2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

Towards Malware Detection via CPU Power Consumption: Data Collection Design and Analytics

Robert A. Bridges*, Jarilyn M. Hernández Jiménez*†, Jeffrey Nichols*, Katerina Goseva-Popstojanova†, Stacy Prowell*
*Oak Ridge National Laboratory, Oak Ridge, TN, †West Virginia University, Morgantown, WV 26506
{bridgesra, nicholsja2, prowellsj}@ornl.gov, jhernan7@mix.wvu.edu, katerina.goseva@mail.wvu.edu

***Abstract*—This paper presents an experimental design and algorithm for power-based malware detection on general-purpose computers. Our design allows programmatic collection of CPU power profiles for a fixed set of non-malicious benchmarks, first running in an uninfected state and then in an infected state with malware running along with non-malicious software. To characterize power consumption profiles, we use both simple statistical and novel, sophisticated features. We propose an unsupervised, one-class anomaly detection ensemble and compare its performance with several supervised, kernel-based SVM classifiers (trained on clean and infected profiles) in detecting previously unseen malware. The anomaly detection system exhibits perfect detection when using all features across all benchmarks, with smaller false detection rate than the supervised classifiers. This paper provides a proof of concept that power-based malware detection is feasible for general-purpose computers and presents several future research steps toward that goal.**

I. Introduction & Background

Protection of networked assets from malicious software (malware) is proving insufficient, as cyber attacks regularly result in significant system failures, financial losses, or unwarranted disclosures. Signature-based malware detection methods while generally the first line of defense, are ineffective against unknown attack patterns and are unable to keep pace with the rate and sophistication of modern malware. To complement signature-based methods, previous works have explored behavior analysis by monitoring physical properties (e.g. time and power) of a device (e.g. embedded devices and mobile devices) [2–7]. The research question addressed by these works is "Can malicious software be accurately identified by monitoring some physically observable feature?". Previous work explored power consumption acquisition aimed at studying power efficiency of servers [8, 9] and mobile devices [10]. Related works exploring power consumption for malware detection have focused on embedded medical devices [2], mobile devices [11, 12], software-defined radio, PLCs, and smart grid systems [3–6].

In this paper we present an experimental setup consisting of a general-purpose computer running Windows operating system (OS) with out-of-band power monitoring hardware. We use this experimental setup to monitor and record the CPU power consumption while running non-malicious applications (only the OS, Internet Explorer (IE), and Windows Registry Editor (Regedit) [13]) both in a clean state and post malware infection. Furthermore, we propose a novel unsupervised learning approach that treats malware detection as a one-class problem based on power consumption. The proposed unsupervised detection method uses an ensemble of single-feature anomaly detectors to establish an anomaly detection method that decreases the false detection rate without sacrificing recall (true-positive rate). We compare the performance of this newly proposed anomaly detection approach with the performance of supervised detection algorithms, namely, several kernel-based support vector machines (SVMs) equipped with the same features as the anomaly detector.

This is the first research effort to attempt malware detection on a general-purpose computer via CPU power consumption monitoring. Our testbed and experimental setup are designed to collect power profiles programmatically with the computer running only non-malicious software, only malware, and the combination of both. The design and unique solutions to the data collection are research contributions.

While the use of ensembles of detectors to increase detection accuracy is not uncommon [14–16], our ensemble detector and results contribute to the anomaly detection research in a variety of ways. Specific novel contributions include: a data-driven technique to learn canonical shapes in the power curves (which permits analysis by sequential learning algorithms); application of Data Smashing [17] on two separate symbol representations of power data; formulation of a z-score, single-feature detector from permutation entropy (Cao et al. [18]) and information variance with application to power-based detection; finally, we introduce a simple "z-score of z-score" anomaly detection ensemble, and exhibit results showing it outperforms the single-feature detectors lowering false positives but raising recall.

Our results confirm, at least in this experimental setting, that (1) malware does leave a noticeable trace in the power

See extended version for more math and engineering details [1].

J.M. Hernández Jiménez led authorship and research involving related work, testbed design and implementation, rootkit analysis, and data collection.

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC, for the US Department of Energy (DOE) and US DOE, Office of Energy Efficiency and Renewable Energy, Building Technologies Office. K. Goseva-Popstojanova's work was funded in part by the NSF grant CNS-1618629.

DOI 10.1109/TrustCom/BigDataSE.2018.00250

profile, and (2) sophisticated malware can be detected using anomaly detection, without prior behavioral analysis.

II. Test Bed Design & Data Collection

Our conjecture is that malware's actions produce sufficient deviation in the power profile to admit detection. Anticipating that the effects of malware execution on power signals are subtle, significant effort was spent sensing and recording the power use data. Our experiments were run on a Dell OptiPlex 755 machine running 32-bit Windows 7. For the power data acquisition, we used a Measurement Computing Data Acquisition (DAQ) system (model USB-1608G Series with 16 channels, www.mccdaq.com). Separate channels on the DAQ were connected through a printed circuit board (PCB) sensor to separate voltage sources on the computer's motherboard power connector. The voltage and current were collected on each of the eight DC power channels. In this paper, however, we only utilized the CPU rail's data.

We developed a program that directly accesses the DAQ to gather power data. Using custom software has two advantages: (1) power data has 16 bits of precision and (2) we were able to better control the sample recording rates and vary the sample timings. As a part of the data preprocessing, the values of the voltage channel and current channel were multiplied to obtain the power consumption for each sample time. The sampling rate, computed as the median difference in power samples, was 0.0170s, with a variance in the order of 10^{-6}s.

As most malware initiates malicious traffic (e.g., click fraud, beaconing, spamming, etc.), unfiltered Internet access for the experimental machine is necessary so malware can operate with full functionality. For our experiments, an unfiltered, segregated network was created using a cellular data connection to the Internet that passed through another computer, the data repository, that was used to monitor and store the power data. More details about the hardware and software configuration can be found in our technical reports [1, 19].

Our data collection experiments consist of power profiles collection, running three benchmarks, first in a clean state and next when the machine was infected with malware. In a clean state, we first collected power data when the experimental machine was idle (with only OS and background processes but no additional software) and when two non-malicious software applications—Internet Explorer (IE) and Regedit used for dumping the Registry—were running on the system. IE was chosen because it has been proven that some malware affect the performance of browsers [20]. The Registry was chosen because malware often make modifications to it and then use a driver to hide the modified Registry keys [21].

The idle benchmark simply let the experimental machine sit idle for three minutes. In the case of the IE benchmark, 15 windows were open with a five-second delay between each, and then each window was closed. For the final benchmark, we used Regedit to print the Windows Registry to a `.reg` file. This sequence, three minutes in idle and each non-malicious software benchmark individually, was automated using a custom Python script for repeatability, and power data was collected for the three clean power profiles. Next, the experimental machine was infected with a particular malware. The Python script was executed again, collecting the power profiles of only malware running on the experimental machine (i.e., Idle benchmark), IE running in the presence of malware, and Regedit running in the presence of malware. These three power profiles were labeled infected.

In this research work we targeted rootkits, which are small malware programs that allow permanent or consistent, undetectable presence on a computer [21]. Typically, a rootkit locates and modifies software on the computer system with the purpose of masking its behavior; for example, patching, a technique that modifies the data bytes encoded in an executable code, is a type of modification that can be made [21]. Specifically, the following five rootkits were used in our experiments: Alureon [22], Pihar [23], Sirefef [24], Xpaj [25], and MaxRootkit [26]. To understand the rootkits' behaviors, we analyzed each of them using the web-based application called VxStream Sandbox [27].

The hypothesis under investigation is if malware executions (when the machine is idle or running together with selected non-malicious benchmark software) produce enough change in the power usage to admit accurate detection in the presence of the ambient noise from the OS background processes. To account for the randomness of different OS background processes running on the machine, we executed the Python script (both in clean and infected states) three times for each malware sample (3 runs $\times$ 5 rootkits), resulting in fifteen datasets, each consisting of three clean and three infected power profiles described above. Note that our approach is designed to work in a controlled environment, with preselected non-malicious benchmark applications whose power consumption profiles have been baselined in clean state.

III. Data Analysis Approach & Results

Exploring the hypothesis that malware execution necessarily changes the power profile, we formulate and test an unsupervised detector, viewing the detection task as a one-class anomaly detection problem. This has the obvious advantage that no prior observations of data collected under the presence of malware is necessary. We compare the performance of our proposed detector against several kernel SVMs, a stereotypical supervised approach, which assumes knowledge of previously observed clean and infected data. More specifically, we train and test each SVM on an equal number of clean and infected profiles, but the testing profiles are those clean and rootkit-infected profiles never seen in training. Hence, both scenarios are designed to test detection capabilities against never-seen-before malware. Both use the same set of features.

A. Features

Modeling diverse qualities of the data gives more avenues of detection; hence, we craft a variety of features modeling statistical properties of the power data sample as well as many time-varying characteristics. Each power profile (IE, Registry, idle) of a given run is transformed into a feature vector. For

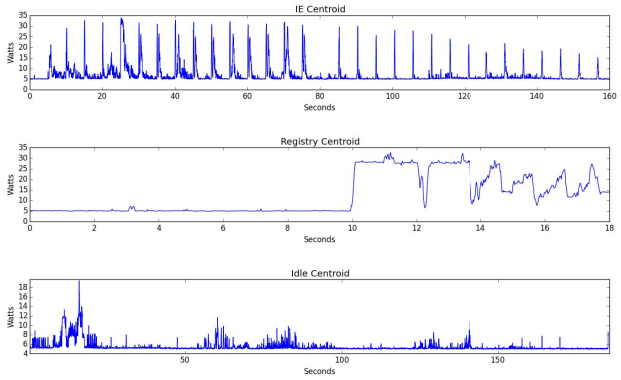

Fig. 1: (Watts vs. Seconds) Canonical profiles (baselines for L^2 error) for IE, Registry, and Idle benchmarks found by clustering unsupervised training data (ten clean profiles) with k-means clustering with L^2 distance. Gap statistic used to find the no. of clusters, k, and $k = 1$ (as expected) is found in each case. Centroids (means of each cluster) depicted.

time-dependent features, all observations in a run are shifted by their initial time, so time uniformly begins at $t = 0$. We sub-sample the power curve at intervals of 10ms to ensure uniformly spaced data. Features are described below, and see the extended version of this paper for math details [1].

1) Statistical Moments: The first four features are (empirical estimates of) the mean (first moment), variance, skewness, and kurtosis (second-fourth normalized central moments). Kurtosis $:= \sum_{i=1}^{N}(x_i - \mu)^4/(N\sigma^4) - 3$ following Fisher's definition, with 3 subtracted so normally distributed data yields 0. The moments encode characteristics of the power data regarded as a set of independent and identically distributed (IID) samples of a distribution.

2) L^2-Norm Error: Regarding each power profile as a function $p(t)$, the L^2 norm (defined as $[\int |p|^2 dt]^{1/2}$) furnishes a natural (Euclidean) distance for pairwise comparison. To construct a single baseline profile, we cluster the clean training data using k-means clustering, with k determined by the gap statistic method [28]. This provides a litmus check—we hypothesize that our clean training profiles should be similar; hence, we expect a single cluster with the centroid a canonical "baseline" function. Indeed, for all benchmarks a single canonical baseline was found by the gap statistic method, as shown in Fig. 1. For the supervised approach, the baseline is created from the 12 clean training profiles. Finally, the feature for a given profile is the L^2 distance to the baseline.

3) Permutation Entropy: Permutation Entropy, is a method of time-series analysis that detects deviations in the short-term wiggliness (roughly speaking) and has been used in various time-series detection applications including epilepsy [18] and motor bearing fault detection [29]. Our formulation follows Cao et al. [18]. From a power profile, $(x_1, \ldots, x_N)$, we extract every m-length contiguous sub-vector, $(x_{i+1}, ..., x_{i+m})$, and sort the values to obtain the resulting permutation of the indices $(1, ..., m)$. Each profile is represented as counts of the number of m-length permutations observed. Our goal is to learn how rare/likely the profile is by understanding the likelihood of its permutations. To do this, we require a probability distribution over the sample space of $m!$ permutations.

For a fixed benchmark let $\mathcal{O}$ be the observations (multiset) of permutations across all clean training profiles. For each permutation γ let $\#(\gamma, \mathcal{O})$ denote the number of times γ is observed in $\mathcal{O}$. Then we use the maximum a posteriori (MAP) estimation with uniform prior to estimate γ's probability, $P(\gamma) := (\#(\gamma, \mathcal{O}) + 1)/(|\mathcal{O}| + m!)$. This allows permutations never observed in training to have positive likelihood, albeit very small. It is clear from the MAP estimate that m must be chosen carefully, as for moderate values of m, $m!$ will easily dominate $|\mathcal{O}|$, washing out the contribution of our observations. We chose $m = 6$, which gives $m! = 720$, while $|\mathcal{O}|$ = 61,680 for IE, 6,240 for Registry, and 53,980 for Idle benchmarks. Finally, for a given profile we compute the entropy of the observed permutations as IID samples from the distribution above. After converting an observed power profile to a bag of permutations, $(\gamma_i : i = i, ..., n)$, we compute the permutation entropy, that is, the expected information of the permutation, $\hat{H} = (1/n)\sum_i -\log(P(\gamma_i))P(\gamma_i)$.

4) Data Smashing Distances: Data Smashing distance (DSD) is a computationally efficient algorithm for quantifying the distance between two sequences of symbols [17]. To apply DSD, we transform time-varying power data into a sequence of symbols that is sufficiently long. We employ DSD on two symbol representations of the power data, separately. The first transforms power samples into symbols "high" and "low" (a stereotypical application of Data Smashing), referred to hereafter as "DSD Data Distance." Given a pair of profiles, the threshold is chosen so the number of high/low values is equal to help ensure the sequences are sufficiently long.

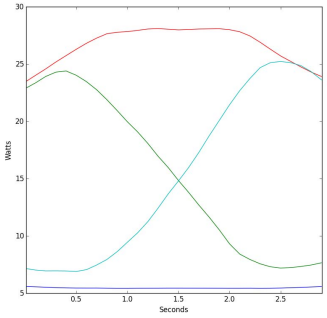

Fig. 2: (Watts vs. Seconds) Canonical shapes found by clustering three-second intervals of training data. Note that the four shapes characterize high power, low power, falling power, and rising power.

The second application seeks to model sequences of shapes in the power curves and is referred to as "DSD Shape Distance." To produce the shapes, the training profiles are cut into three-second intervals with 1.5-second overlap. Then the bag of all three-second snippets are clustered using k-means and L^2 distance as in Sec. III-A2. We manually investigated the centroids as k varied, choosing $k = 4$ as is maximal for which the centroids are not very similar curves. See Fig. 2 depicting the four canonical shapes learned from the data. Finally, given a power profile, we cut it into a sequence of three-second intervals, overlapping by 1.5 seconds, and assign to each interval the closest canonical shape. This transforms a power profile into a sequence of four "shape" symbols. Our conjecture is that profiles that exhibit a clear pattern when clean (e.g., IE baseline, top Fig. 1), will give a pattern of

shapes that may be disrupted by malware execution.

DSD uses a probabilistic algorithm to approximate the distance, and will produce similar but, in general, not identical values as the order is switched. For this reason, given two strings s_1 and s_2, we compute $d(s_1, s_2)$ and $d(s_2, s_1)$ and store the average of the two as the distance. If either distance is not returned, it indicates our strings are not long enough, and this feature is not used. In both transformations to a sequence of symbols, every sequence is concatenated with itself 100 times to ensure the strings are long enough to perform the algorithm. This can be conceptualized as a simulation of running the same benchmarks repeatedly 100 times. DSD Shape distance is not applicable to the Idle benchmark because the strings are not long enough. This is likely caused by the fact that the flat, low power shape dominates the probabilities, making the product of the four symbols' probabilities very small. To obtain both the DSD Data and DSD Shape values for a given profile, we compute the distance between its symbol sequence and that of all ten training profiles concatenated.

B. Unsupervised Approach: Anomaly Detection Ensemble

Our ensemble classifier is constructed from votes of many single-feature anomaly detectors, all obeying a principled threshold universally—any observation that is at least one standard deviation from the mean is considered anomalous, where the standard deviation and the mean are computed on the training observations. Explicitly, for a given feature, let $V = (v_i : i = 1, ..., n)$ be the training runs' values, and let μ, σ be the mean and standard deviation of V, respectively. Then given an observation of this feature, v, from either a training or testing run, we compute the z-score normalization, $z = |v-\mu|/\sigma$, and vote if $z \geq 1$. Since computation of permutation entropy features requires explicit estimation of a probability distribution over the training data permutations, we compute the mean information (permutation entropy of training data), and the standard deviation of the information over the training runs as follows, $\mu := \sum_{i=1}^{m!} -\log(P(\gamma_i))P(\gamma_i) = H(P)$ and $\sigma := [\sum_{i=1}^{m!}[\log^2(P(\gamma_i))P(\gamma_i)] - H(P)^2]^{1/2}$. For permutation entropy z-score computation, μ, and σ, as computed above from the learned distribution are used.

For the ensemble detector, we follow the same one-sigma rule using the number of single-feature votes per benchmark as the lone feature. That is, the mean and standard deviation of the training runs' vote totals are computed; each run's vote count is converted to a z-score; those runs with $z \geq 1$ are labeled infected. The proposed unsupervised detector classifies a run based on the number of votes across all three benchmarks; although we report results for each single-feature detector per benchmark and ensemble detector per benchmark as well. Table I presents the results along with each test profiles' set of z-scores and the vote thresholds.

Analyzing the unsupervised results reveals perfect detection is obtained by the overall classifier. Delving into each benchmark shows perfect detection using IE profiles alone and the Registry profiles alone, while the Idle profiles alone exhibit perfect true positive rate (TPR) with only one false positive of sixteen alerts, i.e., a 0.0625 false detection rate (FDR, defines as percentage of false alerts to alerts). Notice that the test set class bias is 25/75% clean/infected; assigning labels blindly according to this distribution produces an expected TPR of 75% and FDR of 25%. Even against this normalization, our results are overwhelmingly positive and give strong empirical evidence that the power profiles are changed sufficiently by malware executions to admit detection.

TABLE I: Unsupervised Detection Results

	TPR	FDR	TPR	FDR	TPR	FDR
Features	**Idle/Only Malware**		**IE/IE+ Malware**		**Registry/Registry+Malware**	
Mean	1.00	0.21	1.00	0.06	1.00	0.17
Variance	1.00	0.06	1.00	0.06	1.00	0.21
Skewness	1.00	0.06	1.00	0.06	1.00	0.21
Kurtosis	1.00	0.06	1.00	0.06	1.00	0.21
L^2 Error	0.87	0.19	1.00	0.00	1.00	0.12
Perm. Entropy	0.00	0.00	0.00	0.00	0.00	0.00
DSD (Data)	0.87	0.24	0.67	0.09	0.87	0.24
DSD (Shape)	N/A	N/A	0.53	0.27	0.67	0.09
BM Votes	1.00	0.06	1.00	0.00	1.00	0.00
Total Votes	TPR = **1.00**		FDR = **0.00**			

True positive rate (TPR) and false detection rate (FDR) (1 - Precision) reported for each single-feature anomaly detector, per benchmark. BM Votes row gives results for the ensemble per benchmark. Finally, Total Votes row reports the full ensemble detector (all single-feature detectors across all three benchmarks) results. Note that no single feature or ensemble for a single benchmark achieves perfect detection, but the full ensemble does.

The permutation entropy feature never deviated from the training mean by more than a standard deviation. Qualitatively this means that the patterns of variation of the profiles over $6\times$ .01s= .06s is not changed in a noticeable way by the rootkits tested. Quantitatively, this means that inclusion of permutation entropy does not affect the detection outcomes.

Notice that DSD distance, both for shape symbol sequences and on the two-symbol representation of the power data, struggle to beat the random baseline. Our observation is that this method, while mathematically exciting, is not a good detector for the detection task at hand. The distribution of the power data, regardless of the time progression (encoded by the moments), and the overall shape of the profiles (encoded by L^2-norm difference from the baseline profile) are the heavy lifters contributing to the correct classification.

C. Supervised Approach: Kernel SVMs

We compare the results of the newly proposed anomaly detection ensemble with supervised learning based on kernel SVMs. For the supervised approach, we use hold-one-out validation on the rootkits; i.e., in each fold training is performed

TABLE II: Supervised Learning Results

SVM Kernel	TP	FP	TN	FN	TPR	FDR
Linear	15	3	12	0	1.00	0.17
RBF $\gamma = 0.001$	15	3	12	0	1.00	0.17
RBF $\gamma = 0.01$	15	3	12	0	1.00	0.17
RBF $\gamma = 0.1$	14	3	12	1	0.93	0.18
Polynomial $d = 3$	15	3	12	0	1.00	0.17
Polynomial $d = 2$	15	3	12	0	1.00	0.17

on 12 of 15 clean profiles plus 12 infected profiles (3 profiles $\times$ 4 rootkits), and testing is performed on the 3 profiles from the held-out rootikit combined with the remaining three clean profiles. We used kernel SVMs as the kernels allow versatility and non-linear decision boundaries. Our implementation used SciKit-Learn [30] to test the following six kernels: linear, radial basis function (RBF) with $\gamma = 0.1, 0.01, 0.001$, and polynomial of degree two and three. The features for the learner match those of the unsupervised approach, Sec. III-A.

Table II depicts micro-averaged results, showing perfect or near perfect TPR, with 17-18% FDR. As the test sets comprised of non-biased classes (50% clean/50% infected) a random classification would expect 50% TPR and FDR. These results give further empirical evidence that the power profiles are changed in a detectable way by malware and favor the one-class detector.

IV. Conclusion

Our work proves the concept that rootkits change the power profile of the CPU in a detectable manner. By monitoring CPU power consumption under a set of non-malicious software, we programmatically gathered the power profiles in a clean, uninfected state and then in an infected state, with malware running on its own or together with the non-malicious software. To characterize the power data, a novel anomaly detection ensemble and many supervised kernel SVM classifiers are tested using features comprised of statistical moments and more sophisticated time-series-analytic features. Our results allowed analysis of each single-feature detector per benchmark, giving insight to novel applications of time-series algorithms for power-based malware detection. The overall result is that perfect detection is achievable in our test environment by the proposed one-class method, which outperformed the supervised learning with kernel SVMs that have access to labeled infected data during training.

Next-step research involve testing robustness to expected noise and variability in the applications, e.g., IE resolving different URLs. Accurate results were possible with relatively slow sampling rates (power sampled at $\mathcal{O}$100Hz implies that a 1GHz processor will exhibit $\approx 10^9/10^2 = 10$M cycles between each sample), indicating that the malware tested is not subtle when executing. Research to increase the sampling rate is necessary, especially for accurate baselining of very small, fixed instructions sequences. We believe this work is a valuable step towards a promising option for malware detection.

References

[1] R. Bridges et al., "Towards Malware Detection via CPU Power Consumption: Data Collection Design and Analytics (Extended Version)," ArXiv e-prints, May 2018. [Online]. Available: https://arxiv.org/abs/1805.06541v1

[2] S. Clark et al., "WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices," in HealthTech, 2013.

[3] C. González et al., "Power fingerprinting in SDR & CR integrity assessment," in MILCOM. IEEE, 2009, pp. 1–7.

[4] C. González et al., "Power fingerprinting in SDR integrity assessment for security and regulatory compliance," Analog Integrated Circuits and Signal Processing, vol. 69, no. 2-3, pp. 307–327, 2011.

[5] C. González et al., "Detecting malicious software execution in programmable logic controllers using power fingerprinting," in Inter. Conf. on Critical Infrastructure Protection. Springer, 2014, pp. 15–27.

[6] J. Reed et al., "Enhancing smart grid cyber security using power fingerprinting: Integrity assessment and intrusion detection," in FIIW. IEEE, 2012, pp. 1–3.

[7] J. Hernández et al., "Phase-space detection of cyber events," in Proc. 10th Ann. CISRC. ACM, 2015, p. 13.

[8] R. Koller et al., "WattApp: an application aware power meter for shared data centers," in Proc. ICAC. ACM, 2010.

[9] X. Feng et al., "Power and energy profiling of scientific applications on distributed systems," in Proc. IPDPS. IEEE, 2005.

[10] I. Polakis et al., "Powerslave: Analyzing the energy consumption of mobile antivirus software," in DIMVA, 2015.

[11] J. Hoffmann et al., "Mobile malware detection based on energy fingerprints - A dead end?" in RAID. Springer, 2013.

[12] A. Azmoodeh et al., "Detecting crypto-ransomware in IoT networks based on energy consumption footprint," JAIHC, 2017.

[13] M. Rouse, "Windows Registry Editor (regedit)," Retrieved from: http://searchenterprisedesktop.techtarget.com/definition/Windows-Registry-Editor.

[14] R. Bridges et al., "Setting the threshold for high throughput detectors: A mathematical approach for ensembles of dynamic, heterogeneous, probabilistic anomaly detectors," in Big Data. IEEE, Dec 2017, pp. 1071–1078.

[15] M. Christodorescu et al., "Can cooperative intrusion detectors challenge the base-rate fallacy?" in Malware Detection. Springer, 2007, pp. 193–209.

[16] C. Harshaw et al., "Graphprints: Towards a graph analytic method for network anomaly detection," in Proc. of the 11th CISRC. New York, NY, USA: ACM, 2016, pp. 15:1–15:4.

[17] I. Chattopadhyay et al., "Data smashing: Uncovering lurking order in data," J. Royal Soc. Interface, vol. 11, no. 101, p. 20140826, 2014.

[18] Y. Cao et al., "Detecting dynamical changes in time series using the permutation entropy," Physical Review E, vol. 70, 2004.

[19] J. Hernández et al., "Malware detection on general-purpose computers using power consumption monitoring," CoRR, 2017. [Online]. Available: http://arxiv.org/abs/1705.01977

[20] E. Rodionov et al., "The evolution of TDL: Conquering x64," Retrieved from: http://go.eset.com/resources/white-papers/The_Evolution_of_TDL.pdf.

[21] G. Hoglund et al., Rootkits Subverting the Windows Kernel. Addison Wesley, 2008.

[22] Mandiant, "Technical Report M-Trends 2015," www2.fireeye.com/rs/fireye/images/rpt-m-trends-2015.pdf.

[23] Remove-Malware-Tips, "Get rid of pihar.B Trojan," Retrieved from: https://goo.gl/UjJzd7.

[24] Symantec, "Trojan.Zeroaccess," Retrieved from: https://goo.gl/FW7xn1.

[25] B. Miller, "Got infected with rootkit.mbr.Xpaj? How to remove it?" Retrieved from: https://goo.gl/WKopjC/.

[26] "Rootkit.win32.zaccess.c," Retrieved from: https://www.enigmasoftware.com/rootkitwin32zaccessc-removal/.

[27] P. Security, "VxStream Sandbox - Automated Malware Analysis System," 2016, www.payload-security.com/.

[28] R. Tibshirani et al., "Estimating the number of clusters in a data set via the gap statistic," J. of the Royal Stat. Soc.: Ser.B (Stat. Method.), vol. 63, no. 2, pp. 411–423, 2001.

[29] Zhang, X. and others, "A novel bearing fault diagnosis model integrated permutation entropy, ensemble empirical mode decomposition and optimized SVM," Measurement, vol. 69, 2015.

[30] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," JMLR, vol. 12, pp. 2825–2830, 2011.