October, 2014

Chatter: Classifying Malware Families Using System Event Ordering

Aziz Mohaisen
Andrew G. West
Allison Mankin
Omar Alrawi

Chatter: Classifying Malware Families Using System Event Ordering

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 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, use of these fine-granularity data points makes these techniquse computationally expensive. Moreover, the signatures and heuristics this analysis produces are often 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. In its prototype form only network events are considered 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 (with an accuracy of roughly 95%).

I. INTRODUCTION

Malware analysis and classification is an important problem given the rapid growth of malware attacks targeting all types of users: personal, enterprise, and government [1]. While the binary task of distinguishing malware from benign executables is well-known, being able to reliably identify the "strain" or "family" to which malcode belongs is an equally important task, especially in operational and industrial settings [2]. For example, family identification helps orchestrate mitigation, assessment of damage, and reinforcement of disinfection mechanisms. Additionally, discovering longitudinal trends in malware strains can enable researchers to concentrate their efforts on understudied or emergent families [3].

Techniques for characterizing malware samples are of two types: signature-based [4], [5] and behavior-based [6]–[8]. Signature-based techniques rely on known sets of patterns obtained by reverse engineering and manually inspecting malware samples. Accordingly, these techniques require many specialized man-hours of analysis. Signature-based methods are computationally cheap in that they pattern match over static binaries, but they are also easily circumvented by polymorphic obfuscation, packing, and code rearranging. Conversely, behavior-based techniques use run-time execution artifacts to extract features. While computationally expensive, such approaches are more effective than signature-based ones as they are agnostic to the underlying code and any obfuscation [8].

Manually inspecting malware samples to identify strain/class/family – while certainly possible for small datasets – does not scale for real-world malware populations [3]. Machine learning has enabled the automation of malware classification [9]. When a piece of malware infects a system, the generated behavioral artifacts (often monitoring memory, file system, registry, and/or network interfaces) contain a wealth of features that can be used to "fingerprint" that malware. The accuracy of machine learning algorithms rely on three factors: algorithms, ground truth, and feature selection [10]. Many algorithms are well-understood for their benefits/shortcomings and their application is straightforward. On the other hand, both ground truth and feature selection are challenges more specific to the problem at hand [8].

Obtaining ground-truth from blackbox-like sources – including antivirus scans – has been shown insufficient [3], [11]. With malware's growing complexity the limitations of signature-based techniques utilized by major antivirus vendors are significant. Moreover, these signatures and their output tend to be inconsistent across multiple vendors. To the best of our knowledge, no study in the literature relies on labels other than those provided by antivirus vendors for ground truth, despite the common belief that those labels are unreliable.

Determining the correct abstraction level at which to derive features is equally important [10], [12]. Features that accurately represent the malware family are an important and defining criterion for high-fidelity malware classification. Furthermore, the level of complexity needed to deploy associated systems 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 costs in online detection systems [13]. This is, 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 important, the degree of invasiveness is also a significant consideration.

In this paper we address both issues by introducing CHATTER, a "less-invasive" behavior-based system for collecting run-time artifacts and performing feature derivation/analysis. CHATTER relies on the order and frequency with which malware samples generate behavioral artifacts. 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; man-hours not needed during online operation. To this end, the contributions of this paper are as follows:

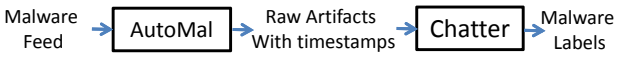


Fig. 1. Flow diagram of CHATTER and its use of AUTOMAL

- We introduce CHATTER, a system for malware analysis and classification based on cheap order-based behavioral features. CHATTER is less invasive than existing systems and we argue for various deployment scenarios, addressing operational needs.
- We demonstrate the operation of CHATTER over three malware families using only the network interface. 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 related literature.

The use of n-grams in the context of malware classification is well established. However, *a novel and major component 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. [14] 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.

The organization of this paper is as follows. In Section II we review the design of CHATTER. Section III evaluates CHATTER on three real-world datasets of different malware families. In Section IV we discuss several aspects of CHATTER and its operation before reviewing related work in Section V. Concluding remarks are made in Section VI.

II. 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.

A. Design Goals and Requirements

CHATTER was designed with the following goals in mind:

- *Cost-effectiveness:* Feature extraction should be computationally inexpensive, especially in online operation. We emphasize that CHATTER ignores solutions requiring deep analysis of a large number of artifacts as in prior literature (AMAL [8] is an example – among many other systems [7], [11] where the whole image of a 10GB hard drive and 256MB of RAM are analyzed to extract features).
- *Less-invasiveness:* While possible to analyze malware in instrumented and virtualized environments, 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 trained and/or 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 IV-C we show two such applications that benefit from such generalized capabilities.
- *Flexible 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.
- *Accuracy:* The system should aim to provide the greatest coverage while minimizing false positives. We strive for operationally acceptable accuracy.

With CHATTER's design goals in mind. we now show how they are achieved as we walk through the system design and subsequently, its evaluation.

B. System Workflow

CHATTER's overall workflow is visualized in Fig. 1. 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 n-gram techniques and the machine learning algorithms which are applied.

1) Sandboxed Execution: While any sandboxed execution environment (or bare metal execution) can be used for extracting the features used by CHATTER, we use a proprietary system named AUTOMAL [8]. 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. AUTOMAL is described in greater depth in [8], but to summarize 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 backend storage unit.

While artifacts from all interfaces are interesting, we make use of only network features herein. This is done in order to concentrate on the novelty of CHATTER's technique and because network behaviors can be monitored external to local hosts (see Section IV). Towards eventual expansion into other interfaces, the CHATTER model should be interpreted as a blackbox that takes the raw-artifacts with timestamps and generates labels. A detailed diagram of CHATTER is shown in Fig. 2 which we follow moving forward.

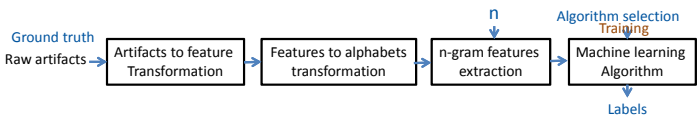


Fig. 2. Flow diagram of CHATTER

2) Behavioral Documents: As input CHATTER takes both raw artifacts generated by AUTOMAL and the family label applied to the associated malware sample. The raw artifacts generated by AUTOMAL are termed the *behavioral profile* of the malware sample. Crucially, the behavioral profile maintains the notion of *order*. CHATTER then processes the profile into features. In the case of network features AUTOMAL outputs an ordered PCAP file which is then parsed for relevant events. For example, A DNS query of type MX on port 53 would flag events like *dns_query*, *port_53*, and *mx_query*.

These events are then mapped onto an alphabet. In this phase, CHATTER assigns a character(s) from a pre-determined alphabet to each event. This allows each behavioral profile (i.e., the PCAP) to be represented as a *document* by concatenating the character(s) associated with network events. Abstracting the behavioral profile into a document in this way simplifies the application of "text mining" and classification techniques.

3) Utilizing n-grams: Given a value for the parameter n and the behavioral document, CHATTER enumerates all unique n-grams in that document. CHATTER *counts the number of times each n-gram occurs in a document; these counts are the most important feature(s) moving forward*. These counts are not represented as an exhaustive feature vector. Such a vector would be extremely lengthy given the growth of permutation operations, in turn slowing machine learning methods. Instead, a sparse representation is utilized based on those n-grams actually observed in an entire corpus of malware samples. Indeed, many of the possible n-gram sequences do not even represent legal network sequences of events, generating considerable spatial and temporal efficiency gains.

4) Machine Learning Component: Once the n-gram feature vectors are computed for the different malware samples, they are fed into the machine learning algorithm of choice. Using a training set the machine learning model is constructed, and then validated against a test set. Further details on use-cases of CHATTER are in Section IV.

III. EVALUATION

To evaluate our proposal 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.

A. Malware Samples

For the evaluation of CHATTER we use three malware families: Zeus, Darkness, and Shady RAT (SRAT). As we will soon describe, these three 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 13 other families, including Ramnit, Bredolab, Zero Access, SillyFDC, and Virut. The systems' operation and accuracy during these larger trails was consistent for the results we present for our three-family trial.

TABLE I. MALWARE FAMILIES USED IN THE EVALUATION OF CHATTER, INCLUDING SET SIZE AND THE AVERAGE NUMBER OF EVENTS PER EXECUTION TRACE (FURTHER DETAILS ARE IN §III-A).

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

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 I reports on the quantity of samples we virtualized and the average document length that resulted. To foster transparency and reproducibility of results, we intend to release the dataset used in this work to the larger community.

For every malware family set we also have an equally-sized sample of random malware samples drawn from an expansive malware repository. For example, our evaluation has 1025 Zeus samples. Thus when testing our model we also include 1025 non-Zeus samples consisting of many different malware families. 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: Zeus [15] is a well-known banking Trojan that is used by cyber criminals to recruit botnet members for purposes of stealing money, credentials, and system resources from the infected victims and their machines. 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 and further instructions. Zeus also periodically sends out stolen data to a "drop site" that is noted in the configuration file. These network artifacts can help network administrators identify possibly Zeus infected hosts in their environment. These network artifacts can also be used by CHATTER to identify Zeus network traffic.

Darkness: The second malware family utilized is the distributed denial of service (DDoS) bot known as *Darkness* or *Optima* (we prefer "Darkness" moving forward). 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 communicating 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, Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) flood attacks. The HTTP flood attack varies user-agent strings for each request making it hard to identify DDoS traffic. The network artifacts generated by the Darkness are characterized on a network due to their high volume, making it a good feature to use for classifying the DDoS bot using machine learning algorithms.

Shady RAT (SRAT): The final malware family considered is "targeted" malware that McAfee terms Shady RAT [16]

TABLE II. 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

(we will refer to it as "SRAT"). The malware targets several high profile organization and government entities to steal intellectual property and sensitive data. SRAT infects systems via spear phishing email using social engineering techniques. After infection. communication starts with the C&C server. The communication is sometimes done by downloading an HTML page and parsing out HTML comments which contain encrypted commands. Another method involves the use of stenography to decrypt commands from images embedded in a web page. This communication channel is very hard to detect by a network administrator because it blends in with regular traffic flowing through the network.

B. Ground-truth

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 [7], [11], [13] 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 [17] are sometimes applied to weed out irrelevant samples and a system called AMAL [8] 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.

Critics may see this labeling methodology as an expensive and constraining factor in the operation of CHATTER. In Section IV-B we argue that certain businesses have a continuous need for such manual inspections, and reliable family labels are a cheap by-product of this process.

TABLE III. 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

C. Feature Space and Its Reduction

Running a malware sample in a sandboxed environment results in many artifacts and large PCAP files. Not all of these are relevant nor meaningful in identifying a malware sample. Accordingly, we rely on expert *domain knowledge* to identify the network-related events of interest. To this end, Table II highlights some of the 26 network-related events used in our analysis. Per Table III observe that some events do not occur for some families ($n = 1$) and far greater unique combinations of events never occur (where $n > 1$ there is no permutation-scale growth). This makes empirical the advantages of using that sparse feature representation we described earlier. For example, when $n = 8$ our feature vector needs only $\approx$ 3800 entries, not the 2.08^{11} entries needed for exhaustive representation.

CHATTER uses the full feature vector in its analysis. While algorithms exist to reduce the feature space, we avoid these for two reasons: (1) The goal of CHATTER is to enable the use of new features rather than testing which subset of them performs best, and (2) Reducing the number of features using off-the-shelf algorithms like recursive features selection (RFS) or principle component analysis (PCA) is orthogonal to this work. Our informal experiments with these techniques suggest little improvement in scalability or degradation of performance. This result is in line with a large body of literature on the problem [7], [8], [12], [15], [18].

D. Evaluation Metrics and Learning

To evaluate CHATTER, we use evaluation metrics widely used in the literature [7], [8]: the accuracy, precision, recall, and F1 score. These are explained in the appendix. For all experiments 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 among the k possible folds. At the end, the result is computed as the average over the k runs. We set $k = 10$ due to its common use.

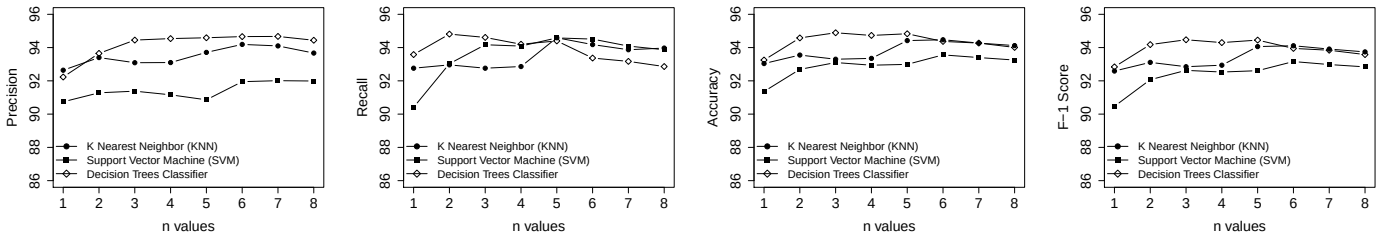
Three machine learning algorithms are used in our evaluation: the k-nearest neighbor (k-NN), support vector machine (SVM), and decision tree classifiers. All three algorithms are intended for binary supervised learning and are capable of identifying the membership of a malware sample into one of two classes. More details on those algorithms and their operation are in the appendix.

E. 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.

TABLE IV. 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



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

Fig. 3. Performance measures for the Zeus malware family with network artifact classification using CHATTER atop a baseline classifier

Results in isolation: Table IV shows performance for selected values ($n = 1, 4, 8$) across all families and algorithms using only order-based features. 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 $n = 4$ tends to produce noticeable performance increases, the performance ramifications of the next increase to $n = 8$ were more mixed. Between families we observe that "Darkness" malware was the most difficult to computationally identify, while performance for "SRAT" models 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, it is also desirable to see if it can contribute when paired with a robust baseline classifier. Figs. 3, 4, and 5 show this result. Here the baseline classifier consists primarily of filesystem features. Comparing the results presented in Table IV with those computed over the same dataset in [15] (without the contribution of CHATTER), 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.

IV. DISCUSSION

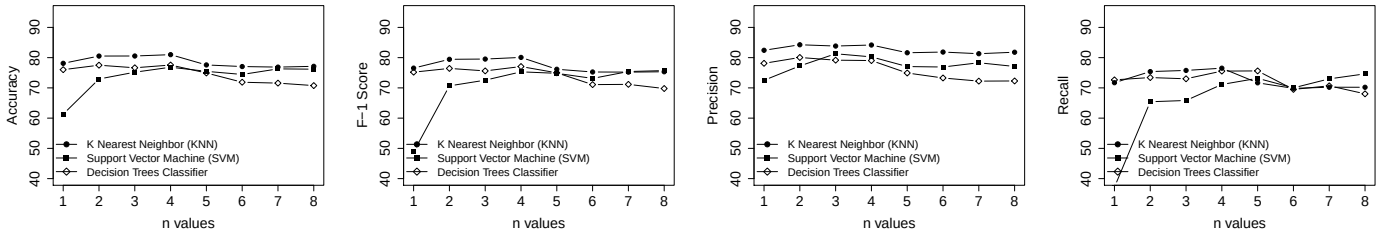
In this section we highlight interesting findings and observations regarding CHATTER. This includes how the proposed system meets its defined requirements and acknowledges weaknesses fo the proposal. One of the foremost questions is the role of gram-length n in performance. When $n = 1$ ordering plays 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 $n = 4$ or $n = 5$. This observation is further validated when combining baseline and ordered network features. The accuracy graphs in Figs. 3, 4, and 5 show a downward trend as n takes on values larger than 5. We conclude that order does improve classification by about 10% to 12%.

We note that the classification for the Darkness/DDoS malware family was less accurate than with other families, even when considering baseline features. This speaks to the way Darkness infects a host machine and uses it as a bot. Darkness creates no files (which would trigger many of our baseline features) and does most of its characteristic manipulations inside the registry. While possible to do order-based evaluation of registry events, having the access to monitor a host registry in real-time presents privacy and complexity hurdles which are not present in a network-only system.

A. Meeting Design Requirements

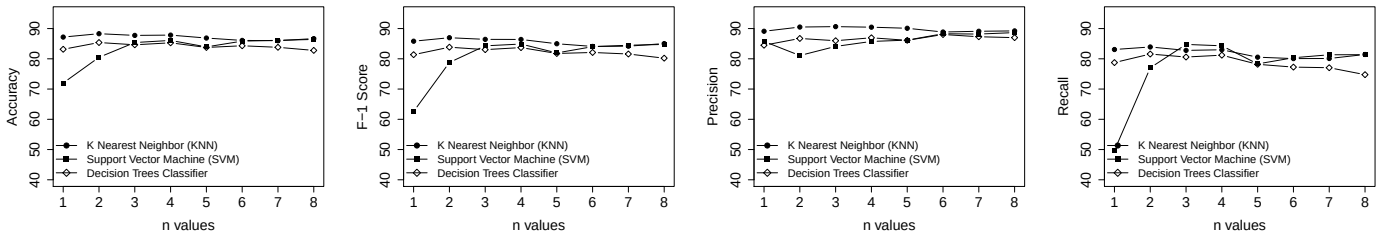
In the following we highlight how CHATTER meets the design requirements outlined in Section II-A:

- *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 [8] 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 [19].



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

Fig. 4. Performance measures for the Darkness DDoS malware family with network artifact classification using CHATTER atop a baseline classifier



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

Fig. 5. Performance measures for the SRAT malware family with network artifact classification using CHATTER atop a baseline classifier

- *Less-invasiveness*: Dependency on network features also makes CHATTER less invasive. The network events of interest 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 [8]. However, we still contend that our performance is *operationally acceptable*. The contexts in which one needs to make a family classification are very different then 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.

B. System Limitations

Discussion next acknowledges our proposal's shortcomings and examines how a knowledgeable attacker could utilize gamesmanship to circumvent our classification strategy.

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 manipulate machine learning algorithms. However, unlike systems that make use of exact matching of behavior profiles, CHATTER provides some flexibly in the grouping patterns based on n-gram features. The problem is a generic one which we address in two ways:

- 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.
- 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 concerning those events as they happen in their partial order by filtering out the noise between them. While this might seem to require 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:

- AUTOMAL, the sandbox environment for CHATTER generates patches to deceive malware samples by providing registry values indicating execution is occurring on bare metal.
- 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 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.

C. 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.

V. RELATED WORK

The literature is rich with work on malware analysis and classification [6], [11], [20]–[28]. 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 [6], [21], [22], [29]. 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 research in each of those classes.

Machine learning for malware. The use of machine learning techniques to automate classification of behavior of codes and network traffic have been thoroughly studied in the literature. Readers can refer to recent surveys in [9] and [13].

Behavior-based Analysis. The work of Bailey et al. in [11] has motivated many of the related works on behavior-based malware classification. In [6], [21], the authors use similar techniques for extracting features and leverage SVM for classifying malware samples. Our work distinguishes itself in two respects. First, although we share similarity with their high level grouping of 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 AV-returned labels.

Memory signature and reverse engineering. While we do not describe the use memory signatures in the CHATTER methodology, great potential can be seen in such a direction. Accordingly, Willems et al. introduced CWXDetector [30] 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. [31] introduced Inspector, which is used in automatically reverse engineering and highlighting code sections responsible for "interesting" behavior. Related to that, Sharif et al. [32] proposed understanding code-level behavior by reverse-engineering code emulators. It is noteworthy that this and the previous work do not generate malware artifacts other than memory-related signatures, which by themselves have limited insight into characterizing generic samples.

Traffic analysis for malware classification. Related to our use of network features is a line of research on traffic analysis for malware and botnet detection. Such works include [33]–[37], with others paying particular attention to the use of fast flux techniques [38], [39]. Support for our use of DNS features for malware analysis come in the form of [18], [40], [41]. None of those studies are concerned by behavior-based analysis beyond the use of remotely collected network features for inferring malicious activities and intent. Our system operates at network interface granularity to extract malware intelligence.

Evasion detection. Also related to our work are systems for overcoming malware evasion/obfuscation techniques. In [42], K-Tracer is introduced for extracting kernel malware behavior and mitigating the circumvention of loggers deployed in the kernel by rootkits. In [43], 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 [44]. A nicely written survey on such systems and tools is found in [45].

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 [21]) or streams of communication traffic [46]), but not higher-level **sequence of events** occuring while executing a malware sample. Other examples of low-level granularity attempts can be found in [43], [46]–[48]. Of particular interest is the concurrent work in [46], which derives n-gram network features for purposes of intrusion detection. Using network artifacts for identification of malicious activities, like botnets, is investigated in [35]–[37], [49], [50]. Further applications of characterizing malicious domain names using network traffic and artifacts (DNS queries, among others) are reported in [18], [26], [51].

Event ordering The basic idea of using event order to characterize processes was first explored by Forrest et al. in their seminal work [14]. 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 subsequences (as in n-grams) and their frequency.

VI. 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 II, and using three malware families, CHATTER is shown to be reasonably accurate at classifying malware samples into their respective families.

Future work. This paper considered order-based behavioral features for classification of malware samples in its simplest form. Addressing the limitations outlined in Section IV-B is pressing 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 IV-C using theCHATTER methodology is another future work that we would like to explore.

Acknowledgements

A shorter version of this work appeared as a poster in the *Proceedings of IEEE CNS 2013* [52]. We would like to thank the iDefense team at VeriSign for providing data and technical support, and the reviewers for their valuable comments.

References

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

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

[3] 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.

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

[5] 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.

[6] 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*, 2008, pp. 108–125.

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

[8] A. Mohaisen, O. Alrawi, and M. Larson, "Amal: High-fidelity, behavior-based automated malware analysis and classification," Verisign Labs, Tech. Rep., 2013.

[9] 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.

[10] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification,," in *ACM KDD*, 2013.

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

[12] 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.

[13] 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.

[14] 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*. IEEE, 1996, pp. 120–128.

[15] A. Mohaisen and O. Alrawi, "Unveiling zeus: automated classification of malware samples," in *WWW (Companion Volume)*, 2013, pp. 829–832.

[16] D. Alperovitch, "Revealed: Operation shady rat."

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

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

[19] Sophos, "Volume of malware threatens security," http://bit.ly/17UVQ19, Mar 2014.

[20] 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.

[21] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

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

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

[24] J. Kinable and O. Kostakis, "Malware classification based on call graph clustering," *Journal in computer virology*, vol. 7, no. 4, pp. 233–245, 2011.

[25] M. Ramilli and M. Bishop, "Multi-stage delivery of malware," in *MALWARE*, 2010.

[26] 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.

[27] 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.

[28] A. G. West and A. Mohaisen, "Metadata-driven threat classification of network endpoints appearing in malware," in *DIMVA*, 2014, pp. 152–171.

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

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

[31] 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.

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

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

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

[35] 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.

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

[37] 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.

[38] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *NDSS*, 2008.
[39] J. Nazario and T. Holz, "As the net churns: Fast-flux botnet observations," in *MALWARE*, 2008, pp. 24–31.
[40] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for dns," in *USENIX Sec. Symposium*, 2010.
[41] 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.
[42] A. Lanzi, M. I. Sharif, and W. Lee, "K-tracer: A system for extracting kernel malware behavior," in *NDSS*, 2009.
[43] R. Perdisci, A. Lanzi, and W. Lee, "Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables," in *ACSAC*, 2008.
[44] L. Lu, V. Yegneswaran, P. Porras, and W. Lee, "Blade: an attack-agnostic approach for preventing drive-by malware infections," in *ACM CCS*, 2010, pp. 440–450.
[45] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 6:1–6:42, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/2089125.2089126
[46] 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*. ACM, 2013, pp. 67–76.
[47] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *The Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.
[48] 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*. IEEE, 2001, pp. 38–49.
[49] W. T. Strayer, D. E. Lapsley, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet Detection*, 2008.
[50] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *USENIX NSDI*, 2010.
[51] 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.
[52] A. Mohaisen, O. Alrawi, A. G. West, and A. Mankin, "Babble: Identifying malware by its dialects," in *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE, 2013, pp. 407–408.
[53] E. Alpaydin, *Introduction to machine learning*. MIT press, 2004.

APPENDIX

In the following appendix we review the machine learning techniques used in this study, and sketch the accuracy measures used in our evaluation.

MACHINE LEARNING 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:

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

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:

We utilize a single split tree for two-class 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 whether 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 the leaf. We omit further details and refer the reader to [53] for more description. There are variations of decision trees in the literature, purported to provide better results (e.g., random forests). We did not try any of those techniques, since the technique we utilized already provided reasonable results. We leave integration of such techniques as future work.

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. In the testing phase, for each sample vector $\mathbf{a}$, we provide the label most frequent among the training samples nearest to it. Given spatial limitations, we refer the reader to a textbook explanation of the technique in [53].

EVALUATION METRICS

For a binary classification problem, in which it is required to determine if a given malware sample belongs to the class of interest S, 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 four outcomes and their associated magnitudes, the precision, recall, accuracy, and F1 score are defined as Precision $= \frac{T_p}{T_p+F_p}$, Recall $= \frac{T_p}{T_p+F_n}$, Accuracy $= \frac{T_p+T_n}{T_p+T_n+F_p+F_n}$, and F1 score $= 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision}+\text{recall}}$.