation is to derive features concerning those events as they happen in their partial order by filtering out the noise between them. While this might seem to requires deep understanding of the



(a) Accuracy (b) F1 Score (c) Precision (d) Recall

Figure 4: Performance measures for Targeted malware (SRAT) with network artifact classification using CHATTER.



(a) Accuracy (b) F1 Score (c) Precision (d) Recall

Figure 5: Measures of performance for Darkness DDoS malware network artifact classification.

studied malware families and their expected behavior, signal processing techniques might be utilized to perform the task in less expensive way. For example, in the future it is worth considering how statistical and information theoretical characteristics of the n-gram features can guide the process of deriving representative and meaningful features. Notice that employing this partial order of events would address the issue where a malware sample will craft a legitimate-looking network artifact to also add noise to the artifacts used for characterizing it.

Adaptive malware: Also, as in other behavior-based systems, adaptive malware samples produce various behavior profiles based on the environment they are run in and are an issue for the operation of CHATTER as well. Even worse, some malware samples would not act upon running in a sandboxed environment, which will be an issue if the sandboxing approach is to be used for deriving networking features in CHATTER. We address this issue in two ways: AUTOMAL, which is used as the sandboxing tool for CHATTER, generates patches to deceive malware samples by returning to them registry values indicating that they are running on the bare metal. Furthermore, for sophisticated malware samples that do not respond to those patches, we run malware samples on the bare metal (or in a hardware virtualized environment, which often proves to work). We notice that this problem, while affects the results of CHATTER, is not specific to the system but to the underlying tool used for deriving the features. One scenario of applying the work in reality is to use on-the-wire monitoring, which will not require sandboxed execution thus eliminating the problem.

Continuous training and cost of labeling: Because of the evolution of malware samples, continuos training is needed in our system to adapt to changes in artifacts generated by them and used for creating vectors of features. While this issue might seem as an inherit shortcoming for machine learning based techniques, it is addressed naturally in CHATTER: as mentioned earlier, many of the malware samples fed into CHATTER belong to customers and require reverse engineering, deep analysis, and manual inspection. To that end, this process provides a natural venue for obtaining features, labels, and training sets for CHATTER.

4.3 Other Applications

While the main application we used in CHATTER relies on transforming behavioral profiles into documents and using them for understanding the behavior of malware utilizing n-gram techniques, the concept is generic and can be applied to a wide variety of applications. In the following we identify several potential applications which can benefit from CHATTER:

- *Process-based DDoS detection:* while our system studies a specific DDoS malware family, our system cab be generalized to understand any process-based DDoS attack by observing traffic on the wire, generating sufficient artifacts that can be used to derive features and footprint or attribute such attack.
- *Advanced persistent threats:* often time, such threats (process-based) result in a lot of artifacts that are generated over a long period of time where the literature systems can be less effective in characterizing them. One potential application in characterizing them is to rely on the inter-event patterns they generate, using CHATTER.

5. RELATED WORK

There has been plenty of work in the recent literature on the use of machine learning algorithms for classifying malware samples [6,10,21,34,37–40,46,47]. These works are classified into two categories: signature based and behavior based techniques. Our work belongs to the second category of these works, where we used several behavior characteristics as features to classify the Zeus malware sample. Related to our work is the literature in [34,39,40,52]. In [34], the authors use behavior graphs matching to identify and classify families of malware samples, at high cost of graph operations and generation. In [39,40], the authors follow a similar line of thoughts for extracting features, and use SVM for classifying samples, but fall short in relying on a single algorithm and using AV-generated labels (despite their pitfalls).

The work of Bailey et al in [6] motivated to many of the related works on malware classification using behavior. Our work is different from their work in two aspects. First, although we share similarity with their high level grouping of features, our system relies on the order of events, which exposes richer behavior. Finally, we use highly-accurate analyst-vetted labels for evaluation, where they use heuristics over AV-returned labels.

While we don't particularly use memory signatures for the operation of CHATTER, a great potential can be seen in utilizing those features the same way CHATTER uses the network features. Related to that, Willems et al. introduced CWXDetector [48] which detects illegitimate code by analyzing memory sections that cause memory faults—artificially triggered by marking those section non-executable. The work can be integrated into our system, although at cost: the mechanism is intrusive to other running processes in the memory. Our current system, on the other hand, does not require any memory modifications. Kolbitsch et al. [22] introduce Inspector, which is used for automatically reverse engineering and highlighting codes responsible for "interesting" behaviors by malware. Related to that, Sharif et al. proposed to understand code-level behavior by reverse-engineering code emulators [43]. Those are examples among other works in the literature. However, all of those works do not generate malware artifacts other than memory-related signatures, which by themselves have limited insight into characterizing generic malware samples.

Related to our use of network features is the line of research on traffic analysis for malware and botnet detection, reported in [14–17, 20] and for the particular families of malware that use fast flux, which is reported in [19, 32]. Related to our use of the DNS features for malware analysis are the works in [4, 5, 9]. None of those studies are concerned by behavior-based analysis and classification of malware beyond the use of remotely collected network features for inferring malicious activities and intent. Thus, although they share similarity with our work in purpose, they are different from our work in the utilized techniques.

Broadly related to our work are systems for overcoming malware evasion techniques. Improving on malware detection, analysis and classification have been investigated as well in several works in the literature. In [25], K-Tracer is introduced for extracting kernel malware behavior and mitigating the circumvention of loggers deployed in the kernel by rootkits. In [35], MacBoost is used for prioritizing malware samples by determining benign (or less severe) from malicious piece of codes. A system to prevent drive-by-malware based on behavior, named BLADE, is introduced in [27]. A nicely written survey on such systems and tools is in [12].

Using n-grams for malware classification is not new. However, *the work in the literature has looked at extracting features from executables (e.g., sequence of bytes in the binary files [40]) or streams of communication traffic [49], but not* **sequence of events** *happening while executing a malware sample*. Examples of such works include [23, 35, 42, 49]. Of particular interest is the concurrent work in [49], which derives features of network contents based on the contents, rather than well-understood behavioral artifacts and events. Using network artifacts for identifying malicious activities, like botnets, is investigated in [15–17, 36, 45]. Further applications of characterizing malicious domain names using network traffic and artifacts (DNS queries, among others) are reported in [8, 9, 37]

The basic idea of using the order of events in characterizing processes is first explored by Forrest et al. in their seminal work in [13], where it is shown that a process-level intrusion can be detected using the order in which system calls happen as a sequence. However, the work differs from our work in three aspects 1) it is concerned with detection rather than classification, 2) it uses system calls rather than networks features, 3) it use a whole sequence as a single feature that is easy to manipulate and break, rather than subsequences (as in n-grams) and their frequency.

Finally, the use of machine learning techniques to automate classification of behavior of codes and traffic are heavily studied in the literature. The reader can refer to recent surveys in [44] and [41].

6. CONCLUSION

Motivating by the need for deriving new and easy-to-obtain features, we introduced CHATTER, a behavior-based malware classification system. CHATTER uses behavioral artifacts generated by malware samples at their runtime and characterizes their use of one or more group artifacts. At its core, CHATTER considers the order in which events in the behavior of a malware happen. We notice that order-based features can be captured using the n-gram technique widely used in information retrieval. With its many advantages advocated in section 2, and using three malware families of various characteristics, CHATTER is shown to provide a reasonable accuracy in classifying malware samples into their own family.

This paper only scratches the surface of order-based features for classification of malware samples, and leaves us with a lot of future work. Addressing the limitations outlined in section 4.2 is an immediate future work. In particular, we would like to explore partial-order based features and their use for fingerprinting and classifying malware samples. Those features would address noised features (intentionally, by a malware, or unintentionally—due to mixed signals in an on-the-wire deployment). Realizing the applications listed in section 4.3 using the same technique outlined and used in CHATTER is yet another future work that we would like to explore.

7. REFERENCES

[1] —. Yara Project: A malware identification and classification tool. `http://bit.ly/3hbs3d`, May 2013.

[2] E. Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[3] D. Alperovitch. Revealed: Operation shady rat.

[4] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a dynamic reputation system for dns. In *USENIX Sec. Symposium*, 2010.

[5] M. Antonakakis, R. Perdisci, W. Lee, N. V. II, and D. Dagon. Detecting malware domains at the upper dns hierarchy. In *USENIX Sec. Symposium*, 2011.

[6] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *RAID*, 2007.

[7] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. Scalable, behavior-based malware clustering. In *NDSS*, 2009.

[8] L. Bilge, D. Balzarotti, W. K. Robertson, E. Kirda, and C. Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *ACSAC*, 2012.

[9] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *NDSS*, 2011.

[10] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust*, 2010.

[11] D. Caselden, A. Bazhanyuk, M. Payer, S. McCamant, and D. Song. Hi-cfg: Construction by binary analysis and application to attack polymorphism. In J. Crampton, S. Jajodia, and K. Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 164–181. Springer, 2013.

[12] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools.

ACM Comput. Surv., 44(2):6:1–6:42, Mar. 2008.

[13] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128. IEEE, 1996.

[14] C. Gorecki, F. C. Freiling, M. Kührer, and T. Holz. Trumanbox: Improving dynamic malware analysis by emulating the internet. In *SSS*, 2011.

[15] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *USENIX Sec. Symposium*, 2008.

[16] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Sec. Symposium*, 2007.

[17] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *NDSS*, 2008.

[18] J. Halliday. Hackers attack european governments using 'miniduke' malware. `http://bit.ly/16bVldV`, February 2013.

[19] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[20] G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking command and control connections from bot traffic. In *USENIX Sec. Symposium*, 2011.

[21] J. Kinable and O. Kostakis. Malware classification based on call graph clustering. *Journal in computer virology*, 7(4):233–245, 2011.

[22] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. In *IEEE Sec. and Privacy*, 2010.

[23] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research*, 7:2721–2744, 2006.

[24] D. Kong and G. Yan. Discriminant malware distance learning on structural information for automated malware classification,. In *19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013.

[25] A. Lanzi, M. I. Sharif, and W. Lee. K-tracer: A system for extracting kernel malware behavior. In *NDSS*, 2009.

[26] M. Ligh, S. Adair, B. Hartstein, and M. Richard. *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley Publishing, 2010.

[27] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *ACM CCS*, pages 440–450, 2010.

[28] A. Mohaisen and O. Alrawi. AMAL: High-fidelity, behavior-based automated malware analysis and classification. Technical report, Verisign Labs, 2013.

[29] A. Mohaisen and O. Alrawi. Unveiling zeus: automated classification of malware samples. In *WWW (Companion Volume)*, pages 829–832, 2013.

[30] A. Mohaisen, O. Alrawi, M. Larson, and D. McPherson. Towards a methodical evaluation of antivirus scans and labels. In *The 14th International Workshop on Information Security Applications (WISA2013)*. Springer, 2013.

[31] A. Mohaisen, O. Alrawi, A. G. West, and A. Mankin. Babble: Identifying malware by its dialects. In *2013 IEEE Conference on Communications and Network Security (CNS)*, pages 407–408. IEEE, 2013.

[32] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *MALWARE*, pages 24–31, 2008.

[33] New York Times. Nissan is latest company to get hacked. `http://nyti.ms/Jm52zb`, April 2013.

[34] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel. Fast malware classification by automated behavioral graph matching. In *CSIIR Workshop*. ACM, 2010.

[35] R. Perdisci, A. Lanzi, and W. Lee. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *ACSAC*, 2008.

[36] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *USENIX NSDI*, 2010.

[37] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu, et al. The ghost in the browser analysis of web-based malware. In *USENIX HotBots*, 2007.

[38] M. Ramilli and M. Bishop. Multi-stage delivery of malware. In *MALWARE*, 2010.

[39] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 108–125, 2008.

[40] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[41] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *IEEE Sec. and Privacy*, 2012.

[42] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001.

[43] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee. Automatic reverse engineering of malware emulators. In *IEEE Sec. and Privacy*, 2009.

[44] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, 2010.

[45] W. T. Strayer, D. E. Lapsley, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In *Botnet Detection*, 2008.

[46] R. Tian, L. Batten, R. Islam, and S. Versteeg. An automated classification system based on the strings of trojan and virus families. In *IEEE MALWARE*, 2009.

[47] R. Tian, L. Batten, and S. Versteeg. Function length as a tool for malware classification. In *IEEE MALWARE*, 2008.

[48] C. Willems, F. C. Freiling, and T. Holz. Using memory management to detect and extract illegitimate code for malware analysis. In *ACSAC*, 2012.

[49] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck. A close look on n-grams in intrusion detection: anomaly detection vs. classification. In *2013 ACM workshop on Artificial intelligence and security*, pages 67–76. ACM, 2013.

[50] G. Yan, N. Brown, and D. Kong. Exploring discriminatory features for automated malware classification. In *10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2013.

[51] H. Yin, D. X. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *ACM Conference on Computer and Communications Security*, 2007.

[52] H. Zhao, M. Xu, N. Zheng, J. Yao, and Q. Ho. Malicious executables classification based on behavioral factor analysis. In *IC4E*, 2010.