Malware Traffic Detection using Tamper Resistant Features

Z. Berkay Celik*, Robert J. Walls*, Patrick McDaniel*, and Ananthram Swami†
*Department of Computer Science and Engineering
The Pennsylvania State University
Email: {zbc102,rjwalls,mcdaniel}@cse.psu.edu
†Army Research Laboratory
Email: ananthram.swami.civ@mail.mil

***Abstract*—This paper presents a framework for evaluating the transport layer feature space of malware heartbeat traffic. We utilize these features in a prototype detection system to distinguish malware traffic from traffic generated by legitimate applications. In contrast to previous work, we eliminate features at risk of producing overly optimistic detection results, detect previously unobserved anomalous behavior, and rely only on tamper-resistant features making it difficult for sophisticated malware to avoid detection. Further, we characterize the evolution of malware evasion techniques over time by examining the behavior of 16 malware families. In particular, we highlight the difficultly of detecting malware that use traffic-shaping techniques to mimic legitimate traffic.**

I. Introduction and Related Work

Decades of experience suggest it is infeasible to completely secure all networked hosts. In other words, it is inevitable that some machines will be compromised by malware. Given this fact, system administrators must instead rely on detection systems to identify infected machines on their networks.

However, malware detection presents its own set of challenges. First, detecting malware at the time of infection is complicated by the myriad attack vectors malicious software may exploit to infect a host. Emails, USB drives, and web-based attacks are just a few of the common mechanisms employed by malware writers to deliver their payloads. Second, once installed on the machine, the malware may carry out a variety of malicious activities such as click fraud, data exfiltration, DDoS attacks, or spam transmission. Some of these behaviors may be indistinguishable from legitimate traffic, *e.g.*, click fraud or data exfiltration [1].

A more robust approach is to detect the malware's command and control (C2) traffic. In addition to their malicious activities, infected hosts must also communicate periodically with special hosts, called C2 servers, to coordinate attacks, download updates, perform maintenance, and send keep-alive messages. In this paper, we focus on detecting such traffic.

Researchers have proposed many network-based techniques to detect or mitigate malware communication channels. Signature-based detection systems are the most common, but they are fundamentally unable to detect malware traffic not previously identified [2] (for which there exists no signature). Supervised learning approaches eliminate the need to manually create signatures, but these techniques also suffer from an inability to detect previously unobserved malware behaviors [3]. To overcome these problems, some have proposed anomaly (*i.e.*, outlier or novelty) detection methods. These methods detect behavior that deviates from the normal hosts. However, because ground-truth labeled anomalous class samples are not typically known *a priori*, it is difficult to select the most discriminating subset of the features between normal and anomalous classes [4].

Recently, malware clustering and signature generation systems have been proposed to group malware based on features extracted from malware communication channels [3], [5], [6]. While promising, existing techniques in this space suffer from several limitations. First, these systems extract features that can be spoofed by an attacker to evade detection: HTTP headers, URLs, protocol specific and payload information [5], [7]. Second, features that rely heavily on payload inspection are vulnerable to privacy issues, payload encryption, and raise performance concerns in high-speed (multi-gigabit) networks [8]. Third, simulated timing-based features may result in unrealistic or optimistic results [5], [9]. Consequently, these systems are limited to only short-lived lifetime of malware, and can be easily evaded by further versions of the malware.

With this vision, we analyze the performance of an early-stage detector based on robust tamper resistant features. We demonstrate that the detector works well despite the structural similarities between the network level behaviors of legitimate traffic and malware traffic that has been blended with normal traffic. Our goal is to analyze the subset of the malware traffic that is in a stealth state where the malware slowly/subtly generates traffic to send control, keep alive, command transfer messages, update requests, or peer list queries. These messages are required to remotely communicate and control the infected hosts. Additionally, they are used to learn the size of the infected machines by the attackers, and hosts use them to receive the query updates. In the rest of the paper, we use *heartbeat traffic* as a representative subset of malware communication packets.

We model flow-based features of the normal network activities by excluding all sharp signals that may result in optimistic detection of malware infections. We model traffic patterns by only using the first n packets (*cf.*, Section II-B) that pass through two endpoints instead of capturing all network packets, or full flow based features. Specifically, we extract reliable and tamper-resistant flow features from only packet headers to detect the suspicious C2 heartbeat traffic of a single

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

978-1-5090-0073-9/15/$31.00 ©2015 IEEE

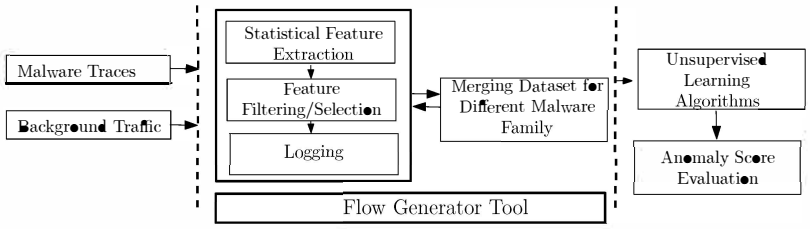


Fig. 1: Overview of framework

infected host in real time. Additionally, before blending the malware traffic with the normal traffic, unnatural heterogeneity between malware and legitimate timing based features is eliminated by sampling from the eligible background round trip times (RTT) [10]. Then, for each malware family, we test whether the anomaly detector is able to recognize an unknown attack.

Essentially, our framework is not by itself a prevention or detection technique. However, this approach gives insights into malware traffic design, and modeling legitimate applications. In addition, evaluating malware traces with legitimate applications can be used to augment existing security measures as a complementary diagnostic in malware detection systems such as [3] to keep the risk of false negatives low, port spoofing detection [11] and to identify applications for intrusion detection [12].

II. System Details

In this section, we describe our framework as shown in Figure 1. We begin by describing feature extraction and calibration of network traces, flow generation procedure, flow features extracted both from legitimate and malware traffic, deployment and adaptation of the framework. We then describe the dataset and unsupervised anomaly detection algorithms used in our experiments.

A. Threat Model

Malware C2 communication patterns have recently evolved from IRC to POP, HTTP and the communication structure has moved from centralized to more sophisticated decentralized structures such as peer-to-peer (P2P) networks where fast-flux DNS services are common. With the further evolution of various malware families, UDP is used as the main communication protocol between bots. However, a majority of malware families commonly use TCP for reliable communications. As an example, Kelihos and Zeus P2P botnets use TCP between C2 server and bots to deliver tasks, exchange information, and update binary/configuration files [13].

For the purposes of this paper, we focus on malware traffic that is either in a sleep or stealth state when malware starts discreetly generating TCP traffic during the malware initialization phase or after the installation phase has been achieved. Recall that this traffic may have multiple purposes: connect to their C2 servers to get commands, send keep alive messages or update themselves (*i.e.*, heartbeat traffic).

Our approach does not seek to detect all traffic activity of C2 channels (*e.g.*, [14]), a group of compromised hosts that generate similar traffic patterns (*e.g.*, [15]), or detect them during the exploitation phase (*e.g.*, [16]). We focus on detecting point anomalies of malware heartbeat traffic in real time by using tamper resistant features (*cf.*, Section II-B) extracted from the transport layer feature space of the network flows.

B. Feature Extraction and Selection

Our framework relies on the statistics of TCP network flows extracted from only five unidirectional or bidirectional sequence packets between two endpoints after a successful 3-way handshake is established. We have selected the number of packets as five, as most malware do not exceed 15 packets in order to be stealthy and reiterate their operations in hard-coded time intervals. Figure 2 shows the empirical cumulative distribution function of number of packet exchanges from client to server and server to client of all malware families. 85% percent of the infected machines generate less than 15 packets per flow, and the remainder shows the packet exchange during malware executable updates. Hence, we use the first five packets to extract features. These extracted features are also good candidates for real-time and early-stage modeling of the traffic behavior. The features are selected from the

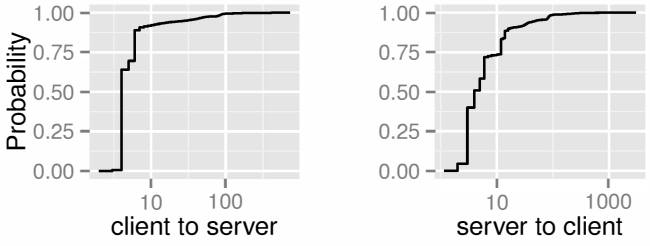
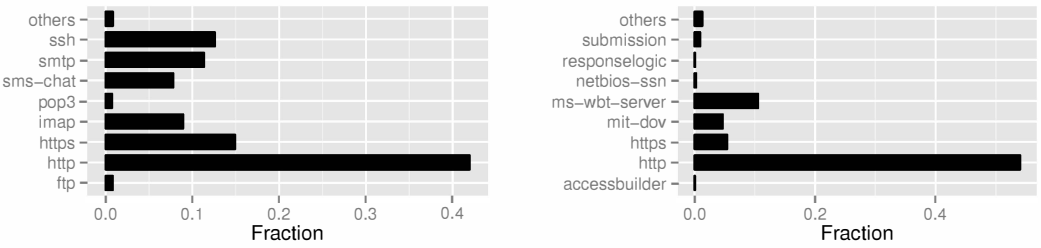


Fig. 2: Log scale plot of number of packets

248 flow-level features described in [17], [18] after applying a correlation-based filtering mechanism on normal traffic. The correlation-based filter examines the relevance of each feature as having minimal correlation between classes but being highly correlated to a specific class [19]. We have selected promising features that are not strongly dependent on TCP. In addition, the features do not include port-based, flag information (*e.g.*, count of packets with the PSH (push) flag set to 1 in the TCP option field) or payload-based analysis, which are vulnerable to the use of dynamic port numbers, encryption of payload, and masquerading techniques to avoid detection. These vulnerable features may result in optimistic detection performance. For example, using only port numbers yields accuracy similar to that obtained by using all the features in a supervised algorithm [11].

We select the following TCP based statistical flow features to model the behavior of traffic with some minor elimination of the feature space proposed in [10], [11], [18], [21] ($*$ indicates that the features are bidirectional (server to client and client to server), $+$ and $-$ represent features from client to server, server to client respectively):

- Flow duration: Difference between last packet time and first packet time.



(a) Application breakdown of legitimate traffic

(b) Port number breakdown of malware traffic

Fig. 3: Applications using server port numbers as a ground truth are determined by Internet Assigned Numbers Authority (IANA)'s list of registered ports [20]

- Count of payload (+): Count of all the packets with at least one byte of payload.
- Min data size (+): Minimum payload size observed.
- Mean of bytes (−): Data bytes divided by the total number of packets.
- Initial data length (∗): The total number of bytes sent before the first ACK packet was observed.
- RTT samples (+): Total number of RTT samples found in total packets.
- Median of bytes (+): Median byte length of packets.
- Variance of bytes (−): Variance of byte length of packets.
- IP ratio (∗): Ratio between the maximum packet size and minimum packet size.
- Goodput (∗): Total number of frame bytes divided by flow duration.

The extracted features can be defined as either static features or dynamic features. Flow duration is an example of a static feature that is not carried in the packet header over the lifetime of the flow. Dynamic features are likely to change as the flow progresses through time, and are calculated using only packet headers. Furthermore, some features can be drawn from both dynamic and static features (*e.g.*, goodput).

In addition, during the feature extraction process, we calibrate the time-sensitive features of malware traces: flow duration and goodput. Fundamentally, the calibration process includes the sampling of the eligible RTTs (both client to server and server to client) from background traffic and changing the RTTs of malware traffic to provide consistency between the timing-based features of the two traces. Otherwise, simulated or real world captured malware may show unrealistic results due to the bandwidth, connectivity, and latency differences. The process of calibration is described in [10], and ensures that synthetic features do not provide an unrealistically sharp signal when merged with background traffic especially if the background traffic is real captured traffic and malware traces are synthetically generated.

C. Adaptation and Deployment of Framework

Our framework can be located at a live network traffic point in large enterprise networks where it can observe the traffic between a large set of hosts and the Internet. More importantly, our framework can be easily adapted for flow-based systems such as NetFlow, sFlow and Jflow with given metrics of number of packets (n) and flow inactivity timeout (t) [22]. First, legitimate traffic flows are collected to model legitimate applications in the target domain with the given metrics. This process is required to learn the target domain applications, and relevant features of the network behavior in order to eliminate portability of imperfect and insufficient features between sites, as the network traces collected at different sites may involve different types of applications, and may show different network behaviors. After sufficient number of flows are aggregated from the target domain, our framework can be efficiently used in terms of storage and computational cost in high speed networks by only using the transport layer features extracted from the first n packets.

D. Dataset

For our evaluation, we have selected publicly available enterprise-like network traces collected in a controlled environment such as behind firewalls and anti-virus software. We use the traces of a small scale organization network recorded at the University of Twente with around 35 employees and over 100 students recorded from May - June 2007 [23]. We observe that the traces are good representations of non-malicious traffic where the quality is not subject to unexpected regimes due to malicious user behavior, or external changes to the network. In addition, the port numbers in the TCP headers are intact, which provides information about type of application when traces were recorded. A subset of the traces is used in our experiments, and 7753 flows are extracted. The protocol distribution of normal traffic is shown in Figure 3a. The protocols include both internal and external traffic, and this variety of traffic is a good example of day-to-day use of a campus network.

We use publicly available network traces of 21 different recent and active malware families collected from 2007 to 2014. Malware families used in our experiments can be downloaded[1] [2] [3], and most of them have been recently analyzed in experiments [24]–[26]. The analyzed malware families are used as malware-kits, spam distribution, DDoS attacks, click fraud and port scanning, and generate a vast variety of network behaviors such as exhibiting obfuscation techniques (*i.e.*, polymorphism in IPs, domains, and payloads), use of Tor-based and fast-flux networks. Our framework accepts these malware traces as a PCAP file, and the IP addresses of infected machines need to be specified to analyze the malware traffic. We extracted a total of 3600 flows of malware traffic. We observe that some malware families cannot establish a 3-way handshake, so we ignore the traffic they have generated. In addition, we are able to extract only a small number of instances from some of the malware families, because of

[1] http://mcfp.felk.cvut.cz/

[2] http://contagiodump.blogspot.co.uk/2013/04/collection-ofpcap-files-from-malware.html

[3] http://www.iscx.ca/datasets

Dataset			Method			
	Date	Number of Flows	OCSVM	k-NN	LSAD	k-Means
Agobot	2002	8	*0.7697*	**0.9779**	*0.7075*	*0.9505*
Donbot	2006	33	0.7632	*0.9979*	0.6987	**0.9983**
Kaiten	2007	49	*0.7726*	**0.7776**	*0.7141*	*0.4186*
ZeusV1	2007	50	*0.7864*	**0.8538**	*0.7207*	*0.7587*
Qbot	2008	126	*0.7994*	**0.9166**	0.7236	0.8410
Sality	2008	4	*0.7686*	**0.8567**	*0.7107*	*0.6196*
Torpig	2008	4	*0.7786*	**0.8412**	*0.7120*	*0.7611*
Neris	2009	1688	*0.8013*	**0.8337**	*0.7361*	*0.8112*
Kelihos	2010	8	0.7762	**0.9846**	0.7136	*0.9734*
Rbot	2010	806	*0.7664*	**0.8966**	0.6969	0.8304
Spyeye	2010	15	*0.7737*	*0.8183*	*0.7161*	**0.8271**
Zeroaccess	2011	363	*0.8065*	**0.8708**	0.7252	*0.7508*
ZeusGameover	2011	48	0.7347	**0.8373**	*0.6923*	*0.7769*
Tbot	2012	384	*0.8073*	*0.9131*	0.7239	**0.9161**
ZeusPonyloader	2012	8	*0.7754*	**0.8815**	*0.7144*	*0.6679*
ZeusV2	2013	6	*0.6868*	**0.7421**	*0.7239*	*0.7350*
Avg. Time			460.75	10.66	91.85	68.64

TABLE I: AUC results

unsuccessful TCP connections, and lack of available traffic. Even though these traces may not be perfect representation of variation or range of malware behavior, we include them in our experiments to evaluate the subset of their behaviors. Figure 3b shows the protocol distribution of malware traffic, which should be compared with Figure 3a for legitimate traffic. We observe that a very large fraction of legitimate and malware traffic overlaps with the HTTP(S) traffic.

E. Anomaly (Novelty) Detection Algorithms

We apply four anomaly detection algorithms that work under different assumptions based on the idea that anomalies are rare compared to the normal traffic. These algorithms are widely adapted to different domains and categorized as clustering based, density-based, SVM based, least-square cost function based. In our experiments, we use one-class support vector machine (OCSVM) with RBF (radial basis function) kernel [27], the distance to the kth nearest neighbor (k-NN), k-means clustering [4] followed by finding the distance from the test data to the nearest cluster centre, and least squares anomaly detection (LSAD) based on the least-squares probabilistic classifier [28].

These algorithms require selection of hyperparameters to be fixed before the experiments are conducted. This adds a challenge in unsupervised learning and may result in selecting different parameters for different families of malware. To solve this problem, we set the parameters of the LSAD smoothness (σ) which controls the kernel length scale, and regularization parameter (ρ) which controls the sensitivity to outliers, margin of the OCSVM (ν), the number of nearest neighbors for k-NN and k-means using a subset of the training set, and other parameters are set as default. We have implemented these algorithms in Python using the scikit-learn2 implementations (*i.e.*, k-NN, k-means, OCSVM (based on libsvm) [29] and public released LSAD python code [28]).

III. Evaluation

We now evaluate each malware family separately to observe whether their traffic is categorized as intrusive or legitimate after blending into the legitimate traffic that is represented as inliers.

A. Evaluation Metrics

We report the average AUC (Area Under Curve) value of each malware family after stratified k-fold cross validation (*i.e.*, k is selected depending on the malware traffic size), or random sampling (depending on the number of malware instances) is applied. We also use ROC curves, detection and false alarm rates to evaluate the behavior of the malware. More specifically, we use ROC curve to visualize the performance of the detector by plotting the percentage of correctly classified malicious samples (true positive rate) against the percentage of clean samples falsely classified as malicious (false negative rate), and AUC is used to summarize the detectors performance as a single number which represents the area under the ROC curve. Finally, we apply a paired t-test with significance level 0.05 to report the differences of each algorithm's AUC values.

B. Results

Table I presents the malware families, and the corresponding number of extracted flows. We report the AUC values for each anomaly detection algorithm, and the total average time required to run the experiments. We also obtain the creation date of the malware families by tracking the malware names and executables (if available) that have appeared in trusted web pages or research papers. Bold values represent the maximal AUC result of the algorithms, and italics represent that there is no significant difference between the algorithms after t-test is applied.

Our experimental results lead to the following interesting observations: First, even though we observe that many recent malware families change their main protocol to UDP, we also observe that TCP traffic is still commonly utilized (*i.e.*, mostly in HTTP(S) traffic) for malicious activities and heartbeat messages. Second, the anomaly detection algorithms are invariant to how the features of the malware families are handled *i.e.*, there is no one global algorithm that outperforms all others. As an example, Figure 4 depicts the ROC curves for all four algorithms. k-NN mostly outperforms the other algorithms, and LSAD performs more poorly than the other algorithms. However, when we compare the average total time of running experiments, LSAD outperforms OCSVM.

We next compare the ROC curves [4] of the malware families. We observe that the performance of the detectors gradually drops with the evolution of malware families as presented in Figure 4. One of the reasons for such a decline in detection is

[4]Due to the space constraints, it is not possible to present all ROC curves. We have focused on the most significant findings of detection decline of malware families over time.

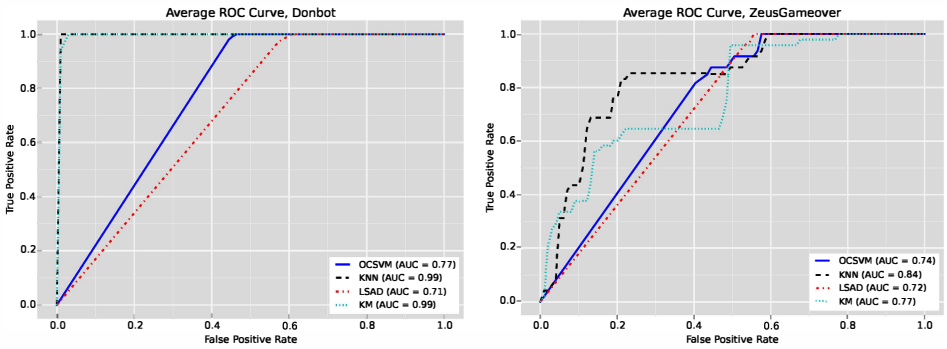


Fig. 4: ROC curves (Donbot vs. Zeus Gameover)

the number of false negatives. To better observe the major source of the false negatives, we apply the C4.5 decision tree classifier [30] by labeling each instance of the legitimate applications depending on the application class (*e.g.*, HTTP(S) traffic is labeled as web, and pop3, imap, imaps and smtp are labeled as email) by using the port numbers as the ground truth labels. The C4.5 algorithm uses normalized largest information gain to split the dataset into sub-groups, ending at the leaf nodes. We also apply subtree raising algorithm to overcome the overfitting problem. We run the experiments for each malware family separately using stratified k-fold cross validation, and average the results of the false positive and false negative counts.

	Sality	Tbot	Spyeye	ZeusV1	ZeusGameOver	ZeusP.Loader	ZeusV2
web	0 (0%)	11 (0.025%)	4 (0.009%)	8 (0.018%)	8 (0.018%)	0 (0%)	0 (0%)

TABLE II: False positive counts

	Sality	Tbot	Spyeye	ZeusV1	ZeusGameOver	ZeusP.Loader	ZeusV2
web	4 (100%)	11 (0.029%)	9 (60%)	23 (45%)	22 (45.8%)	8 (100%)	6 (100%)

TABLE III: False negative counts

Tables II and III shows the number of legitimate web applications that are classified as malware instances and the number of malware instances classified as legitimate applications respectively. Even though we have limited representation of malware C2 instances (*e.g.*, Zeus V2, and Sality), we observe that malware traffic is disguised amongst web traffic with the evaluation of Zeus botnet variants. This explains the decline of AUC results with evolution of the recent malware families. One possible approach to augment the detection would be to jointly investigate other features from HTTP requests or DNS packets with our feature space. As an example, size and sequence information of packets with more advanced probability models to find latent anomalous class [21], [31] or DNS features [32] may reduce the number of false negatives by constructing a fine-grained feature space.

We next plot the ratio of incoming and outgoing packet bytes (*i.e.*, packet size ratio) to explain how malware families change their network behavior, and disguise their patterns in HTTP traffic. Figure 5 shows the box plot of the packet ratio per flow among the most similar HTTP traffic observed between the legitimate and malware traffic. We first observe that recent malware families generate similar packet size symmetry between each other. Second, they use polymorphic network traffic to avoid detection by generating more variable traffic. As an example, Kelihos generates a high level of agreement of packet bytes, whereas Zeus variants generate

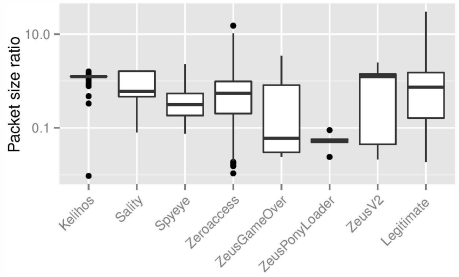


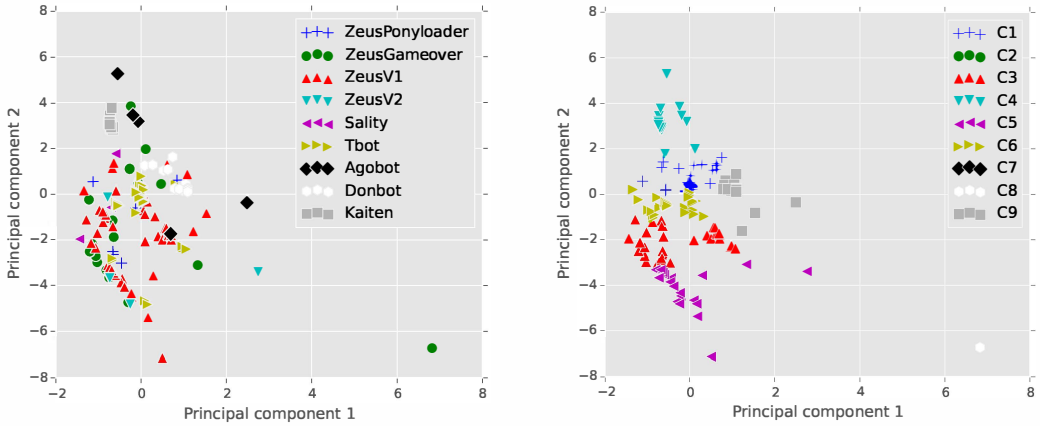
Fig. 5: Packet size ratio distribution for different malware families

variable packet bytes within the range of legitimate HTTP traffic. Similar to the reasons for false negatives, a major source of false positives is the approach used by the malware variants to mimic legitimate HTTP traffic in order to disguise their traffic.

Similarity of packet ratio between malware families may also indicate that code reuse is common in malware. We present an example to interpret the code reuse by projecting the feature space of the malware to a more interpretable two dimensional space. We apply PCA (Principal Component Analysis) [33], and select the eigenvectors corresponding to the two largest variances of the principal components. First, we transform each malware family by removing the mean value of each feature, and then divide by the standard deviation. Then, we plot the transformed feature space of malware families in two-dimensional space. Furthermore, we apply k-means clustering algorithm [34] to the projected features by selecting the number of clusters as the number of malware families, using the Euclidean distance metric. The two-dimensional scatter plot and clustering are helpful in interpreting and identifying relationships within and between malware families. In Figure 6, we observe that the malware instances such as Tbot and Kaiten are close to each other, and form a single cluster. However, Agabot is not as close as the other malware families. Zeus V1, ZeusGameover, ZeusPonyloader, ZeusV2 and Sality are in similar feature range, and most of their instances are assigned to the same clusters. The similarity observed in Figure 6 may be indicative of code reuse, or addition of new patches to previous versions.

IV. Conclusion

We presented a framework that evaluates the detection performance of malware heartbeat traffic that has been blended with traffic from legitimate applications. We demonstrate that our framework effectively discriminates most of the C2 heartbeat traffic from legitimate traffic by only using tamper



(a) Feature projection of malware

(b) k-means clustering

Fig. 6: Similarity of malware feature space

resistant features of the transport layer protocol. However, it is important to note that we observe substantial decrease in detection with the recent malware families, as malware traffic is disguised in HTTP traffic to conduct an evasion attack. Further, we also show that code reuse is common practice in malware families. We also provide a discussion of the importance of using tamper resistant feature space, and multiple sources of information to alleviate false negatives.

REFERENCES

[1] Gregoire Jacob, Ralf Hund, Christopher Kruegel, and Thorsten Holz. Jackstraws: Picking command and control connections from bot traffic. In *Proc. USENIX Security Symposium*, 2011.

[2] P. Garcia-Teodoro, J Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *In Computers & Security*, 2009.

[3] Terry Nelms, Roberto Perdisci, and Mustaque Ahamad. Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In *Proc. USENIX Security Symposium*, 2013.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *In ACM Computing Surveys*, 2009.

[5] M Zubair Rafique and Juan Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. In *Proc. Research in Attacks, Intrusions, and Defenses (RAID)*. 2013.

[6] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proc. Networked Systems Design and Implementation*, 2010.

[7] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically generating models for botnet detection. In *Proc. European Symposium on Research in Computer Security (ESORICS)*. 2009.

[8] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. An overview of ip flow-based intrusion detection. *In Communications Surveys & Tutorials*, 2010.

[9] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian. Detecting p2p botnets through network behavior analysis and machine learning. In *Proc. Privacy, Security and Trust (PST)*, 2011.

[10] Z Berkay Celik, Jayaram Raghuram, George Kesidis, and David J Miller. Salting public traces with attack traffic to test flow classifiers. In *Proc. Cyber Security Experimentation and Test (CSET)*, 2011.

[11] Guixi Zou, George Kesidis, and David J Miller. A flow classifier with tamper-resistant features and an evaluation of its portability to new domains. *Selected Areas in Communications (JSAC)*, 2011.

[12] Bryan Burns, Siying Yang, and Julien Sobrier. Identifying applications for intrusion detection systems, October 16 2012. US Patent 8,291,495.

[13] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus. In *Proc. International Conference on Malicious and Unwanted Software*, 2013.

[14] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proc. Annual Computer Security Applications Conference*, 2012.

[15] Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee, et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proc. USENIX Security Symposium*, 2008.

[16] Luca Invernizzi et al. Nazca: Detecting malware distribution in large-scale networks. In *Proc. Network and Distributed System Security Symposium (NDSS)*, 2014.

[17] Andrew Moore, Denis Zuev, and Michael Crogan. *Discriminators for use in flow-based classification*. Queen Mary and Westfield College, Department of Computer Science, 2005.

[18] Wei Li, Marco Canini, Andrew W Moore, and Raffaele Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *In Computer Networks*, 2009.

[19] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proc. International Conference on Machine Learning*, 2003.

[20] Internet Assigned Numbers Authority (IANA). http://www.iana.org/assignments/port-numbers.

[21] David J Miller, Fatih Kocak, and George Kesidis. Sequential anomaly detection in a batch with growing number of tests: Application to network intrusion detection. In *Proc. Machine Learning for Signal Processing (MLSP)*, 2012.

[22] B. Trammell B. Claise and P. Aitken. Specification of the ip flow information export (ipfix) protocol for the exchange of flow information, rfc 7011 (internet standard), internet engineering task force, 2013.

[23] Rafael Barbosa, Ramin Sadre, Aiko Pras, and Remco Meent. Simpleweb/university of twente traffic traces data repository. Technical report, Centre for Telematics and Information Technology, University of Twente, 2010.

[24] S García, M Grill, J Stiborek, and A Zunino. An empirical comparison of botnet detection methods. *In Computers & Security*, 2014.

[25] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *In Computers & Security*, 2012.

[26] Christian Rossow et al. Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets. In *Proc. Security and Privacy (SP)*, 2013.

[27] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *In Neural Computation*, 2001.

[28] John A Quinn and Masashi Sugiyama. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters*, 2014.

[29] Pedregosa et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 2011.

[30] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[31] Fatih Kocak, David J Miller, and George Kesidis. Detecting anomalous latent classes in a batch of network traffic flows. In *Proc. Information Sciences and Systems (CISS)*, 2014.

[32] Z Berkay Celik and Sema Oktug. Detection of fast-flux networks using various dns feature sets. In *Proc. Symposium on Computers and Communications (ISCC)*, 2013.

[33] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[34] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Mathematical Statistics and Probability*, 1967.