Network-based Analysis and Classification of Malware using Behavioral Artifacts Ordering

Aziz Mohaisen[1,*], Omar Alrawi[2], Jeman Park[1], Joongheon Kim[3], DaeHun Nyang[4], Manar Mohaisen[5]

[1]University of Central Florida
[2]Georgia Institute of Technology
[3]Chung-Ang University
[4]Inha University
[5]Korea University of Technology and Education

Abstract

Using runtime execution artifacts to identify malware and its associated "family" is an established technique in the security domain. Many papers in the literature rely on explicit features derived from network, file system, or registry interaction. While effective, the use of these fine-granularity data points makes these techniques computationally expensive. Moreover, the signatures and heuristics are often circumvented by subsequent malware authors. In this work, 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. In its prototype form only network events are considered and eleven malware families are used. We show the technique achieves 83%-94% accuracy in isolation and makes non-trivial performance improvements when integrated with a baseline classifier of combined order features to reach an accuracy of up to 98.8%.

Received on XXXX; accepted on XXXX; published on XXXX

Keywords: Malware, behavior-based analysis, classification, machine learning, n-grams

doi:10.4108/XX.X.X.XX

1. Introduction

Malware has emerged as a challenging threat with the increased infection rates and levels of sophistication [1–3]. Examples of such threats include data exfiltration [4], denial-of-service attacks [5], and espionage [6], among many others. In order to defend against malware threats, the research community has spent a tremendous amount of efforts understanding their behavior so that detection, classification, and labeling of malware are performed with high accuracy using analysis techniques [7–13]. Threat information sharing for the improvement of the malware detection system by collecting and sharing more information also has been actively studied [14–16].

Malware analysis aims to inspect binaries in various ways, and techniques fall mainly into two categories: static and dynamic analysis. Static analysis is achieved without running the malware and consists of examining meta-data and patterns in the binaries. On the other hand, dynamic analysis utilizes artifacts that are generated by malware at runtime. The process for dynamic analysis entails executing a malware sample in a controlled environment so that changes in the environment including network traces, file system changes, memory access, and registry modifications, are all observed and recorded. The advantages of static analysis are multifold: static analysis techniques allow for comprehensive code examination of malware and are considered fast and scalable because they look for pre-computed signatures in the binaries. However, attackers can evade static analysis techniques through code obfuscation. In contrast, dynamic analysis is capable of detecting unseen malicious behavior contained in obfuscated code, but it requires setting up the sandbox environment and allocating resources for execution, which is an expensive process.

Typically, execution of a fully-fledged analysis system such as AMAL [17] requires observing interactions between

*Corresponding author. Email: mohaisen@ucf.edu a binary and the underlying system's components, such as memory, file system, registry and network, for a certain amount of time to extract behavioral artifacts and build a feature vector from which the malware family is then identified. This, in turn, requires analyzing 100s of megabytes of memory, various gigabytes of virtual file system space, several megabytes of the registry, and large network traces of packet capture.

To reduce the overhead of the dynamic analysis, while benefitting from its advantages over static analysis, we may utilize either of two approaches. First, instead of extracting deep features from the various classes of artifacts, we may only use shallow features. For example, instead of considering types of DNS responses, types of DNS records, counts per type, features of queried name servers, and so forth, we may only consider the total number of queried domains and returned responses, among others as rather a shallow feature. Similarly, instead of looking for deep content-based memory features, we may consider the shallow feature of counts over memory access. We notice that, while this approach is promising in reducing the overhead of dynamic execution it is only limited: shallow features are easy to circumvent to evade detection, and much of the overhead is consumed in executing the malware, rather than analyzing individual artifacts (e.g., online feature extraction can be done using API hooking [18]).

Another approach to reduce the overhead is by focusing on a certain class of artifacts, instead of multiple classes. We notice that not all classes of features are equally important contributors to the accuracy of malware classifiers. Consider the classification results shown in Figure 1 of the malware family Shady RAT (simply ShadyRat, or SRAT), and the contribution of various behavioral artifact classes in isolation and combined; memory, registry, file system, and network. We notice that, while the combined features extracted from the four different classes provide an accuracy of more than 95%, the accuracy per individual class of features varies, and ranges from the mid-50% to 75%, indicating that not all classes of features are equally important. This also suggests that a single class of behavior has the potential of being used independently for classifying malware.

However, we note two issues with even with the best class of features (network). First, it does not provide high enough accuracy to warrant its use independently. Second, it is unclear how difficult (and perhaps easy) it would be for an adversary to circumvent a set of features that rely on a single class of behavioral artifacts. To this end, the objective of this work is to look into a new type of features engineered from a single class of behavioral artifacts to boost the accuracy of classification and to be robust against arbitrary evasions by the adversary.

Determining the correct abstraction level at which to derive features is equally important and yet very challenging [11, 12, 19–21]. Features that accurately represent malware families are an important and defining criterion for high-fidelity malware classification. Furthermore, the level of

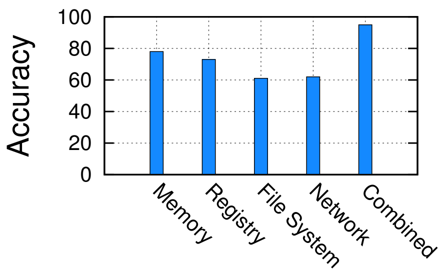

Figure 1. The accuracy of classifying malware (ShadyRAT) based on various classes of artifacts in isolation and combined (using AMAL [17], a fully-fledged behavior-based malware analysis system).

complexity needed to obtain such abstractions in operational settings determines the potential of adopting and accepting such systems at scale. For example, sandboxing and virtual execution is a privilege that comes with high costs in online detection systems [22]. When a piece of malware runs on a host, collecting deep features becomes invasive to users on those hosts. To this end, while obtaining indicative artifacts and features is very important, the degree of invasiveness is also a significant consideration for operational systems.

In this paper, we introduce Chatter, a behavior-based system for collecting run-time artifacts and feature analysis and derivation, to address some of those problems. Chatter is a robust system that relies on the order of the behavioral artifacts that malware samples generate. Chatter can be implemented as a stand-alone system or integrated into more robust execution-based systems. As its ground truth, Chatter relies on labels manually produced by malware analysts. This vetting process requires many man-hours and permits confident learning, although nicely integrated into the operation of Chatter. The man-hours, however, are not needed during the online operation.

To capture the ordering of events in malware execution, Chatter uses n-grams over abstract behavioral profiles. To boost its efficiency and achieve various desirable operational properties, Chatter focuses on one class of behavioral artifacts: network. The use of n-grams in the context of malware classification is well established. However, a novel contribution of 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. This hypothesis is supported by Forrest et al. [23] which analyzed Unix system calls in a similar fashion. Building on this assumption we attempt to classify malware using only the order of network events. Our choice of network features is not arbitrary: as discussed in the rest of the paper, network features are cheaper than file system, registry, etc. features to obtain. For example, they can be captured without residing on the same host as the malware being executed. For our study we looked at three malware families: Zeus, Darkness, and Shady RAT, representing a diversity of malware intentions.

Contributions. 1) We introduce Chatter, a system for malware analysis and classification based on inexpensive order-based behavioral features. We argue for various deployment scenarios, and address operational needs. 2) We demonstrate the operation of Chatter over various malware families using only the network interface for the source of behavioral artifacts. For this evaluation, we rely on manually produced labels. We demonstrate that Chatter is capable of accurately identifying malware family in a binary classification context. This is achievable even when limiting the number of features below a quantity commonly used in the related literature. 3) We demonstrate an in-depth analysis of Chatter across multiple evaluation criteria and outline various open directions.

This paper builds on our work in [24]. While the paper at hand is almost entirely rewritten, and borrows very little text from [24], it also incorporates new motivation, new threat model, use model, and preliminaries, new system description, and new and enhanced experiments and analysis. In particular, we updated sections 1, 2 and 4–7, while adding 3 to outline threat and use models. Technically, this work's extended contribution is towards methodical experiments and validation: unlike our prior work [24] where we study the performance of Chatter in isolation using fixed-length n-gram features, in this work we systematically evaluate Chatter with network features only, using fixed length n-gram, combined grams, and utilizing feature selection for boasted performance (as explained in sections 5.4 and 5.5; with performance gain of more than 20% in accuracy). We finally evaluate our system over a dozen malware family; section 5.2, as opposed to three families in [24].

Organization. The organization of this paper is as follows. We introduce the related work in section 2, preliminaries in section 3, the design of Chatter in section 4, its evaluation in section 5, and a discussion in section 6. We sum up with concluding remarks in section 7.

2. Related Work

The literature is rich with work on malware analysis and classification [3, 13, 25–33]. Broadly, the literature is divided into two schools of thought: signature based and behavior based techniques, with our work belonging to the latter, and most similar in nature to [13, 27, 28, 34]. These works and others can be organized according to their relationship with techniques utilizing: machine learning for malware, general behavior-based analysis, memory signatures, network-related features, evasion prevention, n-grams, and event ordering. In the following, we elaborate on this research.

Machine learning for Malware. Machine learning techniques have been used to automate classification of codes and network traffic in the literature, with hundreds of related studies. The reader can refer to recent surveys in [35] and [22] for the related work in this domain.

Behavior-based Analysis. There has been a large number of studies on using behavioral artifacts for malware analysis and classification [19, 26, 36–38]. The work of Bailey et al. in [26] has motivated many of the related works on behavior-based malware classification. In [13, 27], the authors use similar techniques for extracting features and leverage support vector machine (SVM) for classifying malware samples. Our work distinguishes itself in two respects. Although we share similarity with their high-level features, our system relies on the order of events, which exposes richer behavior. Second, we use analyst-vetted labels for evaluation, whereas the other authors use heuristics over antivirus (AV)-returned labels.

Traffic analysis. Related to our use of network features is a line of research on traffic analysis for malware and botnet detection. Such works include [39–44], with others paying particular attention to the use of fast-flux techniques [45–50]. Support for our use of DNS features for malware analysis comes in [51–53]. None of those studies IS concerned by behavior-based analysis beyond the use of remotely collected network features for inferring malicious activities and intention. Our system operates at network interface granularity to extract malware intelligence.

Evasion detection. Lanzi et al. introduced K-Tracer [54] for extracting kernel malware behavior and mitigating the circumvention of loggers deployed in the kernel by rootkits. In [55], MacBoost is used for prioritizing malware samples by distinguishing benign and malicious code segments. A system to prevent drive-by-malware based on behavior, named BLADE, is introduced in [56]. A nicely written survey on such systems and tools is found in [57].

Leveraging n-grams. Using n-grams for malware classification is not new. However, work in the literature has looked at extracting features from executables (e.g., sequences of bytes in the binary files [27]) or streams of communication traffic [58]), but not a higher-level **sequence of events** occuring while executing a malware sample. Other examples of low-level granularity attempts can be found in [55, 58–60]. Of particular interest is the concurrent work in [58], which derives n-gram network features for purposes of intrusion detection. Using network artifacts for identification of malicious activities, like botnets, is investigated in [41–43, 61, 62]. Further applications of characterizing malicious domain names using network traffic and artifacts (DNS queries, among others) are reported in [31, 53, 63].

Mariconti et al. [19] proposed MaMaDroid, an Android malware detector that builds a behavioral model from sequences of abstracted application programming interface (API) calls that are performed by an app as a Markov chain. While MaMaDroid does not specifically use the n-grams, it has the same effect in capturing the order of events (here API calls). Similar is the work of Shen et al. [64], where n-grams are used over data flows to identify the sub-flows order.

Event ordering The basic idea of event order to characterize processes was first explored by Forrest et al. in their seminal work [23]. There, it was demonstrated effective for the detection of process-level intrusions. However, that work differs from ours in three respects: (1) It is concerned with detection rather than classification, (2) it uses system calls rather than networks features, and (3) it uses whole sequences as a single feature that is easy to manipulate, rather than sub-sequences (as in n-grams) and their frequency.

3. Preliminaries

3.1. Use Model

Chatter adopts dynamic (network) analysis approach, as opposed to static analysis approaches, to classify malware using network traces that are collected by running the binaries in a sandboxed environment. In other words, and in its operational mode, Chatter does not rely on host-based artifacts to characterize the behavior of malware, rather it passively collects network traffic that is generated from the hosts to synthesize malware behavioral patterns. Because this approach is less invasive, Chatter reduces the risks of intentional evasion whereby malware samples may attempt to inject unnecessary artifacts in their behavior to fool the dynamic analysis. Notice that the injection of such random artifacts, by either randomizing the behavior of the malware or inserting unnecessary behavior, is possible, and is addressed by Mekky et al. [65].

Chatter consists of the bare-metal host(s) that run in a segregated (demilitarized) network that is connected to the Internet. A typical host would have processes associated with various activities including software updates, web traffic, and peer-to-peer traffic. Initially, these hosts are clean and only generate regular traffic, and each host is infected with only one malware for analysis and is reinitialized to clean state for every infection. In addition to the analysis hosts, a monitoring appliance is installed at key points on the network with a mirrored interface and network traffic is collected on the wire and analyzed to perform detection and classification of malware. To the observer on the wire, the whole traffic could be blended from multiple sources that include regular (background) traffic generated by the hosts and traffic generated by the malware samples. If the blended traffic is linearly mixed, then it is sent to a filter module that uses independent component analysis (ICA)[65] to separate malware traffic from background traffic. This step is taken before feature extraction and classification.

3.2. Threat Model

We assume an adversary in the form of a malicious software that is used to disrupt the operation of computer systems. Chatter is at heart a profiling system that relies on the behavioral artifact. Thus, while the adversary's goal is to remain stealthy, we assume a certain level of behavior that can be used for extracting features of such an adversary. For the purpose of this study, we assume that the behavioral artifacts generated by the adversary are unmixed with other sources of behavior. We assume that addressing such mixed behavior, where prevalent, is a secondary goal worth a separate investigation [65].

While the adversary could be aware of the defender's capabilities, including the machine learning techniques, the feature set, and the associated parameters, the adversary operates under certain constraints of achieving an end-goal, an attack that would ultimately result in behavioral artifacts used for its profiling. To this end, while the adversary may try to circumvent the system by manipulating the feature set to break the classifier, the defender would be aware of such a behavior to incorporate in further learning. Finally, of particular interest to our study is an adversary that relies on network artifacts for his operation. In particular, Chatter would operate best when such network artifacts are prominent in the behavior of the adversary. Malware that does not use the network, e.g., air gap malware, falls out of the scope of this work. From our use and threat models, we exclude the case where malware uses tunneling of encrypted traffic. Despite continued increase of SSL/TLS-based malware families [66, 67], however, the majority of malware families (about 80%) still do not rely on encryption, so we only deal with the samples which do not generate encrypted traffic in this work.

3.3. The n-Gram Model

In the following, we review the bases of the n-gram model. Interested readers should refer to [68] for more details.

Let $\mathcal{C} = \{c_1, c_2, c_3, \ldots, c_\ell\}$ be a set of ℓ unique characters corresponding to the alphabet in a language $\mathcal{L}$. A collection of characters $c_1c_2c_3\ldots c_s$, where $c_i \in \mathcal{C}$ for $1 \leq i \leq s$, is said to be a sequence if this collection is ordered where repetition of elements is allowed. A word w is a sequence $c_1c_2c_3\ldots$, where the length of w is the number of characters in it. A document $\mathcal{D}$ is defined as a collection of words $w_1w_2\ldots w_d$. Based on that, $\mathcal{L}$ is defined as a system for communication based on words and a combination of them. Both characters and words are elements (tokens) in documents (strings).

The n-grams are collections of adjacent elements in a string of tokens with the length of n. Notice that this definition is general, and it captures n-grams defined for both words and characters in robustly defined documents (strings) from a language $\mathcal{L}$. n-grams can be used to compute the probability of a token when the preceding token is given, using the chain of probability $P(c_1c_2c_3\ldots c_s)$ defined as:

$$P(c_1)P(c_2|c_1)P(c_3|c_1c_2)\ldots P(c_s|c_1\ldots c_{s-1}) = \prod_{k=1}^{s} P(c_k|c_1\ldots c_{k-1}).$$

For a bigram, we have $P(c_1^s) = \prod_{k=1}^{s} P(c_k|c_{k-1})$. Similarly, we define the probability approximation for n-gram as $P(c_1^s) = \prod_{k=1}^{s} P(c_k|c_{k-n+1}^{k-1})$, where the conditional probabilities are estimated using the relative frequency in $\mathcal{D}$ as:

$$P(c_s|c_{s-1}) \approx \frac{C(c_{s-1}c_s)}{C(c_{s-1})}, P(c_s|c_{s-n+1}^{s-1}) \approx \frac{C(c_{c_{s-n+1}^{s-1}}c_s)}{C(c_{s-n+1}^{s-1})} \quad (1)$$

One way for building a feature vector for representing a certain abstraction using $\mathcal{L}$ is through the above probability estimation. Alternatively, we can calculate the probability directly by scanning through $\mathcal{D}$ to find all grams that meet the definition above for the given length n. Defining words over behavioral artifacts is possible, as examined in [65] using the same toolsets used in this paper. However, in this paper, and for the development of Chatter, we limit our attention to the case where behaviors are at the character level: we do not group those behavioral artifacts and consider the entire document $\mathcal{D}$ as a single word with as many characters as there is in the original behavioral profile.

4. System and Design

Chatter characterizes malware samples by executing them, using this intelligence for training purposes. For online operation, this capability is not required. We proceed by discussing Chatter's design goals and then detailing how these are achieved in practice.

4.1. Design Goals and Requirements

In developing Chatter, we have several idealized functional and non-functional goals in mind. In particular, we develop Chatter so that it is cost-effective, less invasive, generalizable and multi-purpose, robust to behavioral changes, and accurate. In the following, we elaborate on each of those goals.

Cost effectiveness. Feature extraction should be computationally inexpensive, particularly in online operation. This cost effectiveness can be achieved by either limiting the number of classes of artifacts used for characterizing malware to a smaller set (e.g., network only, as opposed to fully-fledged systems such as AMAL [17], which uses artifacts that pertain to network, memory, file system and registry usage) or by deriving shallow features across multiple classes. In Chatter, we emphasize the first approach by limiting ourselves to network features only.

Less-invasiveness. While possible to analyze malware in instrumented and virtualized environments, which is the approach we follow for building a baseline, it is desirable to collect such artifacts while running on the "natural" host OS. A system that can be deployed externally to observe malware is ideal. We keep in mind that this is only a design objective for the final system, not necessarily a requirement on how that system is arrived at.

Generalizable and multi-purpose. While the stated goal of Chatter is to characterize malware samples by their behavior, the system should be flexible enough for re-purposing outside the malware realm. In Section 6.3 we show two such applications that benefit from such generalized capabilities.

Robust to behavioral changes. Given that many malware families evolve over time to circumvent behavior-based techniques, one goal of our system is to resist this evolution by providing flexible and longitudinally-aware techniques.

Accurate. An ideal system for malware classification should aim to provide the greatest coverage while minimizing false positives. As such, when designing Chatter, we consider operationally acceptable accuracy within other optimizations (such as cost-effectiveness) of particular importance.

4.2. System Workflow and Operation

Chatter's overall workflow is visualized in Fig. 2. In describing this workflow we begin with our sandboxed execution environment and its output. We then describe how this output is transformed to make use of the n-gram technique to create a feature vector for each malware sample, which is then used for training a model to classify various species of malware in an automated manner. We note that any sandboxed execution environment (or bare-metal execution) can be used for extracting the features used in Chatter. We use our proprietary system named AutoMal, and described in greater depth in [69]. AutoMal requires very little effort to be repurposed as the dynamic execution component of Chatter. AutoMal is a Windows-based system capable of collecting low-granularity artifacts speaking to how malware samples interact with memory, file system, registry, and network interfaces. AutoMal takes as input binaries that are likely to be malware. It consists of 4 components: submitter, controller, workers, and back-end storage. Input samples are queued until resources become available. The controller initiates virtual machines (VMs), loads configurations, and runs the malware sample. Once execution is completed the collected artifacts are logged in the back-end storage unit.

Sandboxed Execution. Chatter starts its operation by ingesting malware samples (provided by analysts as part of their research activities in security operations, customers interested in understanding classes of malware, or obtained from malware feeds of antivirus scanners [69]), as shown in step 1 in Fig. 2. Given the limited resources and large number of samples at any point of time, our sandboxed execution environment (part of AutoMal [69]) has a controller that checks for available VMs in the system (or a bare-metal units) for executing samples from the malware samples queue. If a VM (or bare-metal unit) is available, the controller runs this VM with a set of configurations provided by the analyst and executes the malware sample in the VM. The configurations are passed as a high-level script by the analyst: those configurations determine the operating system used in the VM, the virtual devices and resources associated with the VM, the software packages and programs installed in the VM, etc.). Upon executing the malware sample for a certain amount of time, as shown in step 2 in Fig. 2, our fully-fledged system collects a variety of dynamic execution features. Those include memory artifacts collected by various memory forensics and analysis routines (e.g., memory access, memory deletion, memory modification, size of reads and writes, utilized address space, etc.), file system artifacts (e.g., counts and content-related artifacts of files read, written, deleted and modified), registry artifacts (e.g., similar to the file system artifacts), and network artifacts. The amount of time a sample is executed is determined by an analyst based

Table 1. Events used in composing the behavioral documents of malware samples.

Event class	Component events
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

on several considerations, including a high-level notion of malware type and performance requirements

Because Chatter only relies on one class of artifacts, namely network artifacts, AutoMal is repurposed where only network artifacts are considered, as shown in step 3 in Fig. 2. Those network artifacts are collected and indexed with the order of their appearance in the execution of the malware sample using timestamps. Among the network artifacts collected by Chatter, we mainly considered high-level header-based artifacts shown in Table 1. Those artifacts are mainly header-based, which means that they can be quickly obtained in real-time while executing the malware sample. We note that other deeper features from the contents (payload) of the network artifacts could also be utilized, although we avoid using such artifacts simply because due to efficiency and scalability reasons.

The artifacts in Table 1 consist of multiple classes, including IP and port, connection (type), request and response (type), size information, and DNS-related features. For ports, we focus on a set of ports that are mostly utilized by malware samples in our analyzed feeds, some of which are generic services (e.g., ports 20, 21, 22, 25, 53, 80, 102, 110, 143, 389, 443, 465, 587, 636, 993, 995, 6665, 6347, 6679, 6697, and 8080 are used in our analysis). The rest of the (type) artifacts are as highlighted in Table 1, whereas the size information is highlighted in quartiles (see [69] for more details). In short, we define the size of a flow (request or response) as a number between 1 and 4 (inclusive). This number depends on which quartile the size of the flow fits within. The quartiles are computed over all the flows (requests and responses) associated with the execution of malware samples.

Behavioral Documents. Chatter abstracts the raw network behavior and artifacts obtained in step 3 in Fig. 2 to compose a "behavioral document." The behavioral document, as shown in step 4 in the same figure (for illustration purpose only), is simply an abstracted representation of the network artifacts to simplify their processing. To compose the behavioral document, we start with the network artifacts, and by fixing set of "events" (in advance) as shown in Table 1, we map those events into a set of unique characters in an arbitrary alphabet. Then, using the network artifacts in step 3 we replace each event with the corresponding character to obtain a simplified representation of the network artifacts that preserve the order of the events. For example, in step 4 in Fig. 2, DNS query (of, say, MX type) is mapped to d, an HTTP request (of, say, type GET) is mapped to h, and so forth. By mapping the corresponding appearances of the events in the set in Table 1 to the corresponding unique alphabet for the network behavior of a malware sample (and discarding unmapped behavior), we obtain the "behavioral document", as shown in Fig. 2.

n-gram Tokens. The next step in the operation of Chatter is the extraction of n-gram tokens from the behavioral document using the n-gram technique in section 3.3. By taking as an input a parameter n, Chatter calculates the tokens of length n in the behavioral document, such that those tokens consist of adjacent characters in the behavioral document. For example, as shown in step 6 in Fig. 2, by starting with a behavioral document of `dhrhrhrd`, and for $n = 3$, Chatter calculates the following tokens as the output of the n-gram extraction component: `dhr`, `hrh`, `rhr`, `hrh`, `rhr`, and `hrd`.

n-gram Features. To extract features to represent a malware sample, we use the frequency of n-gram tokens obtained in the previous subsection and as illustrated in step 7 in Fig. 2. For instance, for the sample above, we build an index of the unique tokens and take the count over those tokens as the feature vector. Namely, for the same example, the index calculated over the tokens is [`dhr`, `hrh`, `rhr`, `hrd`, ...] with a feature vector of [1, 2, 2, 1, ...]. Note that the index is calculated over the superset of tokens that appear in the execution of all malware samples. Furthermore, an index and feature vectors could be constructed from the union of multiple grams (e.g., for an arbitrary n, we may consider the index of the feature vector as only the unique grams of length n, or all unique grams with a length less than or equal n; we elaborate more on the choice of the feature in the evaluation).

4.3. Machine Learning Subsystem

The main purpose of Chatter is to provide means for automated malware classification using the behavioral features obtained through dynamic execution. To this end, Chatter supports various supervised machine learning algorithms for classification. Starting with a golden labeled dataset, Chatter builds a model utilizing the malware features extracted in the previous section. Upon validating the model using standard cross validation and accuracy measures, Chatter then uses the built model to classify malware in the wild to a family of interest. A simpler task that we also examine in the evaluation is the power of Chatter of detecting malware (this is, the classification of software into two classes of malicious and benign). In the following, we elaborate on the machine learning component of Chatter, including procedures and algorithms.

Building Ground Truth. Devising methods for building ground truth as an open question in the security research community in particular and the machine learning in general.

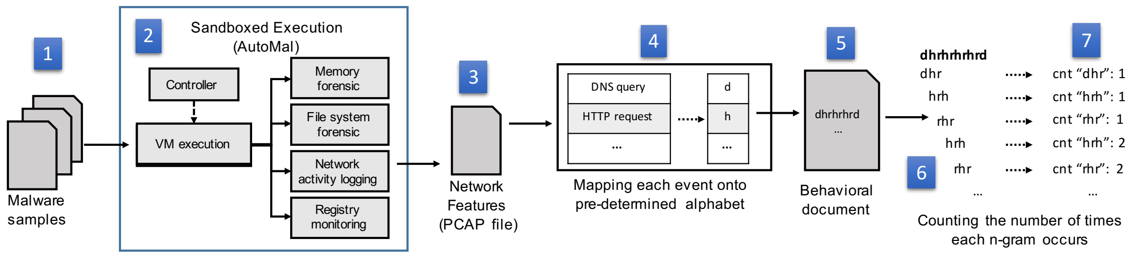

Figure 2. Chatter's sandboxed execution and feature extraction utilizing the network profiling in AutoMal and n-gram-based features extractor.

Methods that rely on manual inspection do not scale, whereas automated signature-based methods that utilize antivirus scanners and labels are susceptible to inaccuracies and incompleteness. Fortunately, and as part of our security operations, manually-assigned names and labels of malware samples were given utilizing the variety of manual analysis, reversing, and memory signatures. While the method is costly, if it is to be used only for the operation of Chatter, such ground truth is obtained independently. We use this ground truth (more details in the evaluation). However, any other reliable ground truth source (that could potentially improve over antivirus scanners) can be utilized for the operation.

Training and Validation. The task that Chatter tries to achieve is a binary classification: telling if a malware sample belongs to a malware family (or a class label) of interest or not. Given a set of malware samples represented by their feature vectors as extracted in the previous section and the corresponding labels for each sample, Chatter builds a model for the class of interest and other families (collectively represented as a second class; in malware detection, such a class corresponds to benign software). The model is trained (built) using part of the dataset and validated using the rest of the dataset. Independent of the algorithm used for building the model (detailed in section 4.3), Chatter uses the k-fold cross-validation method for building the model, thus addressing the problem of overfitting. For evaluating 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 the k possible folds. At the end, the result of the classification algorithm is computed as the average over the k runs. We set $k = 10$ due to its common use.

Upon training the model and testing it, and further establishing a confidence in the model, the underlying features, and the used parameters in the algorithm, we use the model in the wild for classifying (or detecting) malware. This is, we assign a label to a malware sample with the unknown label based on the determination made by our model for the given algorithm (details of how such determination is done is the testing phase of the algorithms in the next section; 4.3).

Machine Learning Algorithms. Three machine learning algorithms are supported by Chatter: 1) the k-nearest neighbor (k-NN), 2) support vector machine (SVM), and 3) decision tree classifiers. All three algorithms are intended for binary supervised learning and are capable of identifying the membership of a malware sample into one of two classes. Details on the operation of those algorithms are provided below for the completeness of our presentation of Chatter.

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 the following optimization: $\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{\ell} \xi_i$ 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 this 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. In evaluating Chatter, we utilize a single split tree for two-class classification using all of the features provided by Chatter. In particular, we use a small variation of the C4.5 algorithm. For a target class label $Y = y_1, \ldots, y_n$ (here, $n = 2$) and a set of feature vectors $\mathbf{x}_1 \ldots, \mathbf{x}_f$, at each internal node of the tree we apply a test to one of the inputs, namely $\mathbf{x}_i$, determining whether to go 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 the leaf. There are variations of decision trees in the literature to provide better results (e.g., random forests). We did not try any of those techniques since this technique provided reasonable results. We leave integration of such techniques as future work.

The k-nearest-neighbor. The k-NN is a non-linear classification algorithm. In the training phase, we provide the algorithm two labels and a set of training samples (that are simply stored for the testing phase). In the testing phase, for each sample vector $\mathbf{a}$, we assign the label most frequent among the training samples nearest (using the Euclidean distance) to $\mathbf{a}$. We refer the reader to a textbook explanation of the technique in [70].

5. Evaluation

To evaluate our work we begin by describing the malware samples utilized and how ground-truth is produced. This data is put through the Chatter workflow and feature vectors are computed. We then apply our machine learning technique of choice, producing evaluation metrics which are then interpreted with respect to system performance.

5.1. Ground-truth and Labeling

Labeling malware to establish a ground-truth is an important step in any supervised classification task. Prior literature relies heavily on family labels and names provided by anti-virus scanners. Several recent works [22, 26, 71] have shown such scans to be unreliable. This is understandable to some extent, as these AV scanners are often designed with the primary goal of distinguishing malware from benign code – not distinguishing between malware families.

To this end, the malware samples we utilize have been collected over a considerable period of time, enabling expert analysts in Verisign's organization to manually identify and label them. This process is time-consuming; a previously unseen malware sample averages 10+ hours of manual characterization. However, family discovery is done in parallel with other important tasks. This time is not spent analyzing the binary just for this research initiative. Instead, Verisign analysts are contracted by customers to produce detailed reports on the modus operandi of a malware sample.

In addition to customer submitted binaries, partnering anti-virus vendors and analyst's research endeavors also contribute to our malware repository. Yara signatures [72] are sometimes applied to weed out irrelevant samples and a system called AMAL [69] is applied to characterize samples from well-understood families. We emphasize that the majority of samples used in this study come from customers, where manual efforts are used for the labeling of samples.

5.2. Dataset and Malware Samples

For the evaluation of Chatter, we use 11 malware families used in [8]: Avzhan, Darkness, Ddoser, Jkddos, N0ise, Shady RAT (SRAT), DNSCalc, Lurid, Getkeys, ZeroAccess, and Zeus. As we will soon describe, these families cover a wide range of network behavior. Our system is applicable for all malware families and our proof-of-concept selection is for demonstration purposes. Subsequent to the analysis described herein, we also ran our system on other families, including Ramnit, Bredolab, SillyFDC, and Virut. The systems' operation and accuracy during these larger trials were consistent for the results we present for our 11-family experiments.

Each malware sample in these families is obtained from an operational product, where sources of the malware samples include Verisign customers, partnering antivirus vendors, and researchers. Once the malware samples are fed into AutoMal they are executed for a fixed amount of time to generate artifacts. Table 2 reports on the quantity of the sample we

Table 2. Malware families used in the evaluation of Chatter, including set size and the average number of events per execution trace.

Family	Quantity	Ch. Avg.	Description
Avzhan	3458	70.31	Commercial DDoS bot
Darkness	1878	61.47	Commercial DDoS bot
Ddoser	502	57.51	Commercial DDoS bot
N0ise	431	77.13	Commercial DDoS bot
Jkddos	333	120.3	Commercial DDoS bot
Shady RAT	1287	52.74	targeted; government and corps
Getkys	953	63.04	targeted; targets medical sector
DNSCalc	403	82.37	targeted; US defense companies
Lurid	399	50.41	targeted; initially targeted NGOs
Zeus	1975	50.74	Banking, targets credentials
ZeroAccess	568	49.93	Rootkit, monetized by click-fraud

virtualized and the average document length that resulted. To foster transparency and reproducibility of results, we release the dataset used in this work to the larger community. We note that the different families in our dataset belong to one of three classes: DDoS, targeted, or mass-market malware.

For every malware family set, we also have an equally-sized sample of random malware samples drawn from an expansive malware repository. In our proof-of-concept evaluation, we, therefore, treat family membership as a *binary classification task*. We now discuss each of these families in greater depth:

⋄ **Zeus.** One of the Trojans that attack financial sector through stealing information from the infected computers. To steal credentials, this Trojan hooks Windows API functions responsible for the communication between clients and the bank's website and modifies the returning results to hide its activities.

⋄ **Avzhan.** The DDoS botnet that was first reported in 2010 [73]. This botnet is similar to IMDDos, a Chinese process-based botnet announced by Damballa around September 2010 [74], where both families can be commercially hired to attack targets of interests. The owners of the botnet claim on their website that their botnet can strictly be hired to attack non-legitimate websites, for instance, gambling sites.

⋄ **Darkness.** a.k.a. Optima, is a commercial malware family developed by Russian criminals and released in 2009. As of the end of 2011, the 10th generation of this bot was released. This malware performs several functions, including launching DDoS attacks, stealing credentials and using infected machines for launching traffic tunneling attacks [75].

⋄ **DDoSer.** a.k.a. Blackenergy, was reported and analyzed in 2007 [76], is a DDoS malware that is capable of performing HTTP DDoS attacks. This malware is unique in the sense that it can target more than one IP address per DNS record.

⋄ **JKDDoS.** Malware that targets the mining industry [77]. The malware was first reported by Arbor Networks in 2011, and its first generation was observed in September 2009.

⋄ **N0ise.** The DDoS tool used to recruit other bots to attack victims using HTTP, UDP, and ICMP flood, among other methods. Other functionalities of this tool include stealing credentials and downloading and executing other malware [78].

◇ **ShadyRat.** is a targeted malware that is used to steal sensitive information like trade secrets, patent technologies, and internal documents. The malware employs a stealthy technique when communicating with the C2 by using a combination of encrypted HTML comments in compromised pages or steganography in images uploaded to a website [79].

◇ **DNSCalc.** a.k.a. APT12, is a malware is targeted towards research sector, where it steals sensitive information through using the responses from the DNS request to calculate the IP address and the port number used for communication [80].

◇ **Lurid.** Malware that targeted US government and non-governmental organizations (NGOs), although it seems that there is no relationship between the targets indicators. This perhaps implies that the malware family is being used commercially as a hit man [81]. Three hundred attacks launched by this malware family were targeted towards 1,465 victims and were persistent via monitoring using 15 domain names and 10 active IP addresses.

◇ **Getkys.** a.k.a. Skyipot, is a single-stage Trojan that targets aerospace, defense, and think tank organizations, through running and injecting itself in three targeted processes: outlook.exe, iexplorer.exe and firefox.exe. This Trojan communicates via HTTP requests and uses two unique and identifiable URL formats, such as the string "getkys" [82].

◇ **ZAccess.** a.k.a. ZeroAccess, is a rootkit-based Trojan that targets most Windows OS and was reported first by Symantec in July 2011. The Trojan is mainly used to enable other malicious functions on the infected machine based on a pay-per-click advertising model. ZAccess is generally used to download other malware, open backdoors on the infected machines, among other functions [83].

5.3. Feature Space and Space Reduction

Running a malware sample in a sandboxed environment results in many artifacts and large PCAP files. Not all of these artifacts and the associated features are relevant nor meaningful in identifying a malware sample. Thus, we rely on expert *domain knowledge* to identify the network-related events of interest. To this end, Table 1 highlights some of the 26 network-related events used in our analysis. Per Table 3, we observe that some events do not occur for some families (e.g., $n = 1$) and far greater unique combinations of events never occur (e.g., where $n > 1$ there is no permutation-scale growth). This makes empirical the advantages of using the sparse feature representation. For example, when $n = 8$ our feature vector needs only $\approx$ 3,800 entries (the size of the feature space), rather than the 2.08^{11} entries needed for exhaustive representation of all combinations when $n = 8$.

5.4. Evaluation Metrics and Scenarios

For a binary classification problem, in which it is required to determine if a given malware sample belongs to the class of interest S or not, we define the following possibilities: (1) True positives (T_p) are those samples correctly identified by the machine learning algorithm to belong to the class S. (2) False positives (F_p) are those samples incorrectly marked by the machine learning algorithm to belong to S. (3) True negative (T_n) are those samples marked by the machine learning algorithm correctly not to belong to S. (4) False negative (F_n) are those samples incorrectly marked by the machine learning algorithm not to belong to S (they are actually in S). Using these outcomes and associated magnitudes, the precision, recall, accuracy, and F1 score are defined as:

– Precision (P) = $T_p/(T_p + F_p)$,
– Recall (R) = $T_p/(T_p + F_n)$,
– Accuracy (A) = $(T_p + T_n)/(T_p + T_n + F_p + F_n)$,
– F1 score (F1) = $2 \times (\text{P} \times \text{R})/(\text{P} + \text{R})$.

In the subsequent analysis, we scale all evaluation metrics to a percentage range by multiplying the result by 100.

Scenarios. In experimenting with Chatter, we consider the following scenarios. 1) Using all features resulting from the n-gram representation for a given n. 2) Using the combination of all features resulting from the n-gram, inclusive of all features with a gram of size less than or equal to n; this is, when a given n is used the feature set would include, grams of size $n \ldots 1$. 3) Using the top significant features selected using the recursive features selection (RFS). RFS takes in two inputs, k value, indicating the number of features desired, and n features. RFS generates $n - 1$ models by removing 1 feature from the n features, then evaluates each $n - 1$ model, then selects the best performing model. The process continues m steps until $k = n - m$. 4) Using multiple algorithms on the same set of features to understand the impact of different algorithms on the performance of our classifiers.

5.5. Quantitative Results

We now present results from our evaluation. This section consists primarily of quantitative results, whereas later sections analyze and discuss these results in greater depth.

Results in isolation: For this evaluation, we use three families that represent the three different malware classes in our dataset: Zeus (mass-market), Darkness (DDoS), and ShadyRat (targeted). The main purpose of this experiment

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

Family / n	1	2	3	4	5	6	7	8
Avzhan	24	102	248	476	910	1506	2304	3252
Darkness	24	103	243	461	875	1503	2266	3149
Ddoser	26	107	250	491	900	1606	2636	3776
N0ise	24	106	249	480	891	1678	2264	3626
Jkddos	25	103	244	469	941	1703	2307	3122
SRAT	25	105	247	460	877	1536	2337	3300
Getkys	24	102	242	470	893	1659	2277	3364
DNSCalc	26	107	249	468	907	1578	2292	3379
Lurid	25	103	243	467	956	1539	2354	3130
Zeus	24	102	250	481	943	1690	2638	3794
ZeroAccess	26	108	250	489	913	1520	2306	3368

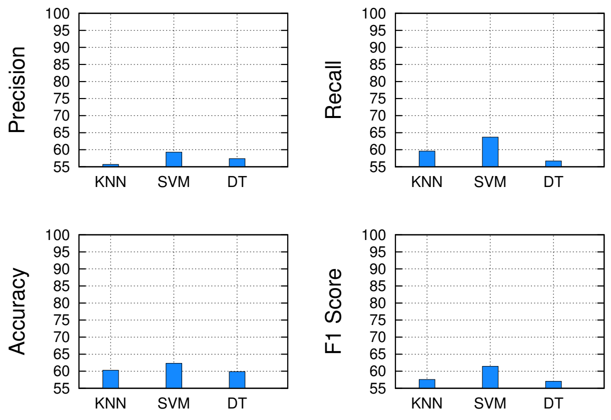

Figure 3. Performance measures for the Shadyrat malware family with network artifact classification using Chatter atop a baseline classifier

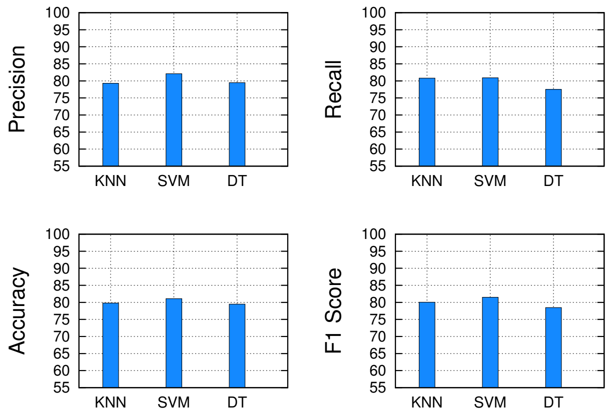

Figure 4. Performance measures for the Darkness malware family with network artifact classification using Chatter atop a baseline classifier

is to understand the power of various machine learning algorithms and parameters when studying our datasets and preliminary features in isolation. Thus, we start with $n = 1$, which corresponds to the bag-of-words method in the literature [17], the three aforementioned machine learning algorithms, the four evaluation metrics, and the three malware families. The results are shown in Figs. 3, 4, and 5. Among other observations, we notice the following. First, the performance, across all evaluation metrics, of the baseline feature is insufficient in almost all cases of malware families. Second, the performance of our baseline experiment varies greatly across families, and best results across all evaluation metrics is obtained with mass-market malware (Zeus), followed by DDoS malware (Darkness), then by targeted malware (ShadyRat), which is unsurprising pattern that holds

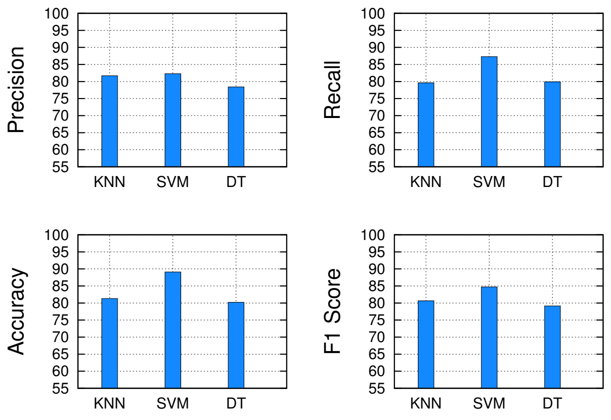

Figure 5. Performance measures for the Zeus DDoS malware family with network artifact classification using Chatter atop a baseline classifier

for other malware families as verified through our analyses. Third, across all experiments, we notice that SVM provides the best results across all evaluation metrics, and the ranking of the two other algorithms is inconsistent and depends on the family studied. However, in all cases, the difference between the different algorithms is insignificant.

Based on the baseline established in the first observation, we pursue the study of the impact of the n-gram features, and how they impact the performance of classification. Based on the third observation, we limit our attention to the SVM as the algorithm of choice when studying the impact of the n-gram parameters on the performance of classification. For the lack of space, we also only evaluate our system based on its accuracy. In the subsequent analysis, we perform classification for all of the eleven malware families in our dataset.

Evaluation using individual n-gram features. In this experiment, we focus on how using the new set of n-gram features affect the performance of the classifier. For this experiment, we use the SVM as the classifier of choice, for its better performance in the previous experiments. We study the performance over all families. In running Chatter, we use individual features resulting from using the parameter n. Where n is fixed, only features of n consecutive events are used in our feature vector. Results are shown in Table 4. Based on those results, we make two observations. First, there is a monotonically increasing trend in the accuracy of the SVM classifier with the increase of the parameter n, even without features selection. This improved performance is clear across the board. Second, we notice, by experimenting with the 11 families in our dataset, a consistent ranking of the power of network-based features in general, and Chatter in particular, in profiling, characterizing, and detecting mass-market and DDoS families than the targeted families. This insight is further highlighted in the ordering of three sample families belong to each class, in Fig. 6.

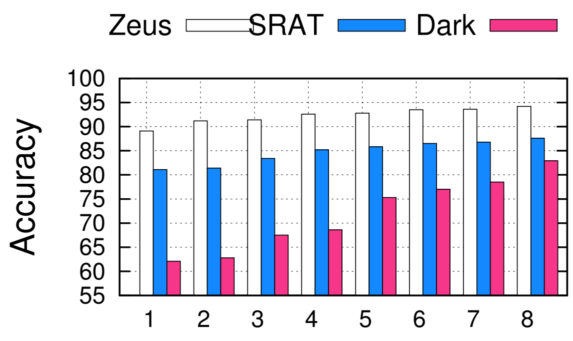

Figure 6. The accuracy of classifying various malware families with varying n.

Table 4. Accuracy of classifying various families for different values of n when calculating the n-gram individual features.

family / n	1	2	3	4	5	6	7	8
Avzhan	74.2	74.8	76.5	76.7	78.7	83.5	86.0	91.2
Darkness	81.1	81.4	83.4	85.2	85.8	86.5	86.8	87.6
Ddoser	84.1	84.9	85.6	86.5	92.8	93.5	93.7	94.1
N0ise	88.0	88.1	88.8	89.2	90.7	91.8	91.8	92.4
Jkddos	75.4	76.1	79.1	85.2	85.3	85.6	85.9	87.7
ShadyRat	62.1	62.8	67.5	68.6	75.3	79.0	80.3	82.9
Getkys	72.9	72.9	73.2	74.8	75.8	77.0	78.5	79.8
DNSCalc	75.7	77.0	78.2	78.4	79.4	81.3	83.5	84.4
Lurid	66.5	67.3	68.2	69.7	76.5	77.5	77.7	85.7
Zeus	91.2	91.2	91.4	92.6	92.8	93.5	93.6	94.2
ZeroAccess	83.3	83.8	84.6	84.8	89.7	91.6	92.9	93.0

Table 5. Accuracy of classifying various families for different values of n when calculating the n-gram combined features.

family / n	1-2	1-3	1-4	1-5	1-6	1-7	1-8
Zeus	93.2	93.5	93.9	94.0	94.1	95.3	96.2
ZeroAccess	84.9	85.2	85.3	90.6	94.1	94.5	95.2
ShadyRat	63.2	68.2	71.2	75.7	79.4	82.8	85.8
Getkys	74.7	76.1	76.7	77.3	78.2	79.8	82.6
DNSCalc	78.3	79.4	79.9	80.7	84.2	85.4	86.0
Lurid	69.0	70.0	71.2	78.1	78.9	79.2	88.4
AVZhan	77.3	79.1	79.4	81.3	83.9	87.5	93.2
Darkness	83.7	83.9	86.6	86.8	87.6	88.7	89.8
Ddoser	86.4	86.5	88.6	93.7	94.5	95.4	96.0
N0ise	89.3	90.3	90.5	92.9	92.9	93.1	93.6
Jkddos	78.5	81.0	85.5	85.5	87.9	88.5	89.4

Evaluation using combined n-gram features. In this scenario, and for a given n, we compile a set of features for all combinations of events that are consecutive and are of the length less than or equal to n. For example, where $n = 3$, we include features for lengths 1, 2, and 3. The results are shown in Table 5. Overall, we notice an improved accuracy across the board in classifying the various families.

Evaluation with selected features. In the following, we show the results with features selection over the previous experiment. In particular, we limit our attention to the 10% top features in each family and compute the accuracy of the classifier for the different families. The results are shown in Table 6. Overall, we notice an improvement across the board. More importantly, we notice a final classification accuracy of 83%-98.9%, as compared to an initial accuracy of 62.1%-91.2% (with more than half of the families having the accuracy of less than 80%). Notice that this improvement in accuracy is not the only improvement, but also includes a massive speed-up for the reduction in the number of features used for classification to only the most significant (10%).

Table 6. Accuracy of classifying malware families with varying n values with recursive feature selection.

family / n	1-2	1-3	1-4	1-5	1-6	1-7	1-8
Zeus	94.2	95.2	95.8	96.4	96.5	97.0	98.9
ZeroAccess	85.9	85.9	89.1	93.4	94.5	94.5	95.9
ShadyRat	66.5	69.5	71.4	78.9	83.7	84.5	86.8
Getkys	75.9	76.2	76.4	77.7	80.8	81.6	83.0
DNSCalc	78.9	80.2	81.8	82.4	82.6	85.6	86.2
Lurid	70.0	71.7	74.4	77.2	80.0	82.1	89.6
AVZhan	78.3	79.2	81.9	81.9	87.4	88.4	94.9
Darkness	86.8	87.8	87.9	88.6	89.4	90.4	90.9
Ddoser	88.3	89.9	91.8	96.2	96.8	97.0	97.3
N0ise	91.3	91.9	92.8	93.8	94.4	96.6	97.2
Jkddos	79.1	81.1	88.1	88.4	89.4	90.3	93.6

6. Discussion

When $n = 1$ ordering plays no role, whereas there are tremendous ordered dependencies when $n = 8$ (our upper limit). We observe that malware families tend to peak in performance when $n = 8$. This observation is further validated when combining features and feature selection. The accuracy graphs in Figs. 3, 4 and 5 show an upward trend as n grows. We conclude that order does improve classification by about 7% to 23%.

6.1. Meeting Design Requirements

In the following we highlight how Chatter meets the design requirements outlined in Section 4.1:

◇ **Cost-effectiveness.** Chatter's cost-effectiveness is two-fold: (1) It uses only a single class of (network) artifacts, and (2) It abstracts features from curated event traces rather than raw interface dumps. Prior literature has shown a system characterizing malware using only network artifacts can run an order of magnitude faster than a system that looks at a large spectrum of features. For example, AMAL [69] is a fully-featured infrastructure that ideally utilizes 128 virtual machines towards processing 23,000 malware samples daily. Chatter, on the other hand, could process 370,000 malware samples per day using the same infrastructure. This number significantly exceeds the sample quantity Verisign's operations receive and analyze on a daily basis. Indeed, the number is also greater than the 250,000 samples a popular antivirus provider like Sophos detects and analyzes daily [84].

◇ **Less-invasiveness.** Using network features makes Chatter less invasive. The network events can be gleaned on the network without having to reside on the host machine. However, this capability is also susceptible to noise from other system processes using the network interface in parallel with the malware sample. This could be limited by running the malware and the host in a monitored mode of operation.

◇ **Generalization and flexibility.** Chatter's ability to evolve with malware binaries is straightforward given its operational context. Because novel malware is being created, analyst efforts continue to be brought to bear on samples' reverse-engineering and analysis. Family labels are a cheap side-effect of this ongoing demand, resulting in a wealth of expert-annotated and longitudinal ground-truth that can be utilized via periodic retraining.

◇ **Accuracy.** In isolation, Chatter's accuracy is less than that of full-featured systems, e.g., AMAL [69]. However, we still contend that our performance is *operationally acceptable*. The contexts in which one needs to make a family classification are very different when determining the presence of malware. Remember also that accuracy can be improved by using more expensive techniques (possibly as a second-pass if Chatter's models indicate low classification confidence). This work is also interested in the trade-off between complexity, accuracy, and operational costs.

6.2. System Limitations

We acknowledge various shortcomings of Chatter, and examine how a knowledgeable attacker could utilize gamesmanship to circumvent our classification strategy.

Noised features. As with most behavior-based systems for malware classification, Chatter performs best when malware samples do not produce extra information to disguise their behavior and manipulate machine learning algorithms. However, unlike systems that make use of exact matching of behavior profiles, Chatter provides some flexibility and robustness in the grouping patterns based on n-gram features. The problem is a generic one which we address in two ways. First, we emphasize that not all the features generated by a malware sample need to be used by learning algorithm: a feature selection algorithm can be used to reduce the impact of the noised features. Second, regardless of injected noise, certain events in the operation of a malware sample must happen in the same partial order. Our future work to address this limitation is to derive features of those events as they happen in their partial order by filtering out the noise between them. While this might seem to require a deep understanding of the studied malware families and their expected behavior, well-understood signal processing techniques show potential in this domain.

Adaptive malware. Certain forms of malware are capable of changing their behavior based on the environment in which they are run. This creates issues for Chatter as it does for other behavior-based sandboxes. We address this in two ways:

First, AutoMal, the sandbox environment for Chatter generates patches to deceive malware samples by providing registry values indicating execution is occurring on bare metal. Second, for sophisticated malware that does not respond to those patches, malware samples are actually run on bare metal (or using hardware virtualization). Of course, this a problem specific to the generation of behavior profiles, whereas actual operation is unaffected by such manipulations.

Continuous training and cost of labeling: Because of the evolution of malware samples, continuous training is needed in our system to adapt to changes in artifacts generated. While this issue might seem an inherent 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.

Our objective in this work is characterizing a threat model against Chatter by defining ways where an adversary can alter the pattern of network traces when running malware samples in a dynamic analysis environment. An adversary may alter the sequence of traces in various ways hoping to confuse monitoring systems and to make the extracted features less discriminatory, either by injecting new packets into the network or modifying existing ones or dropping packets in transit if they can intercept them.

We argue that sensitivity of classification in Chatter is affected by environmental noise or by malicious actions intended to poison traces of execution using operations like: addition, deletion, and modification of behavioral events. Consequently, an adversary who executes these operations seeks to record distorted patterns as input to the classifier and therefore hopes to evade detection by generating faulty results. We identify potential attacks that can target the classifier and affect the overall performance of Chatter. Each of those attacks, while ranging in sophistication, rely on unique and basic features of the proposed approach for discriminatory feature extraction. These attacks can be modeled by simple linear combination attacks or they can be more sophisticated such as non-linear or spatial attacks.

Linear attacks. With linear attacks, the source of the attack can be modeled as a linear combination of background traffic generated from regular hosts (based on web browsing, software updates, etc.) with network events generated by the malware sample. Noise in the behavioral sequence can affect the classifier's performance and to overcome this requires either applying a proper filter or use of tight monitoring in sandboxed environments. However, malware samples may actually simulate linear noise by injecting unnecessary artifacts in their behavior to fool the dynamic analysis, thus mimicking background traffic and evading detection by simulating multi-blended traffic. Methods such as independent component analysis (ICA) can be used to separate linearly combined traffic by analyzing distributions of features extracted from mixed traces into two estimated distributions, provided that traffic sources are statistically independent.

Non-linear attacks. Unlike linear attacks which assume traffic is linearly mixed, spatial attacks are based on non-linear combinations of the features that exhibit spatial distortion properties. Two techniques seemed promising for defending against spatial attacks, namely: sequence alignment and skip-grams. Sequence alignment is a well-established method in bioinformatics used for identifying regions of similarity between two sequences that might be a consequence of functional, structural or evolutionary relationships. Typically, a ground truth of clean sequence is used to align various noisy sequences. On the other hand, skip-grams can be used to model noisy sequences by finding the pattern of characters (or words) that do not have to be consecutive. For example, a 1-skip grams algorithm would be applied the same way as 2-grams, but by skipping a word in every two grams in the analyzed document. They are considered a generalization of n-grams model which can be used to overcome noise in the sequence.

6.3. 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: (1) *Process-based DDoS detection:* While our system studies a specific DDoS malware family, our system can 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 such attacks. (2) *Advanced persistent threats:* Often such threats (process-based) result in many artifacts that are generated over a long period of time, rendering research systems less effective in characterizing them. One potential area of improvement is to rely on the inter-event patterns they generate, using Chatter.

One potential application to the technique is what we term as the "adversary incubation", in which an attacker is led into a controlled environment and observed over a long period of time. The behavior of the adversary can be then used to characterize him and using his network activities in particular. We notice that the adversary incubation is one step for understanding the "advanced persistent threat" (APT), and is widely used in the industry. The incubation usually result in network artifacts generated by the adversary and spread over a long period of time, as a result of low rate activities that the adversary hope to go unseen by anomaly detection defenses. Utilizing systems like AMAL for understanding the adversary might not be as effective, whereas using the inter-event patterns of actions taken by the adversary may reveal valuable about the attacker and help attributing attacks to him.

7. Conclusion

Motivated 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 runtime to characterize malware. At its core, Chatter considers the order in which behavioral events occur. We notice that order-based features can be captured and analyzed using the n-gram technique widely used in document classification. With its many advantages enumerated in Section 4, and using three malware families, Chatter is shown to be reasonably accurate at classifying malware samples into their respective families.

We considered order-based behavioral features for classification of malware samples in its simplest form. Addressing the limitations outlined in Section 6.2 is the immediate future work. In particular, we would like to explore partial-order features for fingerprinting malware samples. Those features would address noised features (both intentional obfuscation by malware authors, and unintentional, due to mixed signals on-the-wire). Realizing the applications listed in Section 6.3 using the Chatter methodology is another future work that we would like to explore.

Acknowledgements. An earlier version appeared in IEEE CNS [24]. Part of this work was done while A. Mohaisen and O. Alrawi were at Verisign Labs. This work was supported in part by the Global Research Lab. (GRL) Program of the National Research Foundation (NRF) funded by Ministry of Science, ICT (Information and Communication Technologies) and Future Planning (NRF-2016K1A1A2912757). This work was also partly funded by Chung-Ang University Research Grant (2018).

References

[1] Halliday, J. (2013), Hackers attack european governments using 'miniduke' malware, http://bit.ly/16bVldV.

[2] New York Times (2013), Nissan is latest company to get hacked, http://nyti.ms/Jm52zb.

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

[4] Wu, Z., Xu, Z. and Wang, H. (2012) Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security symposium*: 159–173.

[5] Wang, A., Mohaisen, A., Chang, W. and Chen, S. (2015) Delving into internet ddos attacks by botnets: Characterization and analysis. In *IEEE International Conference on Dependable Systems and Networks (DSN)*.

[6] Li, F., Lai, A. and Ddl, D. (2011) Evidence of advanced persistent threat: A case study of malware for political espionage. In *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on* (IEEE): 102–109.

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

[8] Mohaisen, A. and Alrawi, O. (2014) Av-meter: An evaluation of antivirus scans and labels. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA 2014, Egham, UK, July 10-11, 2014. Proceedings* (Springer), *LNCS* **8550**: 112–131.

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

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

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

[12] Kong, D. and Yan, G. (2013) Discriminant malware distance learning on structural information for automated malware classification,. In *ACM KDD*.

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

[14] Kampanakis, P. (2014) Security automation and threat information-sharing options. *IEEE Security & Privacy* **12**(5): 42–51.

[15] Park, J., Alasmary, H., Al-Ibrahim, O., Kamhoua, C., Kwiat, K., Njilla, L. and Mohaisen, A. (2018) Qoi: Assessing participation in threat information sharing. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[16] Tosh, D., Sengupta, S., Kamhoua, C., Kwiat, K. and Martin, A. (2015) An evolutionary game-theoretic framework for cyber-threat information sharing. In *Proceedings of the IEEE International Conference on Communications (ICC)* (IEEE).

[17] Mohaisen, A., Alrawi, O. and Mohaisen, M. (2015) Behavior-based automated malware analysis and classification. *Elsevier Computers & Security* .

[18] Dinaburg, A., Royal, P., Sharif, M.I. and Lee, W. (2008) Ether: malware analysis via hardware virtualization extensions. In *ACM CCS*.

[19] Mariconti, E., Onwuzurike, L., Andriotis, P., Cristofaro, E.D., Ross, G.J. and Stringhini, G. (2017) Mamadroid: Detecting android malware by building markov chains of behavioral models. In *NDSS*.

[20] Aafer, Y., Du, W. and Yin, H. (2013) Droidapiminer: Mining api-level features for robust malware detection in android. In *International Conference on Security and Privacy in Communication Systems* (Springer): 86–103.

[21] Jang, J., Brumley, D. and Venkataraman, S. (2011) Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM conference on Computer and communications security* (ACM): 309–320.

[22] Rossow, C., Dietrich, C.J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H. *et al.* (2012) Prudent practices for designing malware experiments: Status quo and outlook. In *IEEE Sec. and Privacy*.

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

[24] Mohaisen, A., West, A.G., Mankin, A. and Alrawi, O. (2014) Chatter: Classifying malware families using system event ordering. In *IEEE Conference on Communications and Network Security, CNS 2014*: 283–291.

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

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

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

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

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

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

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

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

[33] West, A.G. and Mohaisen, A. (2014) Metadata-driven threat classification of network endpoints appearing in malware. In *DIMVA*: 152–171.

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

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

[36] Berlin, K., Slater, D. and Saxe, J. (2015) Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, AISec 2015, Denver, Colorado, USA, October 16, 2015*: 35–44.

[37] Anderson, B. and McGrew, D.A. (2016) Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2016, Vienna, Austria, October 28, 2016*: 35–46.

[38] Firdausi, I., Lim, C., Erwin, A. and Nugroho, A.S. (2010) Analysis of machine learning techniques used in behavior-based malware detection. In *Proceedings of the International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT)* (IEEE).

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

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

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

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

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

[44] Wang, W., Zhu, M., Zeng, X., Ye, X. and Sheng, Y. (2017) Malware traffic classification using convolutional neural network for representation learning. In *International Conference on Information Networking (ICOIN)* (IEEE).

[45] Spaulding, J., Park, J., Kim, J., Nyang, D. and Mohaisen, A. Thriving on chaos: Proactive detection of command and control domains in internet of things-scale botnets using drift. *Transactions on Emerging Telecommunications Technologies* : e3505.

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

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

[48] Skrzewski, M. (2011) Flow based algorithm for malware traffic detection. In *International Conference on Computer Networks* (Springer).

[49] Spaulding, J., Park, J., Kim, J. and Mohaisen, A. (2018) Proactive detection of algorithmically generated malicious domains. In *Proceedings of the IEEE International Conference on Information Networking (ICOIN)*.

[50] Spaulding, J., Nyang, D. and Mohaisen, A. (2017) Understanding the effectiveness of typosquatting techniques. In *Proceedings of the ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*.

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

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

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

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

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

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

[57] Egele, M., Scholte, T., Kirda, E. and Kruegel, C. (2008) A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.* **44**(2): 6:1–6:42. doi: 10.1145/2089125.2089126, URL http://doi.acm.org/10.1145/2089125.2089126.

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

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

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

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

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

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

[64] Shen, F., Del Vecchio, J., Mohaisen, A., Ko, S.Y. and Ziarek, L. (2017) Android malware detection using complex-flows. In *ICDCS*.

[65] Mekky, H., Mohaisen, A. and Zhang, Z.L. (2015) Separation of benign and malicious network events for accurate malware family classification. In *IEEE CNS*.

[66] — (2017), Detecting Encrypted Malware Traffic (Without Decryption), https://bit.ly/2n2JLVt.

[67] — (2017), SSL/TLS-based malware attacks, https://www.zscaler.com/blogs/research/ssltls-based-malware-attacks.

[68] Efros, A.A. and Leung, T.K. (1999) Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* (IEEE), **2**: 1033–1038.

[69] Mohaisen, A., Alrawi, O. and Larson, M. (2013) *AMAL: High-Fidelity, Behavior-based Automated Malware Analysis and Classification*. Tech. rep., Verisign Labs.

[70] Alpaydin, E. (2004) *Introduction to machine learning* (MIT press).

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

[72] — (2013), Yara Project: A malware identification and classification tool, http://bit.ly/3hbs3d.

[73] Arbor Networks (2010), Another family of DDoS bots: Avzhan, http://bit.ly/IJ7yCz.

[74] Damballa (2010), The IMDDOS Botnet: Discovery and Analysis, http://bit.ly/1dRi2yi.

[75] DDoSpedia (2013), Darkness (Optima), http://bit.ly/1eR40Jc.

[76] Jose Nazario (2007), BlackEnergy DDoS Bot Analysis, http://bit.ly/1bidVYB.

[77] Arbor Networks (2011), JKDDOS: DDoS bot with an interest in the mining industry?, http://bit.ly/18juHoS.

[78] Malware Intel. (2010), n0ise Bot. Crimeware particular purpose for DDoS attacks, http://bit.ly/1kd24Mg.

[79] mcafee.com (2011), Revealed: Operation Shady RAT, http://bit.ly/IJ9fQG.

[80] Kelly Jackson Higgins (2013), Dropbox, WordPress Used As Cloud Cover In New APT Attacks, http://ubm.io/1cYMOQS.

[81] Trend Micro (2011), Trend Micro Exposes LURID APT, http://bit.ly/18mX82e.

[82] — (2012), Sykipot is back, http://www.alienvault.com/open-threat-exchange/blog/sykipot-is-back.

[83] — (2011), ZeroAccess, http://bit.ly/IPxi0N.

[84] Sophos (2014), Volume of malware threatens security, http://bit.ly/17UVQ19.