A Hybrid Static Tool to Increase the Usability and Scalability of Dynamic Detection of Malware

Danny Kim
University of Maryland
dannykim@terpmail.umd.edu

Daniel Mirsky
University of Maryland
gomirsky@gmail.com

Amir Majlesi-Kupaei
University of Maryland
majlesi@umd.edu

Rajeev Barua
University of Maryland
barua@umd.edu

***Abstract*—Malware detection is a paramount priority in today's world in order to prevent malware attacks. Malware detection comes in three methods: static analysis, dynamic analysis, and hybrids. Static analysis is fast and effective for detecting previously seen malware where as dynamic analysis can be more accurate and robust against zero-day or polymorphic malware, but at the cost of a high computational load, which results in an often-prohibitive dollar cost for the needed server farm to handle all incoming traffic at an organization's network entry point. Most modern defenses today use a hybrid approach, which uses both static and dynamic analysis to maximize their chances of detecting malware. However, current hybrid approaches are suboptimal. We propose a solution to utilize the strengths of both while minimizing their weaknesses by using a two-phase hybrid detection tool. The first phase is a static tool, which we call a "static-hybrid" tool, that is based on machine learning and static analysis to categorize incoming programs into three buckets: definitely benign, definitely malicious, and needs further analysis. Only the small fraction of programs in the third bucket are run on the dynamic analyzer. Our system approaches the accuracy of the dynamic-only system with only a small fraction of its computational cost, while maintaining a real-time malware detection timeliness similar to a static-only system, thus achieving the best of both approaches.**

A key feature of our system is that the first (static) phase can run in active mode, *i.e.* it blocks malware in real time, which is possible because of the low 0.08% rate of mistakenly blocking benign programs as malicious (all results in our salient configuration). The second (dynamic) phase is run in passive mode, *i.e.* it send alerts for suspected malware without blocking them, and has a higher false positive rate of 0.75%. The first phase blocks 88.98% of malware, whereas the second phase brings up the detection rate to 98.73%. Since only a small fraction of malware missed by the first stage but caught by the second stage generates alerts, our system reduces alerts by 9.5X vs any highly accurate system running by itself in the typical passive mode seen in practice. Since only 3.63% of programs that need further study are sent to the second phase, this reduces the computation load for dynamic analysis by 100/3.63 = 27.5X.

I. Introduction

Advances in malware detection have been invaluable in stopping the growing number of malicious threats. Malware has continued to grow at an increasing rate causing malware detection to be at the forefront of cybersecurity research.

Modern intrusion detection systems (IDSs) are responsible for stopping unwanted software from entering an enterprise's network, while ensuring that benign programs are not falsely flagged. IDSs that rely on static analysis can be deployed as active defense systems because static analysis is fast and can handle at scale the thousands of programs coming into the network. An active defense system is one that can stop programs in real time from coming into the network if flagged as malicious. However, active systems must maintain an extremely low false positive rate to ensure that benign programs are not falsely flagged. This is done by setting the threshold very high for how confident the IDS is in order to classify something as malicious, allowing some malware to slip through the detection system. Thus in order to deploy an active static IDS, the detection rate of malware is sacrificed at a cost of reducing false positives.

IDSs that analyze every file dynamically present a different strengths and weaknesses. Dynamic analysis is based on the behavior rather than the static attributes of malware, which can make it more robust against packed, obfuscated, and previously unseen malware. Work such as [1] show that dynamic analysis can obtain useful information, which often leads to higher accuracies of detection, that static analysis cannot. IDSs that use dynamic analysis on every incoming file are typically deployed as passive systems, meaning that programs that arrive at the network are not blocked in real time. Because dynamic analysis requires the execution of the program in a protected environment, also known as a sandbox, analysis takes minutes to complete. Because malware is not blocked in real time, it is allowed to go past a network's defenses during analysis and wreak havoc before being potentially detected by the dynamic IDS a few minutes later. At this point, remedial action is taken at the infected endpoint to remove the malware and perhaps re-image the endpoint if the malware has run. We define the time elapsed between when a malware enters the network and is stopped as timeliness.

Most modern malware defense systems rely on hybrid approaches, which combine both static and dynamic analysis in order to defend against malware. However, we found from a literature and industry review that the way in which static and dynamic analysis have been combined is suboptimal. Most hybrid approaches work one of two ways. The first is a system which performs both static and dynamic analysis on all incoming traffic to obtain static and dynamic information in order to maximize a system's accuracy. The problem with this method is that although both sets of static and dynamic information are obtained, the computational and timeliness costs are prohibitively high because all programs are dynamically analyzed. The other primary hybrid approach is using a

978-1-7281-0155-2/18/$31.00 ©2018 IEEE

static analysis-based tool to detect any incoming threats. The detected threats, based on static analysis alone, are sent to a Security Operations Center (SOC) to be analyzed further by other methods such as dynamic analysis. This method is problematic because it relies solely on the capability of static analysis to detect threats. As mentioned above, any active system maintains an extremely low false positive rate (to remain usable in the real world) by sacrificing its detection rate. Thus by only relying on static analysis, the malware detection system is incapable of defending robustly against all types of threats.

We present a new hybrid malware detection configuration that runs at the network entry point to an enterprise that strategically leverages the strengths of both static and dynamic analysis while minimizing their weaknesses.

Our proposed detection system contains a two-step process. First, a tool which we call a *static-hybrid* tool analyzes all incoming programs statically and categorizes them into one of three buckets: very benign, very malicious, and needs further analysis. It is thus named to distinguish it from usual static tools, which categorize incoming programs into only two buckets: malicious or benign. The very benign bucket contains programs that the static-hybrid is highly confident are benign. This bucket contains near zero false negatives (*i.e.*, missed malware). The very malicious bucket contains programs that the static-hybrid tool is highly confident are malicious. This bucket contains near zero false positives (*i.e.*, benign programs falsely detected as malware). The needs further analysis bucket contains programs the static-hybrid tool is unable to make a strong determination on and thus is passed to the dynamic analysis tool, which is the second step of the process.

The programs located in each bucket provide unique improvements to existing IDSs. The programs in the benign bucket can directly bypass the dynamic analysis portion of our tool, reducing the computational resources needed for dynamic analysis. The programs in the malicious bucket can be blocked immediately, improving timeliness and reducing the need to conduct damage control after a program is identified as malicious in a passive IDS. The programs in the needs further analysis bucket are the only programs passed to the dynamic analysis tool. These programs, as our results will show, have a lower chance of being categorized correctly by a static analysis tool compared to a dynamic analysis tool. Thus by utilizing dynamic analysis on this subset of programs only, our tool's overall accuracy is higher than a strictly static IDS. In addition, because only the programs located in the very malicious bucket are immediately blocked, our static-hybrid tool is able to operate at a much lower false positive rate than a comparably built static-only tool, as our results will show.

Our scheme reduces the computational resources by 27.5X in our salient configuration. This reduction is important for the practicality of a dynamic scheme. For example, a system that analyzes every file dynamically using a sandbox that encounters 200,000 files/day at an organization's network entry point (a typical file volume for a mid-size organization), we estimate would cost $24,000/year in cloud computing costs alone to run the full sandbox load, based on commercial cloud costs today. By reducing the computational load by 27.5X, the cost of protection would also drop by 27.5X with a very small loss in accuracy of protection.

In practice, any active system must maintain a near-zero false positive rate to avoid adversely affecting an enterprise's users. Typical active IDSs block 100% of the programs flagged as malicious. However, our tool's two-step process allows it to only block a subset of programs flagged malicious. This is unique to our tool and we define it as the Blocked Benign Rate or BBR. The BBR is the percentage of benign programs stopped in real-time, which is distinct from the overall false positive rate that measures the total percentage of benign programs flagged as malicious. Our goal is to produce a practical, scalable malware detection tool that strategically combines the benefits of both static and dynamic analysis. Thus, in order to ensure the practicality of our system, we chose our salient configuration to have a BBR of 0.08%. Our results show that our salient configuration can still block 88.98% of malware immediately. Additionally, in practice we believe that the BBR can be further reduced by combining our system with other AV tools or file reputation websites.

Our work differs from previous work because we are the first to use a static-analysis based tool to generate three answers instead of two. This allows us to utilize the strengths of static analysis to quickly and correctly detect a subset of malware and goodware for which it is very confident, while using dynamic analysis to more accurately classify the remaining programs. We are the first to propose such a system to improve existing hybrid malware defense systems' scalability and performance. Because of our two-phase design, we can reduce computational resource requirements and improve timeliness of schemes that use dynamic analysis on all incoming traffic by much larger factors in comparison to previous work. We introduce a system that combines both static and dynamic analysis in an innovative way that takes maximizes the strengths of both while minimizing their weaknesses.
Our contributions are as follows:

- Propose a two-phase malware detection tool that uses a static-hybrid-based machine learning tool to reduce the computational resources needed, improve the timeliness, and reduce the alerts generated by dynamic analysis.
- Prove that on programs a static analysis tool has difficulty classifying, dynamic analysis can provide a dimension of information which enables it to more accurately classify them, ultimately improving our system's overall accuracy compared to static-only based methods.
- Show that our resulting hybrid system is able to reduce computational requirements of typical dynamic analysis by 27.5X, block 88.98% of malware in real-time with a 0.08% BBR (detecting 98.73% of malware eventually), and reduce the number of alerts generated by 9.5X.
- Obtain the results on our system by training its machine learning component on a set of over half a million programs, equally split between malware and benign programs (also known as goodware).

The rest of the paper is organized as follows. Section II covers the related work in the field along with our distinctions. Section III covers our design and explains in detail the technology behind our scheme. Section IV covers the specifics of our machine learning implementation, feature set, dataset, and testing setup. Section V covers our results.

II. Related Work

Our goal has two main components. In comparison to dynamic analysis-based IDSs, our technology aims to reduce the computational load, while improving the timeliness of dealing with a malware. In comparison to static IDSs, our technology aims to perform more accurately. On these fronts, there are several related works that are explained below.

[2] had the most similar goal to ours of improving the efficiency of dynamic malware analysis, but tackles the problem through reducing the number of polymorphic samples tested. Polymorphic malware are malware that contain different bytes as to avoid signature-based detection, but perform the same actions. [2] states that due the high number of polymorphic samples, dynamic analysis is often unnecessary, thus it only dynamically analyzes each malware sample for a short period of time, then compares the dynamic analysis results with that of previous malware. If it matches, then analysis is stopped, thus saving time and resources. However, their work was only able to forgo 25% of analysis, while our work reduces the computational requirements by 27X. Additionally, spawning a sandbox takes a non-trivial amount of time meaning that their scheme cannot in real time stop malware (*i.e.*, there is no improvement in timeliness).

Another class of related works are papers that aim to detect malware similarity [3]–[10]. These works use static or dynamic analysis in order to cluster malware into families or find similarities between them. However, these works do this to try to aid forensic malware analysis, rather than trying to reduce the computational load of dynamic IDSs. All these works were only tested on malware. Our goal and implementation of our static-hybrid is fundamentally different in that it is used to distinguish malware from goodware.

[11]–[13] specifically aim to reduce the time dynamic analysis takes to analyze each sample. [11] uses static features to try to predict how long it will take for dynamic analysis to uncover malicious behaviors. [12] uses network analysis to specifically detect if dynamic analysis should be suspended. Both works still require the use of a sandbox, which means their tools cannot stop malware in real-time or reduce the number of programs analyzed dynamically. [13] proposed using cloud computing features to support and enhance malware analysis with the goal of reducing analysis response time, but only could improve it by 23%. Additionally, their tool is strictly a malware analysis tool, not a malware detection tool like ours.

[14] aims to maximize the value of the information obtained from dynamic analysis by using static analysis to cluster malware behavior. Their goal is to reduce duplicative runs of dynamic analysis for malware to maximize the efficiency of sandboxing. Although they do improve the efficiency of dynamic analysis, they do not aim to produce a malware detection tool such as ours.

[15] built an endpoint tool to detect maliciousness within the first five seconds of analysis. Their work is similar to ours in that the authors try to quickly detect maliciousness, but is unrelated because their tool is an endpoint tool, rather than a network IDS like ours. Our work aims to prevent malware from reaching the endpoint all together. [16] similarly built an endpoint tool, but uses a set of malicious dynamic models of behavior generated during dynamic analysis to detect malware on an endpoint. Their technology enables fast detection of malware on an endpoint, but again does not try to stop malware from reaching the endpoint all together. Endpoint tools have an entirely different set of trade offs from network-entry-point tools. In practice, the two types of tools are not in competition, with many enterprises using both types simultaneously to achieve a multi-layered defense.

[17]–[20] all use a combination of static and dynamic analysis to either detect or forensically analyze malware. Although these works use a combination of static and dynamic analysis likes ours, they combine the analysis into one set of information with the goal of maximizing accuracy. Our work uses the analysis types separately to reduce computational requirements and timeliness of dynamic analysis.

[21] proposes a two-phase classification system for detecting android malware. In the first phase, they use two separate bloom filters, one for malware and one for goodware, to classify incoming programs as malicious or benign. If their initial-phase classifiers produce conflicting results, then the android program is passed to the second, more accurate classifier. Their work differs from ours for the following reasons. First, their work is specific to android software. Second, their initial phase uses two classification tools, whereas we rely on one and use the probability of maliciousness as a method of determining which samples are passed to our dynamic component. Third, their work was only tested on 190 samples and obtained an accuracy of less than 90%. Our tool was trained and tested on over 500,000 samples and obtained an overall accuracy over 99%. Further, the first phase of their tool still passed nearly 50% of the incoming programs to the second phase whereas our tool passed less than 10%.

The following works are tangentially related to ours, but differ significantly as explained below. [22] uses static analysis to fingerprint binary code for the purpose of speeding up forensic analysis and not malware detection. [23] uses dynamic analysis to enhance static analysis for Internet malware binaries, but their technology cannot be used to reduce computational resources like ours. [24] introduces of method of smartly storing signatures on an endpoint because of the enormous amount of signatures needed to detect all malware today using a signature-based approach. [25] uses a two-phase classification tool to detect malicious obfuscation code, but performs poorly when used to detect malware.

III. Our Two-phase System

A. Architectural Comparison VS Existing Hybrid Tools

To understand the novelty behind our work, we must first look at how existing hybrid tools operate. Figure 1 shows a hybrid configuration that utilizes both static and dynamic analysis to detect malware and generate behavioral reports. 100% of the the network traffic is analyzed both statically and dynamically. Although this configuration offers the best accuracy, its costs in terms of computational resources and timeliness are prohibitively high because of the use of dynamic analysis on all incoming programs.

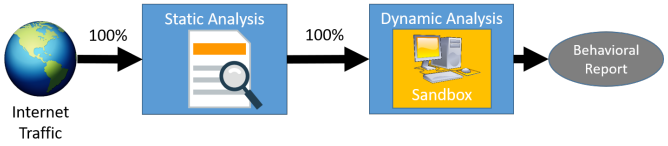

Fig. 1: Advantages: Maximum accuracy
Disadvantages: High computational costs, decreased timeliness

Figure 2 shows a hybrid configuration that uses static analysis to filter incoming programs into two categories: benign and malicious. The traffic classified as malicious generates an alert, and then is sent to the SOC to be analyzed dynamically. This model is problematic because it relies solely on static analysis to detect malware. As mentioned previously, static analysis alone can struggle to detect heavily obfuscated or packed malware, limiting its overall accuracy [26]. In addition, the alerts generated all have to be analyzed dynamically, which increases the amount of work the SOC has to perform.

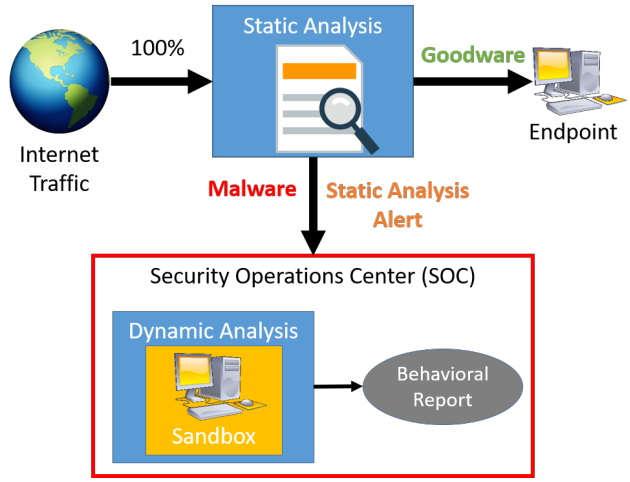

Fig. 2: Advantages: Fast detection, some computational benefits
Disadvantages: Limited to static-only accuracy, analyze all alerts

Figure 3 shows our configuration and its stark improvement over existing hybrid designs. Our configuration utilizes a static analysis tool to filter incoming programs into three categories: very benign, very malicious, and needs further analysis. Because of this innovative three bucket design, our tool can strategically deploy the use of dynamic analysis only when it truly is necessary to gain an accuracy benefit.

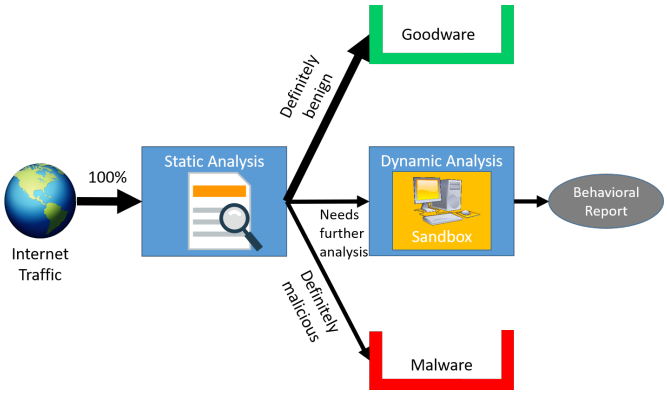

Fig. 3: Advantages: Bandwidth and timeliness improvements, near maximum accuracy

B. Our Design

The novelty in our work comes from our use of our static-hybrid-based machine learning tool. As shown in Figure 4, this static-hybrid tool divides incoming programs in real time into the following three categories based on thresholds T_1 and T_2. For the programs that the static-hybrid tool is confident is benign, the tool places them in the definitely benign bucket, bucket 1. The programs in the benign bucket bypass dynamic analysis entirely. For the programs that the tool is confident are malicious, it places them in the definitely malicious bucket, bucket 3. The programs in the malicious bucket are blocked in real-time, preventing malware from reaching the endpoint. For the programs that the tool is unsure about, it places them in the needs further analysis bucket, bucket 2. The programs in bucket 2 are passed to the dynamic analysis tool.

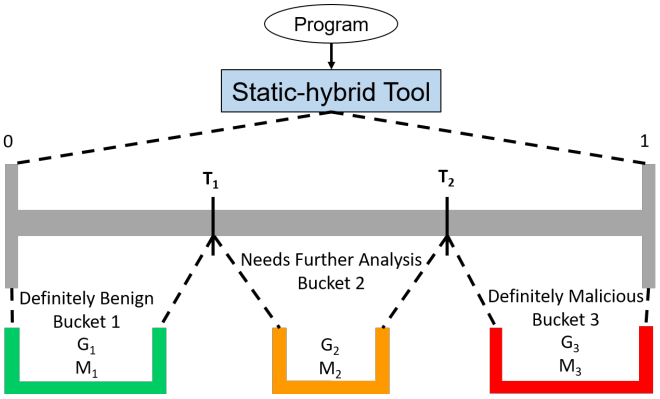

Fig. 4: Static-hybrid Categorization

The key to our system is our static-hybrid tool. This tool has two main benefits that help our system approach the accuracy of the configuration shown in figure 1 with a fraction of the computational resource usage and timeliness. First, on the programs the static-hybrid tool has a confident prediction for, it is very accurate. This helps our system maintain an overall accuracy similar to hybrid systems that use dynamic analysis on all incoming programs because a large percentage of programs are correctly categorized with very low error rates. Second, the static-hybrid tool can classify a large percentage of goodware and malware into the definitely benign and definitely malicious

buckets, respectively. By classifying a large percentage of goodware and malware into buckets 1 and 3 respectively, the static-hybrid tool maximizes the computational resource reduction and improvement in timeliness. Because it categorize both malware and goodware into their respective buckets, our configuration gains more computational benefits as compared to the hybrid configuration shown in figure 2.

For the subset of programs that the static-hybrid tool is unable to make a confident prediction for, our system's uses dynamic analysis because it can obtain behavioral information that static analysis cannot. This dynamic information can be helpful in more accurately detecting malware because it is based on what the malware does rather than static attributes that can modified by processes like packing. In the second phase of our tool, the dynamic information obtained is combined with the static information to give our machine learning classifier the best chance at accurately detecting the remaining malware and goodware.

Static and dynamic analysis are equipped to detect malware in fundamentally different ways. This observation is key in how and why our system is built. Our results show that a large portion of malware can be detected using static analysis. However, the percentage of malware that remain undetected still have massive financial and safety implications. Thus, dynamic analysis is key in modern defenses as well because it provides an additional source of information complementary to static analysis.

C. Computational Resources Reduction

The benign bucket is correlated to the amount of computational resources saved by not having to run dynamic analysis. As explained above, dynamic analysis is resource and computationally intensive. Especially since typically more than 99% of programs coming into a enterprise network are benign, dynamic analysis is wasteful and creates prohibitively high cost for relatively small benefits. By first learning to categorize goodware into bucket 1, our static-hybrid tool can save a tremendous amount of resources. This equates to reducing the cost of an accurate IDS for an enterprise.

D. Timeliness Improvement

The malicious bucket is correlated to the timeliness improvement by blocking malware in real time. In typical passive IDSs that utilize a configuration such as figure 1, malware is allowed through to the endpoint because dynamic analysis takes on the order of minutes to process per sample. In practice, since the percentage of programs that are indeed malicious is small, this is a compromise that is acceptable to users. However, in the cases that the program is malicious, the endpoint has to be quarantined and potentially restored from a previous backup to ensure that any infected files are not still on the machine. This is an ineffective and costly method of dealing with malware. Our tool relies on static analysis, but is coupled with the power of machine learning to detect a high percentage of malware. Its ability to detect a large percentage of malware with high confidence, as shown in our results, gives it the ability to stop most malware prior to it reaching the endpoint. This saves an enterprise the time and cost of quarantining and performing damage control after a typical passive IDS would have let malware pass through.

E. Hard-to-Classify Programs

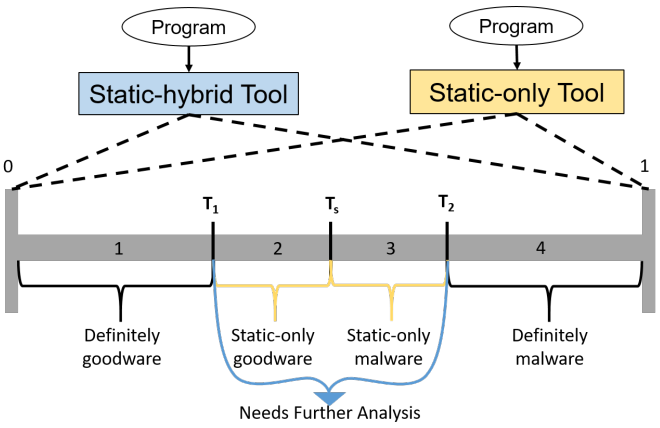

Fig. 5: Static-only VS. Static-hybrid Comparison

The needs further analysis bucket has the programs that the static-hybrid tool was unable to make a strong classification prediction for and thus are passed to the dynamic analysis tool. We refer to these programs has *hard to classify*. Our belief is that the dynamic portion of our tool is able obtain information that can help it more reliably determine the correct classification for this subset of programs. Our conjecture is that the programs located in this bucket are harder to classify than the programs in buckets 1 and 3 and thus would benefit from dynamic analysis. The goal of our tool is to minimize the this subset of programs to maximize our system's benefits.

Our results will show that on the subset of programs located in bucket 2, information from dynamic analysis is indeed valuable for more accurately classifying these programs compared to static analysis. In many cases, the increase in accuracy is dramatic, which quantifies the benefit of utilizing dynamic analysis versus static analysis. Figure 5 shows the overlap of our static-hybrid tool versus a tool based on static analysis alone, such as the configuration shown in figure 2. In a typical static-only-based tool, the programs with probabilities of maliciousness between T_1 and T_s would be categorized as benign and the programs with probabilities between T_s and T_2 would be categorized as malware. However, our work shows that static-only-based tools are ill equipped to reliably categorize these and thus passing them to dynamic analysis maximizes the chances of being classified correctly.

IV. Implementation

A. Dataset

Our total dataset contained 264,769 goodware and 259,356 malware Windows PE32 executables. We obtained our malware and goodware datasets through a combination of Virus Total and Virus Share [27], [28]. Both graciously allowed us to download a large number of malware and goodware. Although

we cannot guarantee that our dataset is perfectly representative of all programs in the wild, we took every precaution we feasibly could while trying to amass a large, diverse set.

For our malware, we collected programs that had at least 15+ detections on Virus Total. Virus Total queries nearly 60 Anti-Virus (AV) tools to determine a program's maliciousness. We chose 15+ as an acceptable threshold to ensure the ground truth in our dataset because it meant that at least 25% of the AV tools believed this program is malicious. We ensured that our malware dataset was temporally diverse by choosing a specific number of malware per year. Our malware dataset had 5,000 samples per year from the years 2001 to 2008. For the years 2009 to 2017, we chose 20,000 per year. Lastly, we chose 40,000 malware from our total dataset that had a PE timestamp that was prior to 2001 or after 2017, as these timestamps were clearly tampered with. This distribution was chosen based on distribution of a malware test set we received from a private malware analysis company. The malware chosen per year were chosen randomly.

For our goodware, we collected all of our programs from the years 2005 to 2016 and ensured that each had zero detections on Virus Total. We chose to download programs from Virus Total with zero detections because we believed that if a program was not flagged by any AV tools for more than a year, then it was most likely benign. A similar method was taken in these works [15], [29]. Additionally, this was the only way of obtaining a dataset that was large enough to effectively train and test our machine learning algorithm.

B. Machine Learning

Our machine learning-based detection tools were trained and tested using a Multi-Layer Perceptron (MLP) neural network implemented with Tensorflow with Keras [30]. Our MLP configuration, we used one hidden layer with a size equal to the input layer with dropout set to 0.5 between each. We used a learning rate of 0.01, learning rate decay of 1e-6, and Nesterov momentum set to 0.9. For our first two layers, we used a rectified linear unit activation function. We trained the MLP for 100 epochs as we observed in our training that our tool's accuracy on the validation set leveled off after 100 epochs. The final layer of our tool was a Softmax layer used to output a probability of maliciousness. We trained and tested our machine learning algorithm using a 4-fold cross validation.

The goal of our system was not to extensively test machine learning algorithms, but to show that our tool is capable of utilizing the unique strengths of both static and dynamic analysis in a previously unseen way.

C. Feature Set and Testing

For the static portion of our test, we generated static reports for each sample with the following class of features. Our static features comprised of header data such as imported APIs, and dynamically linked libraries (DLLs), section entropy, and YARA rules. We chose to analyze APIs and DLLs based on previous work's successes using them as features. We also used YARA rules to detect signatures that were obtained from the open-source YARA-Rules project [31]. Due to the high number of unique imported APIs and DLLs found across over half a million programs, we filtered out the ones that did not occur at least 10,000 times in either the goodware or malware set. We chose 10,000 because it produced a feasible number of features that our machines could process and train on given our large dataset. This resulted in a set of 4,002 static features. The number of static features is large because each class of static features we collected, such as imported APIs, produced a large number of individual features.

For the dynamic portion of our tool, we executed each sample in a Cuckoo Sandbox for 2 minutes. Cuckoo Sandbox is an open-source sandbox manager built to dynamically analyze programs [32]. Each sandbox was run with Windows 7, 1/2 GB of ram, and 1 core. Each sandbox was given Internet access using the Whonix gateway to obscure where the HTTP requests were originating from [33].

Our dynamic feature set was a superset of all the static features in addition to operating system (OS)-level and program-level behavioral signatures. The OS-level behavioral signatures were obtained using Cuckoo Monitor, Cuckoo's dynamic analysis tool that monitors the interaction between a program and the OS to build dynamic signatures. For example, Cuckoo Monitor will generate an HTTP_REQUEST signature if the program attempted to connect to the Internet. The program-level behavior was obtained from a dynamic binary instrumentation (DBI) tool based on the technologies found in [1]. The program-level features detected instruction-level obfuscations and collected instruction-level statistics. We used DynamoRio, an open-sourced DBI tool, in our implementation [34]. Our dynamic feature set was 4,594 features in total.

It should be noted that our hybrid configuration, which is our main contribution, can be used by any existing static and dynamic analysis hybrid malware detection system that utilizes machine learning. Although untested, our conjecture is that our configuration's benefits are not unique to our system's feature set or implementation.

V. Results

A. Static VS Dynamic IDS

Here, we cover the performance of the static phase versus the dynamic phase of our two-phase system on our total dataset to prove the usefulness of dynamic analysis as a complementary addition to static analysis.

Table I below shows the following information for using a strictly static analysis-based MLP tool, and a combination of static and dynamic analysis-based MLP tool. First, it gives the detection rate, which is the percentage of malware detected out of the malware test set. Second, it shows the false positive rate, which is the percentage of goodware that are falsely flagged as malicious. Third, it shows the area under curve (AUC), which corresponds to a machine learning classifier's receiver operating characteristic that shows the detection rate across a range of false positive rates. Lastly, it shows the F1 score, which is common machine learning metric used in practice.

TABLE I: Static analysis VS. Dynamic analysis Detection Results

Tool	Detection Rate	False Positive Rate	F1 Score	AUC
Static-based MLP tool	97.71%	0.75%	.9850	.9974
Dynamic-based MLP tool	99.02%	0.75%	.9919	.9980

We used the same dataset for our dynamic tool as our static tool: 264,769 goodware and 259,356 malware. However, not all the samples executed dynamically due to issues such as dependencies not being met. As a result, we generated dynamic reports for only 195,255 goodware (70.95% of our original goodware set) and 223,352 malware (86.27% of our original malware set). When training and testing our dynamic tool, we used the same training and test set as for our static tool during the 4-fold cross validation, but only included the ones that ran. Thus, the results shown above for the dynamic-based MLP tool is based on this subset of programs that ran.

Table I shows that the false negative rate (% of missed malware) for our dynamic tool is 0.98%, compared to 2.29% for our static tool. This is a decrease of 57.2% for the dynamic tool. These results show that dynamic analysis provides behavioral information that help a malware detection tool be more accurate than static analysis, which is why dynamic analysis is necessary in real-world tools. We observe similar increases in the F1 score and AUC compared to our static tool.

Although dynamic analysis has benefits in terms of accuracy, there are drawbacks related to the resources needed and timeliness as explained in Section III. First, spawning a sandbox per sample is unscalable for many enterprise networks due to the number of programs coming in daily. Second, dynamic analysis takes on the order of minutes meaning that malware cannot be stopped in real time, and instead are allowed to pass through to the endpoint. Only after the dynamic tool determines that the program tested is malicious is it stopped and the endpoint fixed. Additionally, not all programs execute dynamically, as we experienced in our experiments.

B. Definitions

This subsection defines the metrics used to quantify our improvements and results of our two-phase detection tool. G_1 and M_1 are the number of goodware and malware located in bucket 1, respectively. The same notation is used for buckets 2 and 3. In our notation, N_G and N_M are the total number of goodware and malware respectively.

1) Computational Requirements Reduction: We define the computational requirements (CR) of traditional dynamic IDSs as the amount of resources needed to spawn a single sandbox per sample arriving at the network. Using a typical dynamic IDS, the CR would be equal to 100%. We calculate the computational requirements of our system by Equation 1.

$$CR = \frac{G_2 + M_2}{N_G + N_M} \tag{1}$$

To calculate the CR in our system, we have to find the percentage of goodware and malware located in bucket 2 (the bucket of programs that will be passed to dynamic analysis) and divide by the total number of goodware and malware. However, we know that in the real-world, a majority of programs that arrive at a typical enterprise firewall are benign, thus the number of goodware greatly outweighs the number of malware. We can then approximate the CR of our system to be simply:

$$CR \approx \frac{G_2}{N_G} \tag{2}$$

2) Timeliness Reduction: We define timeliness as the amount of time elapsed from the malware reaching the IDS to being stopped at the endpoint. We quantify it by calculating the BMR, which is how many malware are blocked and thus stopped in real time, shown in Equation 3. In a strictly dynamic IDS, live network traffic cannot be blocked, thus all malware that comes into the network is run on the endpoint for minutes prior to being detected and removed. Because our static filter is fast enough for it to be used with stopping live network traffic, it can block a percentage of malware from reaching the endpoint as seen in the results below.

$$BMR = \frac{M_3}{N_M} \tag{3}$$

3) Alert Generation: In typical dynamic IDSs, an alert is generated for every program flagged as malicious, which is then analyzed by a network administrator. In a dynamic scheme, the number of alerts generated are typically overwhelming and cause only a small percentage to be analyzed. In our scheme, because only a small portion of programs are dynamically analyzed, our system dramatically reduces the number of alerts sent to the network administrator.

Our alert generation (AG) metric is determined by the percentage of alerts that our dynamic portion generates compared to the number of alerts a fully dynamic scheme would produce. An alert is generated only when a dynamic tool flags a program as malicious (*i.e.*, a true positive, or false positive). The formal equation is defined in Equation 4. First, we calculate the percentage of malware that are in bucket 2 that are detected as malicious, or our detection rate (DR) on the portion of malware in bucket 2. We then account for the percentage of goodware that are in bucket 2 that are falsely flagged, or our false positive rate (FPR) on the goodware in bucket 2.

$$AG = (DR_{M_2} * \frac{M_2}{N_M}) + (FPR_{G_2} * \frac{G_2}{N_G}) \tag{4}$$

C. Static-Hybrid Tool: Minimizing BBR

The following results are for our salient configuration, where we chose thresholds T_1=0.2 and T_2=0.999999 to minimize the BBR. We show this configuration to explain our scheme's improvements in comparison to static-only and dynamic-only analysis. Static-only analysis refers to using a strictly static IDS while dynamic-only refers to using a strictly dynamic IDS.

In Tables II and III, we show the overall detection rate (DR), overall false positive rate (FPR), CR, AG, percentage

of malware blocked in real-time (BMR), and percentage of benign programs wrongly blocked in real-time (BBR). The overall DR is the percentage of total malware detected as malicious; in our tool, this is by either the static and dynamic phase. The overall FPR is what percentage of total goodware were falsely detected as malicious; in our tool, this is by either the static or dynamic phase.

TABLE II: Static-dynamic-analysis Results for T_1=0.2, T_2=0.999999

Tool	DR	FPR	CR	AG
Static-only	97.71%	0.75%	0%	0%
Dynamic-only	99.02%	0.75%	100%	100%
Static-dynamic	**98.73%**	0.75%	**3.63%**	**10.68%**

TABLE III: Static-dynamic-analysis Results for T_1=0.2, T_2=0.999999

Tool	BMR	BBR
Static-only	97.71%	0.75%
Dynamic-only	0%	0%
Static-dynamic	**88.98%**	**0.08%**

Tables II and III shows the followings conclusions about our scheme compared to a scheme that relies solely on static analysis to detect malware. First, our hybrid scheme reduces the overall false negative rate, or the percentage of missed malware, by 44.54%. Additionally, because our tool only blocks programs in real time that are located in bucket 3, our tool only blocks 0.08% of goodware (10.7% the goodware that the static-only tool blocks). A static-only tool stops any program flagged as malicious in real time. The static-only tool does stop more malware in real time compared to our tool (97.71% versus 88.98%), but ultimately does not detect as many malware as our tool because of our use of dynamic analysis. Static-only schemes only have one chance to stop malware and if it does not, the malware is passed into the network to be executed for an indeterminate amount of time. Our scheme also holds a significant advantage over static-only based detection tools because our tool can operate at almost 1/10 of the false positive rate, while ultimately detecting more malware. This greatly increases the usability and practicality of our system over existing static technologies today.

Tables II and III also shows the following conclusions about our scheme compared to a scheme that analyzes every file dynamically. The dynamic-only scheme is the most accurate, but has the costs of high computational resources, alert generation, and the inability to stop malware in real-time (as evidenced by BMR equal to 0%). Our results show that our tool, for the nearly the same DR as dynamic-only, is able to use only 3.63% of the computational resources (a near 27.5X reduction), generate only 10.68% of the alerts (a 9.5X reduction), and block 88.98% of malware from entering the network (a 9X reduction). Our scheme produces nearly the same overall DR as a dynamic-only scheme with a small fraction of the costs.

The reductions in computational resources and alert generation are related to increasing the scalability of dynamic analysis. By reducing computational resources by 27.5X, dynamic analysis's cost in terms of servers, virtual machines, and total analysis times is similarly reduced to ensure it can still be used even as the amount of network traffic continues to grow. Additionally, we argue that the alerts that are generated by our tool on the hard-to-classify programs are potentially more valuable than the alerts that would be generated by a dynamic-only scheme on every predicted malicious file. The alerts generated for a large percentage of malware we expect does not need human analysis because it is clearly malicious. Our tool automatically produces a set of alerts for programs that were hard-to-classify and thus may need human analysis to truly understand.

Our tool's BMR is an invaluable part of our system as compared to a dynamic-only scheme because a dynamic-only scheme cannot stop any malware in real-time. Our scheme's ability to block 89% of malware immediately saves an immense amount of post-infection damage control and removal. This ultimately leads to reduction in cost for an enterprise to maintain a secured network.

The dynamic-only metrics shown in Tables II and III were only on the programs that executed within the sandbox (approximately 78% of programs tested). This is a detriment to using a strictly dynamic IDS because not all programs execute within a sandbox. In those cases, our static-hybrid tool proves to be invaluable in maintaining a high detection rate, while still utilizing dynamic analysis's information when available.

1) Bucket 2 Analysis: In our scheme, our hypothesis is that current hybrid schemes do not optimally deploy dynamic analysis- meaning using dynamic information to boost confidence of prediction when static analysis alone is incapable of doing so. To prove that our system more optimally combines static and dynamic analysis, we tested the programs in bucket 2, the hard-to-classify programs, with static analysis and dynamic analysis and show the correctness of classification of goodware and malware. Correctness for malware is calculated by the DR of malware on the set of malware in bucket 2, while correctness for goodware is calculated by the FPR on the set of goodware in bucket 2. For the programs located in bucket 2 that did not able to execute dynamically, our scheme relied on static analysis for a classification.

TABLE IV: Bucket 2 Correctness Results for T_1=0.2, T_2=0.999999

Tool	Bucket 2	
	G (FPR)	M (DR)
Static	37.85%	6.90%
Dynamic	18.83%	96.72%

Table IV shows that on this subset of programs, each tool's correctness is significantly lower than their accuracies overall on goodware and malware. However, this table shows important conclusions. First, it proves that this subset of programs is hard-to-classify as evidenced by the worse accuracies. This means that the programs located in bucket 2 cannot be reliably classified by static analysis alone. Second, it shows dynamic analysis is the most capable of distinguishing these hard-to-classify programs. The dynamic analysis tool is able to nearly increase the DR of malware in bucket 2 by more than 14X while reducing the FPR by half as compared to the static-

only scheme. This result shows that by utilizing our system, dynamic analysis can still be used strategically (maximizing its complementary information to boost accuracy), while incurring minimal costs as shown in the Section V-C.

VI. Conclusion

Malware detection is a fundamental tool necessary to prevent attacks on information and security. IDSs play a pivotal role in preventing malware attacks, but the way that current IDSs use both static and dynamic analysis are suboptimal. Our two-phase system is a stark improvement over existing hybrid schemes. By understanding the unique strengths of static and dynamic analysis, our system is able to obtain an overall DR near that of a system that analyzes every file dynamically with a 27.5X reduction in computational resources, 10.5X reduction in alert generation, and a 9X reduction of malware entering a system's network. Additionally, our system can operate at a much lower FPR than a system that only relies on static analysis, greatly increasing its practicality. Existing hybrid technologies are correct in utilizing both static and dynamic analysis when detecting malware as they are complementary to one another. However, they fail to strategically deploy these technologies to maximize efficiency and usability. Our work introduces a novel method of accomplishing both.

VII. Acknowledgments

We would like to thank the US Government Laboratory for Telecommunication Sciences for its support via contract #H98230-13-D-00560037. We would also like to thank VirusTotal for its academic license.

References

[1] D. Kim, A. Majlesi-Kupaei, J. Roy, K. Anand, K. ElWazeer, D. Buettner, and R. Barua, "Dynodet: Detecting dynamic obfuscation in malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2017, pp. 97–118.

[2] U. Bayer, E. Kirda, and C. Kruegel, "Improving the efficiency of dynamic malware analysis," in *Proceedings of the 2010 ACM Symposium on Applied Computing*. ACM, 2010, pp. 1871–1878.

[3] G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, "A static, packer-agnostic filter to detect similar malware samples," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2012, pp. 102–122.

[4] G. Wicherski, "pehash: A novel approach to fast malware clustering." *LEET*, vol. 9, p. 8, 2009.

[5] A. Automatisierungssysteme and D.-I. U. Bayer, "Large-scale dynamic malware analysis."

[6] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.

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

[8] S. Cesare and Y. Xiang, "A fast flowgraph based classification system for packed and polymorphic malware on the endhost," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010, pp. 721–728.

[9] I. Santos, F. Brezo, J. Nieves, Y. K. Penya, B. Sanz, C. Laorden, and P. G. Bringas, "Idea: Opcode-sequence-based malware detection," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2010, pp. 35–43.

[10] V. P. Nair, H. Jain, Y. K. Golecha, M. S. Gaur, and V. Laxmi, "Medusa: Metamorphic malware dynamic analysis usingsignature from api," in *Proceedings of the 3rd International Conference on Security of Information and Networks*. ACM, 2010, pp. 263–269.

[11] S. Kilgallon, L. De La Rosa, and J. Cavazos, "Improving the effectiveness and efficiency of dynamic malware analysis with machine learning," in *Resilience Week (RWS), 2017*. IEEE, 2017, pp. 30–36.

[12] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–7.

[13] O. L. Barakat, S. J. Hashim, R. S. A. B. R. Abdullah, A. R. Ramli, F. Hashim, K. Samsudin, and M. Ab Rahman, "Malware analysis performance enhancement using cloud computing," *Journal of Computer Virology and Hacking Techniques*, vol. 10, no. 1, pp. 1–10, 2014.

[14] M. Neugschwandtner, P. M. Comparetti, G. Jacob, and C. Kruegel, "Forecast: skimming off the malware cream," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 11–20.

[15] M. Rhode, P. Burnap, and K. Jones, "Early stage malware prediction using recurrent neural networks," *arXiv preprint arXiv:1708.03513*, 2017.

[16] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X.-y. Zhou, and X. Wang, "Effective and efficient malware detection at the end host." in *USENIX security symposium*, vol. 4, no. 1, 2009, pp. 351–366.

[17] J. B. Fraley, "Improved detection for advanced polymorphic malware," Ph.D. dissertation, Nova Southeastern University, 2017.

[18] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference CISIS12-ICEUTE 12-SOCO 12 Special Sessions*. Springer, 2013, pp. 271–280.

[19] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*. IEEE, 2006, pp. 289–300.

[20] P. Shijo and A. Salim, "Integrated static and dynamic analysis for malware detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.

[21] P. M. Kate and S. V. Dhavale, "Two phase static analysis technique for android malware detection," in *Proceedings of the Third International Symposium on Women in Computing and Informatics*. ACM, 2015, pp. 650–655.

[22] L. Nouh, A. Rahimian, D. Mouheb, M. Debbabi, and A. Hanna, "Binsign: fingerprinting binary functions to support automated analysis of code executables," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2017, pp. 341–355.

[23] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis," in *European Symposium on Research in Computer Security*. Springer, 2008, pp. 481–500.

[24] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, "Splitscreen: Enabling efficient, distributed malware detection," *Journal of Communications and Networks*, vol. 13, no. 2, pp. 187–200, 2011.

[25] D. Swathigavaishnave and R. Sarala, "Detection of malicious code-injection attack using two phase analysis technique," *Pondicherry Engineering College Puducherry, India*, 2012.

[26] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007, pp. 421–430.

[27] V. Total, "Virustotal-free online virus, malware and url scanner," *Online: https://www. virustotal. com/en*, 2012.

[28] J.-M. Roberts, "Virus share.(2011)," *URL https://virusshare. com*, 2011.

[29] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*. IEEE, 2015, pp. 11–20.

[30] F. Chollet, "Keras: deep learning library for theano and tensorflow. 2015."

[31] "Yara-rules," https://github.com/Yara-Rules/rules, 2017.

[32] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "The cuckoo sandbox," 2012.

[33] [Online]. Available: https://www.whonix.org/

[34] D. Bruening and S. Amarasinghe, "Efficient, transparent, and comprehensive runtime code manipulation," Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2004.