POSTER: Towards Architecture and OS-Independent Malware Detection via Memory Forensics

Rachel Petrik*, Berat Arik†, Jared M. Smith† §
*University of Kentucky, †University of Tennessee, Knoxville, §Oak Ridge National Lab

ABSTRACT

In this work, we take a fundamentally different approach to the problem of analyzing a device for compromises via malware; our approach is OS and instruction architecture independent and relies only on having the raw binary data extracted from the memory dump of a device. Our system leverages a multi-hundred TB dataset of both compromised host memory dumps extracted from the MalRec dataset [8] and the first known dataset of benign host memory dumps running normal, non-compromised software. After an average of 30 to 45 seconds of pre-processing on a single memory dump, our system leverages both traditional machine learning and deep learning algorithms to achieve an average of 98% accuracy of detecting a compromised host. Finally, we lay out the future work that would form a broader set of contributions, encapsulating and enhancing the work presented in this paper.

1 INTRODUCTION AND BACKGROUND

According to a recent 12-month-long study of 477 companies, the mean time to discover a breach is 197 days, with costs to remediate these breaches exceeding $3.5 million on average [5]. Having an effective Digital Forensics and Incident Response (DFIR) capability remains one of the most effective ways to both prevent and respond to modern breaches. However, with the proliferation of intrusion detection systems and security products on the market, DFIR professionals and security analysts must monitor an increasing amounts of tickets and alerts from ever-growing infrastructure. To that end, one of the most effective and widely-used methods to investigate hosts and their operating systems (OS) for breaches relies on memory forensics, or the examination of a device's volatile RAM. Among the binary data in RAM are the names and metadata of running and recently exited processes, system kernel modules and libraries, executable source code, file information, network information, registry keys, and more. Although tools such as Volatility and Rekall [3, 12] are able to successfully extract the OS-specific data and potential indicators of compromise (IOCs) from memory, they require experts to build and maintain the systems, requiring an addition of new profiles for every new OS and version. Furthermore, any artifacts found in memory must be verified by an expert with relevant OS and architecture-specific knowledge, even when these artifacts may not actually represent a compromise.

By capitalizing on the ongoing breakthroughs in deep learning [9], as well as more traditional machine learning techniques, we present a system to detect malware-compromised hosts via captured memory snapshots *alone*. By not relying on OS-specific information, we aim to reduce the amount of time an analyst must focus on a host by flagging potentially compromised hosts with only a memory snapshot as input. Most importantly, our system could be deployed in any operational environment where a memory snapshot can be ingested from connected hosts, *without* relying on endpoint IDSes that require OS and domain-specific IOCs. Systems such as Akatosh by Smith *et al.* [10] provide the ability to scalably ingest memory images from hosts, as well as others such as Endcase [2]. Our system utilizes a multi-hundred TB dataset of both compromised host memory snapshots extracted from the MalRec dataset [8] and the first known dataset of benign host memory snapshots running normal, non-compromised software. [1] After an average of 30-45 seconds of pre-processing on a single memory snapshot, our system leverages both traditional machine learning and deep learning algorithms to achieve an average of 98% accuracy of detecting a compromised host, with over 3,000 samples for our Convolutional Neural Network, and over 9,000 samples for our traditional machine learning models.

2 DATASETS

2.1 Compromised (Malicious) Snapshots

We employed the MalRec dataset from Severi *et al.* [8] created by Georgia Tech over the course of two years. With over 66,000+ malware recordings from Windows 7 32-bit machines using the QEMU virtualization software, our infrastructure extracts 1 GB memory snapshots at 0% of the QEMU recording, a random percent between 5% and 95%, and finally a snapshot at 99% of the execution of the recording. Given that the malware for a particular recording executes shortly after the beginning of the recording, we are able to use the 0% as a baseline snapshot. In practice, we believe the random percentage capture models the real-world best, where an operator cannot ensure a memory snapshot is captured after malware has completely executed in memory. Finally, these recordings contain traces of memory of normal programs, depending on what the malware calls into and passes input to; this could be PDF viewers, the command-line, browsers, and more pivot points for the malware to infect the machine.

2.2 Benign Snapshots

Next, we created a dataset of benign host memory snapshots running normal, non-compromised software, including software that executes in many of the malicious snapshots. We do not claim our dataset is completely robust, and work remains to build additional representations of "normal" state represented by volatile RAM. To generate this dataset, we leverage VirtualBox and Python to instrument a running VM. After ensuring the VM uses the same size snapshots as MalRec (1 GB), each sample generated includes the execution of between 1 and 9 programs from a list of non-malicious programs that a user might run such as Google Chrome, PDF Viewers, command-line consoles, Spotify, Slack, and Skype. Once the programs launched, we waited $20 + (5 \times n_p)$ seconds, where n_p is the number of programs ran. After this time, a benign snapshot is extracted from memory after ample time has passed for the chosen programs to open. By generating samples in parallel to the separate malicious environment, we formed our benign memory snapshot dataset. Additional work should be done to done building 'profiles' of users in order to have distinct sets of benign snapshots, centered around different demographics and job functions of users in practice. Furthermore, the time between program execution and memory snapshot collection should be varied further, while user input should be passed automatically to the executed programs to further simulate real users. Figure 1 below shows the dataset collection phase of our system, as well as highlights the next phases.

[1] Our datasets will be published upon acceptance of this paper.

Statistical Features		
Min/Max	Geometric, Harmonic, Standard Error, Arithmetic Mean	Skew, Kurtosis, Variance, Standard Deviation
Entropy	Hamming, Energy, Eucliden, Wasserstein, Bray-Curtis Minkowski* Distance	Jaccard-Needham Dissimilarity
Chi-Square Test	P-adic Valuation*	Shapiro-Wilk Test

Table 1: Total of 43 domain-unaware features extracted from memory snapshots. Starred features were extracted for several different parameters.

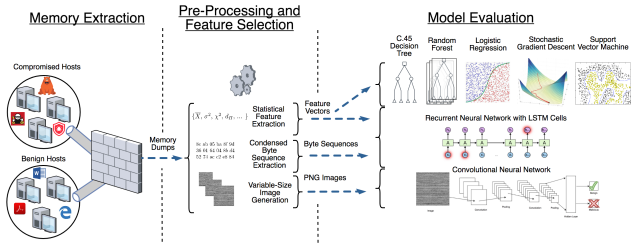

Figure 1: High-Level Architecture of the memory extraction, pre-processing and feature selection, and model evaluation.

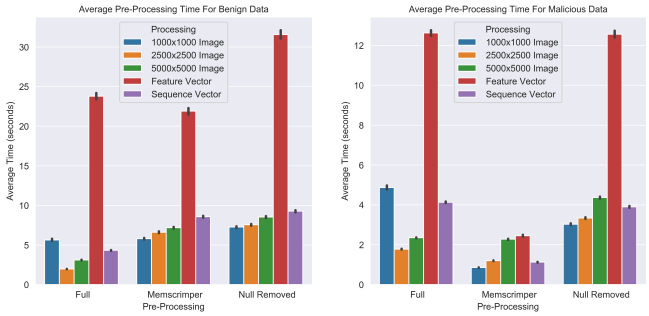

Figure 2: Average Pre-Processing Time for Benign (*left*) and Malicious (*right*) Memory Snapshots

3 PRE-PROCESSING AND FEATURE SELECTION

With the first phase of snapshot extraction completed, we investigated various approaches for representing the memory snapshots, **without relying on** domain-knowledge of the OS. Challenges arose with many aspects of this problem, particularly with how to store the data generated earlier, and more importantly, how to derive statistical and numerical features from binary data (the snapshot) that are comprehensive enough to uniquely identify each class of snapshots, benign or malicious. Beyond extracting features in vectors per sample, we generated *images* for use in a Convolutional Neural Network and *byte sequences* for a Recurrent Neural Network with LSTM cells. Overall, we attempted to rid ourselves of the problem of expert knowledge by leveraging modern machine learning to learn the data itself after our system provides adequate feature representations.

Shown in Figure 1, for every malicious or benign snapshot, we applied two additional transformations beyond using the full snapshot before generating representations: *(1)* we removed the null-bytes in the images, reducing the snapshot size by over half, and *(2)* applied a recent system called MemScrimper [1] to deduplicate the snapshots based on a reference snapshot (which is the 0% image for the malicious snapshots, and the clean state of the VM for the benign snapshots). After this, we extracted images of various sizes (1000x1000, 2500x2500, and 5000x5000) by condensing the entire binary matrix, which is done by averaging every n bytes, where n depends on the original size of the snapshot. A sample of these images is shown in architecture diagram in Figure 1. We also generated condensed raw byte sequences in a similar manner. Finally, we applied a plethora of statistical and numerical methods to each snapshot representation to extract 43 features, including features for prime-number metrics, distance metrics, statistical metrics such as variance and standard deviation, and even probability tests such as the Chi-Squared metric. These features are shown in Table 1. For all of these feature selection methods and running on a single core of a Linux-Based machine with a 2.5 GHz Intel Xeon processor, we show in Figure 2 that the average time to complete processing of one snapshot is *less than 35 seconds* for feature vector extraction. Furthermore, the average time for generating all images or extracting the byte sequences is *less than 10 seconds*. These methods were written in Python with some metrics using Numpy or Scipy; therefore, so significant speedups could potentially be gained when implemented all in C or C++.

4 TRADITIONAL MACHINE LEARNING

For our approach to detecting compromises within the snapshots using traditional machine learning, we employed a C.45 Decision Tree, a standard Logistic Regression model, a Random Forest model, a Stochastic Gradient Descent model, and a Support Vector Classifier. All machine learning methods used Scikit-Learn and an Ubuntu Linux 12-core 2.5 GHz Intel Xeon system with 32 GB of RAM. For each model, we conducted a probabilistic grid search over all available hyper-parameters to tune the model, and when testing we used cross-validation and a non-overlapping 75%/25% train/test split.

Overall, our traditional machine learning methods recorded an average of 98% accuracy of detecting a compromised host with 9,000 samples. We found the feature vectors generated from the memory snapshots that had been de-duplicated using MemScrimper performed the best across all traditional machine learning algorithms. These results were found both when comparing the benign dataset to the random, assorted percentage for the malicious memory snapshot and when comparing the benign dataset to the 99% malicious memory snapshot. Notably, the false positive rate of this particular setting was also less than 0.003% on average, indicating that our system would add only a small fraction of false alerts to an analyst's dashboard in an environment with similar hosts and active software. The summary of results for the best setting of parameters and feature vectors from de-duplicated snapshots can be found in Table 2. Due to space constraints, we leave out the results for the full and null-byte-removed snapshots for both the random or assorted percentage and the 99% snapshots, though we still found the accuracy was above 95% on average but with slightly higher false positive rates averaging 0.1%.

5 DEEP LEARNING APPROACH

Our primary deep learning approach was through the use of a Convolutional Neural Network (CNN), trained using various sized images from the different pre-processing methods outlined earlier. Our model was composed of multiple layers, including pooling and fully connected layers. We used a similar probabilistic grid search as used earlier, except allowing both the hyper-parameters and the model layers to be actively tuned in response to better performance. Under cross-validation and 75%/25% train/test split, we show our results demonstrating the

Traditional ML Algorithms: Benign vs. Assorted%, Memscrimper				
ML Algorithm	Mean Test Score (Accuracy)	Mean Fit Time (s)	Mean Score Time (s)	FPR
Decision Tree	0.995851	0.017205	0.001449	0.002667%
Logistic Regression	0.994814	0.081668	0.0247176	0.002667%
Random Forest	0.996507	0.781202	0.002633	0.002667%
SGD	0.991555	0.033968	0.012932	0.001778%
SVM	0.994518	0.125464	0.013279	0.002667%

Table 2: Results obtained from running 9,000 deduplicated memory captures through several different traditional machine learning algorithms with a train/test split of 25%.

accuracy over 50 epochs in Figure 3. For over 3,000 samples evenly balanced between benign and malicious image representations, our best CNN model finished with an accuracy of 98% when using the largest size of images generated based on a non-condensed memory snapshot. Because of our low sample size, we also include rigorous data augmentation aligned with the best practices in the Keras deep learning library, which increases the generality of the model when training. Though the full image representation indicates strong accuracy, worse performance was found for the null-byte-removed and MemScrimper representations. Regardless, our future work includes training and testing on more samples, which our experiment infrastructure continues to extract from the benign and malicious host testbeds.

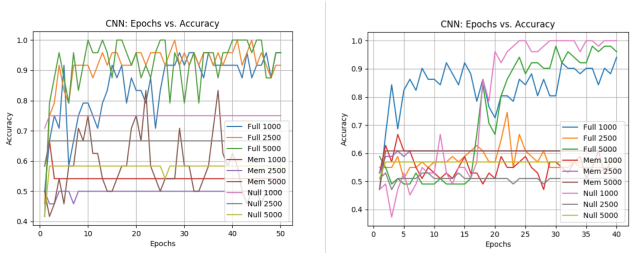

Figure 3: CNN Accuracy over 50 Epochs for Random Percent Malicious Snapshot (*left*) versus 99% Malicious Snapshot (*right*)

6 RELATED WORK

Most closely related to our analysis of memory snapshots as images is work originally by Nataraj *et al.* and Kamundala *et al.* [6, 7]. Nataraj focused on representing malware executables as images, which was then followed by extensions to the original study applying a CNN-based strategy to other forms of malware, including datasets of Android samples [4] and the traffic flows of active malware samples [13]. Our work differs significantly from these systems once we consider the *entire* memory snapshot of a device containing a malware's trail in volatile memory.

With a similar motivation and a benign dataset developed concurrently but independently of ours, the DeepMem system by Song *et al.* [11] to appear in CCS 2018 leverages graph-based deep learning to locate domain-specific data in memory snapshots. Beyond the similar datasets, our work diverges in that our approach requires no specific representation of the data in the snapshot, rather leveraging an array of statistical and numerical features independent of any inherent structure.

7 CONCLUSION AND FUTURE WORK

In this paper, we presented to the best of our knowledge, the first steps towards an OS and architecture-independent malware detection system, exploiting an array of pre-processing techniques, domain-unaware feature selection, and a suite of machine learning algorithms. We built a system for capturing memory snapshots compromised by malware, as well as a completely novel benign memory snapshot dataset, totalling several hundred TB in space before pre-processing. With our datasets, we ultimately evaluated the feasibility of our approach by testing against 3,000+ samples balanced between benign and malicious snapshots and with cross-validation. Our system achieved over 98% accuracy with low false-positive-rate for most traditional machine learning algorithms and saw similar results for a more complex, image-based CNN architecture.

Future Work: In order to truly realize the feasibility of our system in practice, several extensions must be made to our work presented here. A non-exhaustive list of next steps include: **(1)** continue to collect more samples from our distributed infrastructure for analysis. **(2)** Evaluate the performance of our models and the pre-processing time for memory snapshots larger than 1 GB. **(3)** Present results from an upcoming deployment of our models on real-world IT infrastructure at several pilot sites, which should indicate whether our combination of MalRec and our own benign dataset truly represent the state of machines in practice. **(4)** Demonstrate similar results on a dataset of compromised and benign Mac and Linux memory snapshots, though work must be done to create benign *and* compromised datasets for each, since MalRec only provides Windows data. **(5)** Include an evaluation of LSTMs and RNNs for this task, both on the binary sequences and feature vectors. **(6)** Attempt to correlate a compromised snapshot's feature vector of statistical metrics back to the **OS-specific** cause for the sample to be flagged as malicious.

REFERENCES

[1] Michael Brengel and Christian Rossow. 2018. MemScrimper - Time- and Space-Efficient Storage of Malware Sandbox Memory Dumps. *DIMVA* (2018).
[2] Endcase. 2018. Endcase Forensics. https://www.guidancesoftware.com/encase-forensic
[3] Google. 2018. Rekall Forensics. http://rekall-forensic.com
[4] TT Huang and Kao HY. 2017. R2-D2: Color-Inspired Convolutional Neural Network (CNN)-Based Android Malware Detection. https://arxiv.org/abs/1705.04448. *arxiv.org* (2017).
[5] IBM, Ponemon Institute. 2018. Cost of Data Breach Study. https://www.ibm.com/security/data-breach
[6] Espoir K Kamundala and Chang Hoon Kim. 2018. CNN Model to Classify Malware Using Image Features. *KIISE Transactions on Computing Practices* 24, 5 (2018), 256–261.
[7] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and BS Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*. ACM, 4.
[8] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. 2018. Malrec - Compact Full-Trace Malware Recording for Retrospective Deep Analysis. *DIMVA* (2018).
[9] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations (ICLR)* (2015).
[10] Jared M Smith, Elliot Greenlee, and Aaron Ferber. 2017. DEMO: Akatosh - Automated Cyber Incident Verification and Impact Analysis. *CCS* (2017), 2463–2465.
[11] Song, Wei, Yin, Heng, Liu, Chang, and Song, Dawn. 2018. DeepMem: Learning Graph Neural Network Models for Fast and Robust Memory Forensic Analysis. *ACM Conference on Computer and Communications Security (CCS)* (2018).
[12] The Volatility Foundation. 2018. Volatility Foundation. https://www.volatilityfoundation.org
[13] W Wang, M Zhu, X Zeng, and Ye X. 2017. Malware Traffic Classification Using Convolutional Neural Network for Representation Learning. *IEEE International Conference on Information Networking (ICOIN)* (2017).