AI ATAC 1: An Evaluation of Prominent Commercial Malware Detectors

Robert A. Bridges†, Brian Weber† Justin M. Beaver‡, Jared M. Smith¶ Miki E. Verma*,
Savannah Norem†, Kevin Spakes†, Cory Watson†, Jeff A. Nichols†, Brian Jewell†,
Michael. D. Iannacone†, Chelsey Dunivan Stahl†, Kelly M.T. Huffer†, T. Sean Oesch†,

†Oak Ridge National Laboratory Oak Ridge, TN ‡Lirio LLC Knoxville, TN
¶Security Scorecard, New York, NY *Stanford University Palo Alto, CA

{bridgesra, weberb, spakeskd, watsoncl1, jewellbc, iannaconemd, nicholsja, hufferkm, dunivanck, oeschts} @ornl.gov
jbeaver@lirio.com, jared@jaredsmith.io, meverma@stanford.edu, savannah.norem@gmail.com

***Abstract*—This work presents an evaluation of six prominent commercial endpoint malware detectors, a network-based malware detector, and an file-conviction algorithm from a popular cyber technology vendor using 100,000 (50/50% benign/malicious) file samples with a realistic distribution of filetypes. Approximately 1,000 zero-day program executables are included (98% of malware were public). This experimen was administered as the first prize challenge in the Artificial Intelligence Applications to Autonomous Cybersecurity (AI ATAC) series. We present a novel evaluation process of delivering a file to a fresh virtual machine (VM) donning the detection technology, waiting ∼90s to allow static detection, then executing the file and waiting another period for dynamic detection. In order to execute all 800K trials (100K files × 8 tools) a software framework is presented that choreographed the whole experiment into a completely automated, time-synced, and reproducible workflow with substantial parallelization. Detection results, time, and host resource requirements are observed. A cost-benefit analysis was tailored for this competition to integrate the tools' recall, precision, time to detection, and resource requirements into a single comparable quantity that simulates real-world use. Summarized results are as follows: endpoint detectors achieve ∼50% recall; ML-based tools far outperform signature-based tools on the zero-day files, but otherwise signature-based tools are competitive if not better; dynamic detection increases recall substantially; detection abilities on all tools are relatively strong for program executable (PE) filetypes; detection abilities on other filetypes vary widely across technologies, with some technologies categorically failing on any non-PE filetype.**

***Index Terms*—malware detection, endpoint detection, network detection, evaluation, test, intrusion detection, cost benefit analysis, static analysis, dynamic analysis, machine learning**

I. Introduction

Malicious software or malware refers to any file that seeks to disrupt operations, corrupt data, allow unauthorized access to data or systems, or otherwise cause unwanted consequences to users. Given the repeated reports of widespread use of malware, e.g. [1], antivirus (AV) software, or endpoint malware detection technologies—which reside on each host and provide detection and quarantine capabilities against malware—are one of the most common and critical security defenses for personal computing devices to large enterprise networks.

Research of the efficacy of commercial malware detectors is fairly nascent in the, with the largest study to date of Bridges et al. [2] testing two endpoint and two network-level malware detectors on ∼3,500 files. Recent evaluations by industry e.g. exist but are either small in terms of test samples use, e.g,. [3], or lack enough difficulty in the testing samples to differentiate tools e.g., [4, 5]. See the Related Works section of Bridges et al. [2] for details. Notably, the evaluation of Bridges et al. [2], revealed: near perfect precision in all four tested tools; detection rates of ∼35-55%, as compared to near perfect recall ($> 99\%$) in industry evaluations, with these rates dropping for filetypes other than program executables (PEs); and that machine-learning-based (ML-based) tools achieve similar detection rates (∼40%) on zero-day (never-before-seen) files while signature-based tools fail to detect nearly all zero-days (∼4% detection rate). These recent results are alarming and illuminate a glimpse at the state of our defenses; hence, experimentation to further evaluate commercial malware detectors is warranted to understand both our defense capabilities and areas for future research.

Contributions of the AI ATAC 1 Endpoint Malware Experiment

This paper documents the largest research evaluation (in terms of numbers of files) of commercially available malware detectors, using 100,000 files to test eight detectors—six endpoint detectors, one network detector, and a malware detection algorithm provided by a security vendor. This evaluation was administered as the first prize challenge in the Artificial Intelligence Applications to Autonomous Cybersecurity (AI ATAC) series called the Endpoint Malware Detection Challenge [6], with the goal of enticing technology creators to submit endpoint malware detection capabilities that use machine learning (ML) to accurately identify malicious files on a host. As an

This manuscript has been co-authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the US DOE. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

agreement of the challenge, we cannot disclose the names of the technologies submitted nor the vendors, but all were from prominent vendors in the cyber technology market space.

Descriptions of the design and configuration of the research range (data center) built to accommodate this an other experiments is previous work [7]. This experiment and analysis builds directly on previous work, Bridges et al. [2], but contains novel contributions discussed in the following subsubsections.

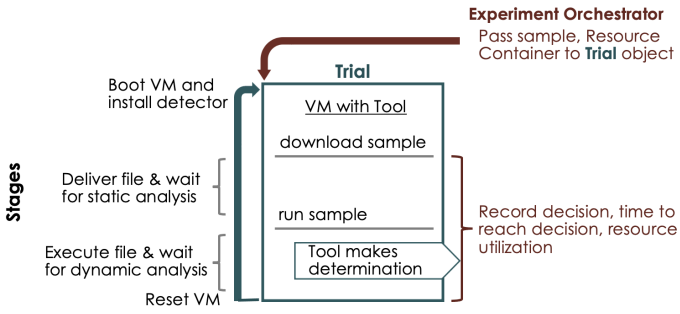

Fig. 1: Trial workflow depicted. Experiment Orchestrator passes file and resource information to Trial class, which, after instantiating a VM with the detection tool under test, executes stages of the trial: (1) downloads file and waits for static detection (2) executes file and waits for dynamic detection. The Experiment Orchestrator records decision by tool, time of decision, and resource utilization. In practice, the Experiment Orchestrator ran ~2,000 trials in parallel.

1) Experimental Design: The design of the experiment provides a higher fidelity evaluation for two reasons (besides the increase in file quantity). We define a *trial* as a sub-experiment testing a single tool's ability to test a single file. Each trial progressed through *stages*: first the file was delivered to a fresh virtual machine (VM) instantiated with the detection technology; then after a period of ~90s the file was executed; finally after a period of time the VM is closed. (See Figure 1.) These stages allowed for the tool under test to detect the file if not statically (before execution), dynamically (during or after execution). At the cost of much greater computational expense, this provides higher fidelity in the experiment with respect to how a detector will react when presented with a file on a host (e.g., our Baseline 2 tool seemingly only attempts to analyze a file once it is executed), and therefore allows more accurate time-to-detection, resource usage, and accuracy observations. Secondly, resource utilization of the tool under test was recorded and taken into account in this study. Thirdly, in our corpus we took care to try to match the percent of files of each type to the distribution reported by VirusTotal (www.virustotal.com) to gain realism. Fourthly, with the help of a security operation center, we use real statistics on the number of files encountered per year by a endpoint detector to inform the cost-benefit analysis of the results.

2) Experiment Implementation Framework: As a second contribution, we discuss a novel software framework for repeatably implementing the experimental workflow, allowing thousands of parallelized trials. Given the experiment required 800K trials (8 tools $\times$ 100,000 files), a simple serial implementation, taking an estimated ~4.5 years, was completed in ~15 hours per tool or ~5 days in total.

3) Cost-Benefit Analysis: While the scale (100K files with static and dynamic execution) of this detection experiment allowed for more fine-grained results (recall, precision, time to detection, resource metrics), a scoring framework that translates the many experimental observations to a real-world context was needed to reason about the tools' abilities, and for the sake of the competition, to make the multi-dimensional measurements comparable. A general cost-benefit analysis [8] designed for this scenario was adapted to this experiment. There are a two benefits of this cost model simulation: (1) the model integrate the many experimental measurements into a single, comparable, quantifiable cost by simulating of how the tool would perform in a security operation; (2) it provides a framework for reasoning about the varied results, e.g., it allows scaling the detection results to costs for a year of use while respecting a real-world percentage of malware/benignware, e.g., it allows varying the attack costs of zero-day files to weigh the benefits of ML-based vs. signature-based detectors.

The cost model in this work builds on the implementation of Bridges et al. [2] with three main additions. First it accommodates the higher-fidelity experiment used in this paper, with added mechanisms for accruing lower costs for pre-file-execution detection than post-execution; second, this cost model instantiation accommodates the observed host resources of the detector; third, we consult data observations from a SOC to understand the quantity of file decisions a host makes in its first year. We illuminate some inherent drawbacks to the cost model that can help steer future research.

4) Commercial Malware Detectors Summarized Findings: Our main contribution is our summarized findings on the commercial endpoint detectors from this experiment.

- All tools provide near perfect precision and except for the lone dynamic, network-based malware detector, achieve recall of ~50% (on corpora of 98% public files).
- Host, signature-based detectors are competitive if not better in recall and time to detection than host, ML-based detectors on public malware.
- Signature-based detectors fail almost categorically on zero-day malware (~4% recall), while ML-based tools provide about 10$\times$ the recall on never-before-seen files.
- Our cost model analysis confirms the previous two bullets with a signature-based tool prevailing when costs per malware are constant, but ML-based tools will prevail if one assumes zero-day files accrue much greater costs.
- The lone dynamic detector has much higher recall than the other tools.
- Many tools only detect PE files, while other tools provide competitive recall across all filetypes tested. There is large variance in abilities on non-PE filetypes.
- Time from file delivery until detection is only a matter of seconds (in median) for most tools, with one tool taking median ~1m, and one tool waiting until file execution to begin analysis, then taking median 9s. Overall, modern detectors are fast.

Our takeaways reaffirm those of Bridges et al. [2] now with more tools and much more data.

II. Related Works

In order to investigate what an adversary can learn about a blackbox malware detector, Christodorescu & Jha (c. 2004) [9] evaluated three commercial malware detectors using on only eight malicous samples. Findings included dismay at the "dismal" state of malware detectors.

Pandey & Mehtre [10] (c. 2014) compute accuracy, recall, specificity, and a poorly described "efficiency" quantity for 17 different file analysis or online services using 29 malicious samples.

Aslan et al. (c. 2018) [11] use 200 samples, 100 benign, 100 malicious, to evaluate many static, dynamic, and online file analysis tools. Detection rates vary from 62-77% with dynamic analysis tools outperforming static analysis tools, and combinations of tools outperforming providing better coverage than a single member of the ensemble.

Fleshman et al. (c. 2018) [12] perturb known malware to create obfuscate variants from public malware samples. Using 1,000 malware samples, commercial malware detectors and non commercial supervised learning algorithms are tested. The two ML algorithms are found to be more robust to perturbations than commercial detectors.

Abdelsalam et al. (c. 2019) [13] leverage the expected behavioral similarity between auto-scaled VMs in the cloud to train shallow CNNs to be used as anomaly detectors. The intuition behind their approaches are that auto-scaled VMs (identical VMs which are spawned/destroyed in response to application load) should have similar behavioral characteristics at any given point in time. They introduce two novel approaches: Mulitple VMs Single Samples (MVSS) and Multiple VMs Paired Samples (MVPS). MVSS trains on the behavior of auto-scaled VMs running at the same time. Since the training process uses the behavior of VMs running in parallel, it captures VM interactions. During inference, the behavior of each individual VM for a slice of time is classified as benign or malicious. MVPS classifies whether the behavior of any two VM pairs for a given slice of time is similar. If the behavior is different, it is treated as a malicious classification. To test their approaches, they execute an experiment on a test-bed which utilizes VM auto-scaling, testing 113 malware samples randomly selected from VirusTotal. They claim accuracies of 90% and 97% for MVSS and MVPS respectively, but only used a test set of 23 samples.

Zhu et al. (c. 2020) [14] investigate the benign/malicious convictions of ~70 commercial malware detection engines on over 14,000 files for over a year via VirusTotal (www.virustotal.com), an online threat intelligence website. Zhu et al. show that using a threshold voting hueristic (i.e., if more than n of 70 vendors detect, return malicious) is quite accurate and robust to each detectors day-to-day decisions, which are volatile. Further analysis using 60 obfuscations of two known ransomware (120 zero-day malware) and 256 benign samples finds wide variation in recall and precision that is much lower than our study finds; notably, Zhu et al. continue to compare results of 36 endpoint counterparts to the VirusTotal detection engines and exhibit greater recall but worse precision in the online analogue.

By far the closest work to, and indeed the building block for this work, is Bridges et al. [2], which investigated two popular commercial host-based malware detectors—one signature-based and one ML-based—and two popular commercial network-level detectors—both ML-based, one using static analysis, and one using dynamic analysis—with a test corpus of ~3,500 files. Bridges et al. focus on comparing ML-based detectors to signature-based detectors using zero-days (as does this work), and are the first work to use tailor the general cost-benefit framework introduced by Iannacone & Bridges [8] for a file conviction experiment. Results showed detection rates in the 34-55% range, with any pair of network and host detectors achieving ~60% recall. As in this work, the signature-based detector is superior if all malware, i.e., zero-days and public malware, accrue cost identically, yet, ML-based tools win if the maximum zero-day attack costs (an input parameter) far outweigh the maximum public malware costs. Bridges et al. go on to reconfigure the cost model to simulate savings of adding a network detector to a given host-based detector concluding that (under the experiment's assumptions) adding a network malware detector will save money after the first year. Notable differences between this previous work and our current work is that Bridges et al. used only ~3,500 files, delivered over multiple protocols in a test network to enable the network-level detectors, and did not execute files to enable dynamic detection.

III. AI ATAC 1 Endpoint Malware Detection Competition

AI ATAC 1 focused on evaluating endpoint malware detectors, and required use of machine learning for the file conviction [6]. Notably, signatures can be used alongside or in a pipeline with machine learning. By "endpoint malware detector" we mean the technology should reside on a host, automatically identify the presence of a file on the host, and provide malware classification for the file.

A. Tools under test

Four commercial vendors submitted ML-based, endpoint malware detection engines to the competition, and two signature-based tools were used as baselines for comparison. For comparison, we also test two other detectors: (1) a popular commercial network-level malware detector that claims to use dynamic analysis in virtual environments to build feature vectors before applying supervised learning; (2) a pre-trained supervised learning malware detection algorithm provided by a popular cyber tech company in the form of a software development kit, using the same 100K file test corpus. Because these two tools are not host endpoint detection tools, the cost model, as configured for endpoints cannot be directly compared. Similarly, timing results per file are not comparable. On the other hand, the statistical detection results

are comparable as and are provided to gain awareness of the state of the art in practice and provide context for the endpoint detectors' abilities.

All tools were installed and configured according to and with vendor support. All ML-based tools were pretrained before submission.

B. Experimental Workflow

The experiment was designed to test the detectors' abilities when instantiated on an endpoint that receives a new file. To implement a *trial*—testing a particular detector in the presence of a particular files—a virtual machine (VM) was instantiated with a fresh Windows OS and the detection technology and configured to forward any alert to an out-of-band experiment orchestration node. The experiment orchestration node would progress through the following *stages*: (1) deliver the test file to the VM, wait an average of 90s, then (2) execute the file, wait another 90s, and (3) close the VM. Timestamps for these three events were recorded and could be compared to the timestamps of the alert logs to identify the time to detection and if the file was executed before detection happened. Further, the experiment orchestration node collected statistics on resource needs of the malware detector under test; specifically, CPU (percentage), RAM (bytes), and disk I/O (bytes read, bytes written) were programmatically collected. Network bandwidth was desired, but our experimental design was unable to isolate only the network resources used by the detector. Independent of any file, for each tool, the same resource measurements (CPU, RAM, disk I/O) are recorded for five minutes of the detection process sitting idle. This is needed as a baseline.

C. Parallelized Experimental Framework

Altogether, this experiment involved running this test workflow for 800K trials (100K files for all eight tools) with an average of 180s per trial. Implementing a repeatable experiment of this scale required orchestration software engineered to run many simultaneous, identical tests. To prevent malware infections from affecting results, per-file instantiations of virtual machines, detection tools (under test), and the resource and accuracy monitoring capabilities were needed. In order to execute all 800K trials in a serial implementation, about 4.5 years would be needed; hence a large focus of our research was on designing and implementing a software framework that choreographed the whole experiment into a completely automated, time-synced, and reproducible workflow with substantial parallelization. Here we describe this software engineering feat. With our hardware (see previous work of Nichols et al. [7] for details), the described framework running on one machine could orchestrate ~2,000 trials in parallel on a remote cluster through a web API and reduced the experimental time to 15 hours per tool. Importantly, this allowed experimental results that fit the timeline of the competition.

1) Design (with Do's and Don'ts): Our initial effort for a parallelized framework used one process to manage one VM (trial). Timing experiments were run to gain data on the performance of simply creating n such consecutive process, for $n = 1, ..., 1,000$. Notably, the growth was non-linear, seemingly quadratics, and simple regression, which yielded an extrapolated estimate of 12 hours to simply create 10,000 processes. Further experiments on the overhead latency encountered when using one-to-one relationship between processes and trials steered us to a many-trials-to-one-process design. We found using one process to orchestrate all 2,000 simultaneous trials was not appropriate either, since it took on the order tens of seconds to check the status of every trial due to compounding network latency. This led to a non-negligible amount of VMs sitting idle or powered-off at any given point in time, waiting for the orchestrator to kick off the next stage of its trial. Ultimately, we settled on a balanced approach, where 100 orchestrator processes handled ~20 trials each, mitigating the drawbacks of either extreme.

Our framework uses several **Experiment Orchestrator** objects, which run in parallel and are given a list of trials to execute with their parameters and together, run the whole experiment. The Experiment Orchestrators:

- share a **Resource Warden** object which is passed the list of resources and holds a dictionary of semaphores (each representing a resource) used to implement load balancing;
- each instantiate a **Resource Container** object which tracks, fields requests for, and releases resources;
- instantiates, in parallel, many **Trial** objects and passes the Trial objects their respective parameters along with the Resource Container, which allows the Trials to acquire or release resources if necessary;
- receives and handles **Trial Results** object upon completion of each trial;
- resets the finished Trial objects and passes them the next set of trial parameters, until the list of trial parameters is exhausted and results are obtained;
- collects performance metrics and logs them to Kafka to monitor experiment runs. A separate process consumes these logs and displays aggregate performance statistics.

Trial objects are responsible for running every stage of a trial and returning results. The Trial objects:

- contain a list of references to each **Stage** object (in order);
- run, in serial, each Stage object;
- check and store the current stage status, data, and results;
- pass a Trial Results object to the Experiment Orchestrator upon finishing the trial.

Stage objects contain methods for starting and checking on the status of individual steps in a Trial, and are not intended to store any data. The first stage to testing an individual file is to restore a VM to a live snapshot. The next step uploads a script to the VM and executes it. The script that is uploaded is executed and passed a URL to an internal webserver to download the sample of malware to be tested. The script then waits one minute to give the tool time to detect the malware statically. After one minute the sample is executed to give the tool to detect the sample dynamically. The script then waits

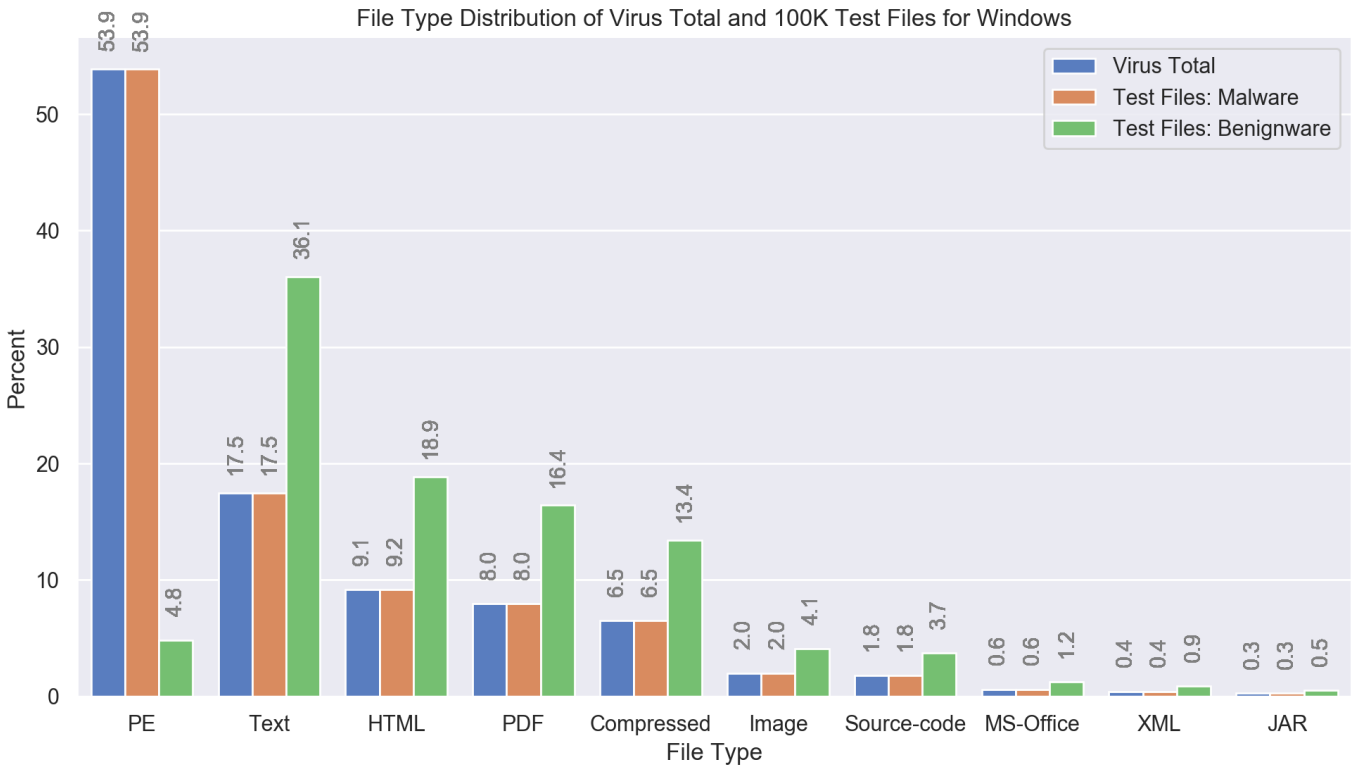

Fig. 2: Distribution of file types in the test corpus versus VirusTotal. A corpus of 100K test files (50/50% malware/benignware) was compiled for the AI ATAC 1 Endpoint Malware Detection Competition. This bar chart shows the VirusTotal distribution of file types, considered in this work to be the "real-world" distribution of file types, against the distribution of malware and benignware used in the competition.

another minute before finishing. Note that latency in execution resulted in ~90s, a 30s delay. In addition, the script collects all resource usage from each of the malware detection tool's processes and logs it to be downloaded later. In addition, exit codes, standard output and standard error from the executed malware are logged. Finally, the last Stage downloads the results files from the VM containing all the results and powers it off to save host resources until it is needed again. If it was possible, detection results were pulled from local logs and reported in the results immediately. Otherwise, they were downloaded from a management interface and merged with other results after the full experiment was run.

After scaling, this setup created some issues due to the workload being generated on the VMWare cluster. We discovered VMWare is not designed for this use case and does very minimal load balancing on its own, e.g., restoring a VM from snapshot consumes a large amount of disk I/O. Consequently, trying to restore over 2,000 VMs to a snapshot at one time generated induced stability issues and very significant degradation in performance. This is the motivation for the load balancing—the Resource Warden and Resource Container objects.

Load balancing greatly improved throughput and stability by limiting the amount of concurrent disk-heavy VM operations, but a small amount of unexpected VMWare errors still occurred. To handle the remaining errors, we created function decorators and custom **Exception** classes to catch exceptions and apply selected generic remediations. Generic responses to exceptions include:

- ignoring the exception and continuing;
- restarting the current stage;
- skipping the current stage;
- restarting the current trial;
- aborting the current trial without collecting results;
- restarting the Experiment Orchestrator object;
- aborting the experiment.

To select responses, the Experiment Orchestrator is configured by providing a list of responses to apply to exceptions, in order, at the Stage, Trial, and Exception level. For example, Stages can be configured to retry the current stage twice at the stage level. If that doesn't resolve the issue, the Trial can be configured to be restart at the Trial level. Stage and Trial exception responses reset after the successful completion of that Stage or Trial. Our configuration ignores exceptions three times before trying to reset the stage between zero and two times depending on the stage. If none of the stage responses work, Trials are configured to reset three times before aborting the Trial and throwing out results.

After adding load balancing and generic error handling, we were able to complete the testing of 100,000 files in roughly 15 hours without the loss of any results due to VMWare API Exceptions. Due to time constraints, we were unable to fully tune performance related parameters, so greater throughput

may have been achieved.

D. Samples Used

Our file corpus if driven by the previous research work of Bridges et al. [2] who tested four malware detectors using ~3,500 files. Notably many different file types are used, which led to discovery of wide variability in detection capabilities across filetypes of the four commercial detectors under test. Bridges et al. [2] also used zero-days to quantify the generalization capabilities of machine learning. Our corpus has three notable differences: (1) we scaled the experiment to 100K files; (2) we provided balanced classes (50K/50K malware/benignware); (3) we matched the filetype distribution in our test corpus to the "real-world" distribution, i.e., the distribution of filetypes found in VirusTotal.

To collect benignware, we leveraged websites such as Softpedia (https://www.softpedia.com), FileHippo (https://filehippo.com/), and the Maven (https://maven.apache.org/ [15]) repository of JARs. We automated the scraping of these websites to collect what we deemed benignware. We used these sites because their popularity, along with the popularity of the downloads they host, lends them to trustworthiness. For example, kommandotech (https://kommandotech.com/statistics/chrome-usage-statistics/) puts global Google Chrome users at 2.94 billion, with the software being updated roughly every six months. After eliminating duplicates, we have 1,049,981 benign samples. These samples consist mostly of PE, text, HTML, PDF, and compressed files. We then took a sub-sample of these files to reduce our selection to 50,000 benign files that match the VirusTotal distribution as closely as possible. Most malicious files were obtained through VirusShare[1], and a collection of about 1,000 zero-days were provided from Bridges et al. [2]; see Bridges et al. [2] for descriptions of the zero-days. After gathering our samples, we used VirusTotal (https://www.virustotal.com/) to get data on how often specific file types show up as malware. Figure 2 shows the distribution of file types from VirusTotal, our benign data set, and our malware set.

E. Scoring Framework for AI ATAC 1

In all we have four documents of raw data to be used for the cost model scoring:

- File information: A table of 100K files used for testing with filename, filetype, and label (malicious/benign) metadata;
- Measurements from the tests: A table providing for each tool and file, the experiment produced timestamps for file download and execution, the resource observations (CPU, RAM, HDD) for that file-tool test
- Alert outputs of the tool: A table curated from each tool's alert logs providing the file name of the alert, and the time-synced timestamp.
- Ambient resource observations: A table providing for each tool the ambient CPU (percentage), RAM (bytes), and HDD (bytes) used.

[1] https://virusshare.com/

For each tool, we will estimate a cost incurred for 1 host for first year of use.

Initial Costs - Each tool's subscription fees, setup, and/or hardware needed was estimated. Based on our setup experience, if a tool requires more than eight labor hours, we added that time into initial costs at a rate of $70/hour, the SOC operators fully burdened cost used in previous work [2]. Tools that require an on-premises appliance, database, etc. have a cost added to account for the added hardware, subscription, electricity, and labor during setup. We assume a 10K IP network and divide the estimated total appliance costs by 10K as we are estimating costs for 1 host.

Ongoing Resource Costs - We leverage rates provided by cloud services providers for CPU, RAM, and HDD costs. In particular, we based our rates on recent publication of Dreher et al. [16]. The resources on the host are charged cloud costs divided by 3 (to remove profits made by cloud costs), namely: $0.02444 / 3 per hour for full CPU; $.00328 / 3 per hour for 1GB RAM ; $0.05 / 3 for 1GB HDD / month $\times$ 1 month / 30.5 days $\times$ 1 day / 24 hours. As we could not measure labor for tuning tool and reconfiguring tools over time, this on-going real-world cost is not taken into account in this model. From the 5 min of ambient resource observations we compute the average ambient resource cost for a minute.

For each tool we compute the average resource cost per benign file and per malware, which are reported in the top of Table I, and will extend these linearly to an estimated cost for the year. To this end, we accrue the resource costs for every tool and for every file used in the experiment according to the costs in the preceding paragraph. We cannot use the resource measurements from the time of file execution until the VM is closed because these measurements include both the detector's and the file's resource needs. Hence, we use the resource costs from the delivery of the file until the execution, then we linearly scale the ambient resource costs for that detector to account for the resources used from the time of execution until detection. If there is a file for which the resource observations were not present in the data (small minority), we use the average ambient resource costs to impute the resource cost.

Detection Costs - As with the resource costs, we compute the detection cost from each of the 100K test files, then we scale the average files costs (reported in the bottom half of Table I) for each tool to the estimate the cost expected in the first year of use.

Detection costs account for:

- *Triage* (alerts both false positives true positives) - for every alert a triage cost is $35.05 = 0.5 hours at $70/hour (fully burdened labor cost) + $0.05 SIEM indexing fee (based on Splunk pricing);
- *Incident Response (IR)* (only for true positive alerts) - for IR we charge $140 = 2 hours @ 70/hour (fully burdened cost)
- *Attack Costs* (accounts for true positive and false negative alerts and for the time to detection) - Attack cost is an increasing function of time that seeks to estimate the cost to the organization from the malware running.

TABLE I: Average Resource Costs (Top) and Average Detection Costs (bottom) Per Malware & Benignware

	Tool 1	**Tool 2**	**Tool 3**	**Tool 4**	**Baseline 1**	**Baseline 2**
Ave Benignware Resource Cost	$ 0.002596	$ 0.002594	$ 0.013464	$ 0.013959	$ **0.002279**	$ 0.003078
Ave Malware Resource Cost	$ 0.002987	$ 0.002632	$ 0.012398	$ 0.015949	$ **0.002543**	$ 0.003276
Ave Benignware Detect Cost	$ 0.136695	$ 0.032947	$ 0.023133	$ 0.012618	$ **0.003505**	$ 0.007711
Ave Malware Detect Cost	$ 559.932	$ 533.515	$ 494.544	$ 566.961	$ 614.967	$ **353.252**

TABLE II: Cost Model Results with Subtotals (top) and Detection Statistics (bottom)

	Tool 1	**Tool 2**	**Tool 3**	**Tool 4**	**Baseline 1**	**Baseline 2**	**Network**	**Algorithm**
Initial Cost	$ 39	$ 10	$ 12	$ 8	$ -	$ -	n/a	n/a
Annual Malware Resource Cost	$ 2	$ 2	$ 7	$ 9	$ 1	$ 2	n/a	n/a
Annual Benignware Resource Cost	$ 128	$ 128	$ 665	$ 690	$ 113	$ 152	n/a	n/a
Annual Ambient Endpoint Resource Cost	$ 814	$ 439	$ 2,618	$ 2,337	$ 453	$ 842	n/a	n/a
Annual Appliance Resource Cost	$ -	$ 10	$ 2	$ -	$ -	$ -	n/a	n/a
Annual Resource Cost	$ 944	$ 569	$ 3,291	$ 3,036	$ 567	$ 996	n/a	n/a
Annual Malware Detect Cost	$ 324,760	$ 309,439	$ 286,835	$ 328,837	$ 356,681	$ 204,886	n/a	n/a
Annual Benignware Detect Cost	$ 6,755	$ 1,628	$ 1,143	$ 624	$ 173	$ 381	n/a	n/a
Annual Detect Cost	$ 331,516	$ 311,067	$ 287,978	$ 329,461	$ 356,854	$ 205,267	n/a	n/a
Total Cost	$ 332,499	$ 311,636	$ 291,279	$ 332,505	$ 357,422	$ **206,263**	n/a	n/a
Recall	0.44816	0.46750	0.50628	0.45818	0.48660	**0.64852**	0.79060	0.59954
Precision	0.99137	0.99799	0.99870	**0.99980**	0.99966	0.99922	0.99352	0.98875
F1 Score	0.61728	0.63673	0.67193	0.62827	0.65460	**0.78669**	0.88052	0.74646
Median Time to Detect (s)	33	**0**	1	55	99[1]	4	n/a	n/a

[1] This detector seemingly never began to analyze a file until the files was executed. Given the execution time was on average 90s after the file was introduced (and given linearity of expectation), this detector used on average 9s to analyze and make a decision on a file, once the file was executed. Bold figures indicates the best of the six head-to-head tested tools in Total Cost, Recall, Precision, F1, whereas underlined figures indicate the network-level detector (Network) or supervised learning algorithm (Algorithm) outperformed all others.

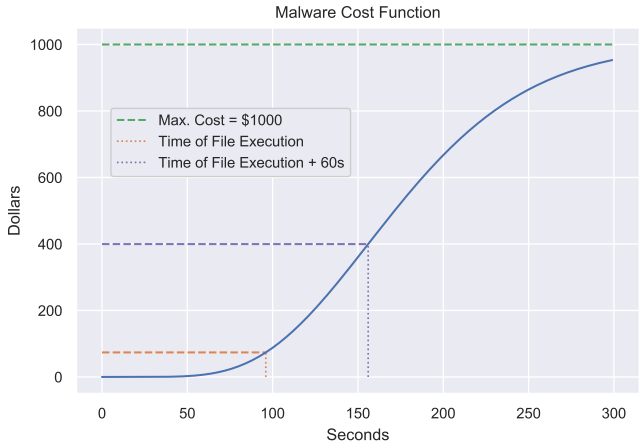

Fig. 3: Attack Cost S Curve

We model the per-malware attack cost using an S curve (as in previous works [2, 8]), but using a different formulation tailored for our needs. Let t_0 be the malware download time and t_e the execution time of the malware (both measured/recorded in our experiment). Our goal is to identify an S curve $f(t; t_0, t_e)$, that has value 0 for $t < t_0$, is positive beginning at $t = t_0$ but remains small, starts increasing quickly at execution time, $t = t_e$, hits its inflection at one minute post execution, $t = t_e + 60$, and approaches a maximum cost of \$1,000 (horizontal asymptote M =\$1000). To find this curve, note that all CDF (cumulative distribution functions) with sample space $[0, \infty)$ are S shaped increasing functions that begin at $(0, 0)$ and approach horizontal asymptote $y = 1$. Given our desires above, we select the CDF of the Γ distribution $f(t)$ so $f'(t) = c\ t^{\alpha-1} \exp(-\beta t)$, and shift by t_0 (replace $f(t)$ with $f(t - t_0)$). Then we solve for α and β with three constraints: (1) t_e is the first root of f''', forcing the curve to be small and only begin to grow at execution time; (2) $f''(t_e + 60) = 0$ forcing the curve to be accruing cost most rapidly a minute after execution. The solution gives $\alpha = (1+60/t_e)^2+1, \beta = (\alpha-1)^2/60$. Finally, multiplying f by 1000 ensures $\lim_{t\nearrow\infty} f = 1000$, forcing $\$1,000$ to be the max accrued cost. See Figure 3 for an example. This curve is fit to each malware file experiment as the execution time, t_e does vary per experiment. Although we held \$1,000 max cost constant, as suggested in Bridges et al. [2], one could also change the max cost (\$1,000) per malware to simulate different scenarios.

If a malware sample is detected before execution, i.e., in time $t \in [t_0, t_e)$ attack cost is simply $f(t; t_0, t_e)$ (near 0) and no triage or IR costs are incurred, as we assume the endpoint client quarantine's the file. Else, if the malware is detected in time $t \in [t_e, t_e + 60]$ (from execution to a minute after) the attack cost is $f(t; t_0, t_e)$ + Triage + IR. If the VM closed and the detection technology did not yet alert we charge the max cost, $1000 = \lim_{t\to\infty} f(t; t_0, t_e)$. Note that in some cases alert logs were received after the VM closed, as technologies that

leverage a central appliance could may have been processing info from the file. In this case we charge C + Triage + IR where $C = \texttt{ave}\,(f(t_e+60; t_0, t_e)$, 1000 - Triage - IR)). Note this is higher than a detection cost at $f(t_e)$, but less than the max cost $1,000$ as the detector, given more time may have eventually alerted. This is designed so that the overall cost (attack + IR + triage costs) for detection after the VM closes the experiment is strictly more than during the VM and strictly less than never detecting. For each tool, after computing the per-malware cost incurred, the average cost per malware is computed.

If a benignware is detected (false positive) we charge a Triage cost; else (true negative) no cost is incurred. For each tool, after computing the per-benignware cost, the average cost per benignware is computed.

Total Cost: The total cost is the sum the initial cost, ongoing resource cost, and detection cost estimates for the first year of use. The average resource and detection costs per malware and benignware, as reported in Table I, are each linearly scaled to 50,000 files—our estimate for the number of files a host detector sees in its first year—with 1.16% malware, 98.84% benignware. This overall number of files (50,000) was informed by SOC operators who pulled endpoint detector logs from ~30 IPs for a full calendar year. The 1.16/98.84% malware/benignware ratio follows the previous cost model scaling ratio of Bridges et al. [2] and is based on real-world findings in the empirical research of Li et al. [17]. These annual estimates are lines "Annual Malware Resource Cost", "Annual Benignware Resource Costs", "Annual Malware Detect Costs", and "Annual Benignware Detect Costs".

Note that the the "Annual Malware and Benignware Resource Costs" account only for the resource costs of the detector when a file is present. To account for the whole year, we simply use scale the ambient resource cost estimate observed for this tool to the the remaining time in the year. This is line "Annual Ambient Endpoint Resource Cost". Resource Appliance costs are also included

Because precision is so close to perfect for these tools (as we shall see), the overall costs scale approximately according to an affine function ($mx + b$, here x is the number of files seen or equivalently time in this model) with the slope m an increasing function of the observed recall, the detection latency, and the max attack cost. Over a long enough time (or after encountering enough files) those tools with the best recall and time to detect will prevail.

IV. AI ATAC 1 Results & Discussion

This section provides the results of the endpoint malware experiment. We present the summarized and itemized takeaways on the findings in the introduction, Section I-4.

First note that the initial and resource costs pale in comparison to the detection costs, indicating that this cost model differentiates tools based on their detection statistics—precision, recall, and detection time. Precision is near perfect for all tools, which is reflected in the low detection costs incurred from benignware, especially considering that our simulated ratio of files is 98.84% of the files in the simulation. Consequently, recall and time to detect are the two measurements that differentiate these detectors.

As seen in the bottom of Table II, Tools 1-4 (all four ML-based endpoint detection tools), and Baseline 1 (signature-based endpoint detection tool) achieve recall ~45%, whereas Baseline 2 (the second signature-based endpoint detector) has recall near 65%. This combined with the face that Baseline 2 has relatively fast detection time (4s) drives the best results in the cost model, and it can been seen beginning in Table I, where Baseline 2 has the lowest Average Malware Detection Cost ($353/malware) leading to the lowest Annual Malware cost ($204K = $ $353 \times 50,000$ files/y $\times$ 1.16 malware /100 files). Second place in this cost simulation is Tool 3, which leveraged an on-premises appliance in coordination with the endpoint clients. While this increases the resource costs, it pays off in this cost model as the recall of ~50% and low detection time led to a good cost score.

Median detection time seems to be a differentiator of these tools, as times vary over two orders of magnitude. This has an imprint in the Annual Resource Cost, save noting that the anomalously high resource costs of Tool 3 is due to its use of an on-premises appliance.

Next, considering the Network dynamic, ML-based detector, we see an enormous increase in recall to ~79% with no sacrifice to precision. This shows the enormous gain in detection abilities provided by this tool. This network tool was evaluated in previous work [2] for detection latency, where authors found "While seeing data rates of up to 7 GB/s, the [tool] achieved median detection [time] from file delivery until alert of ... 258 s = 4.3 m." Hence, the increase in detection abilities comes at the cost of both latency, and various drawbacks associated with network-level detection, in particular, need for decrypted traffic and files, the inability to immediately quarantine files on hosts, and varying detection abilities depending on protocol.

The "Algorithm" column, representing results of a supervised ML algorithm submitted by a cyber technology vendor, also recall of ~60%, and also near perfect precision. We can report that latency between submission of a file to the algorithm and receiving the detection results had median 1s, but we note that this is not a direct comparison to any other tools, as our experiment required Tools 1-4 and Baselines 1,2 to automatically identify the file in a running VM, then analyze it.

1) ML versus Signature-Based Detection: Our dataset includes ~1000 zero-day (i.e., never before seen) malware, all program executables and ~26,000 Public Malware PEs. Approximately 23% of these zero-days were not detected by any tool in this experiment, whereas only 2% of the public PEs were not detected by any tool. Table III provides many statistics comparing the six ML-based detectors to the two signature-based detectors on only the zero-day and separately the public PE files. As indicated by the recall on zero-day files, signature-based tools identified ~4%, whereas the ML tools' recall was in the 28-60% range! Our results also confirm that detection latency is not affected by zero-day files. Finally,

TABLE III: Machine Learning Versus Signature-Based Detection Statistics

Statistic	Samples	ML-Based						Signature-Based	
		Tool 1	Tool 2	Tool 3	Tool 4	Network	Algorithm	Baseline 1	Baseline 2
Recall (% Malware Detected)	Zero-day	45.5%	40.1%	38.1%	27.8%	59.4%	59.7%	3.8%	4.4%
Recall (% Malware Detected)	Public P.E.	80.7%	87.9%	59.8%	71.0%	90.8%	93.6%	68.2%	76.6%
Time to Detect Median (s)	Zero-day	36	0	0	47	-	-	97	5
Time to Detect Median (s)	Public P.E.	33	0	1	50	-	-	99	5
Malware no other tool detected (#)	Zero-day	1	0	35	0	86	61	0	0
Malware no other tool detected (#)	Public P.E.	24	16	9	8	54	468	2	11

TABLE IV: Recall for each Filetype

Filetype (# malware)	Tool 1	Tool 2	Tool 3	Tool 4	Baseline 1	Baseline 2	Network	Algorithm
Compressed (3,252)	31.55%	0.37%	31.98%	2.31%	5.32%	44.59%	51.66%	33.06%
HTML (4,575)	0.00%	0.00%	44.85%	8.07%	0.00%	64.90%	73.11%	0.00%
Image (986)	0.00%	0.00%	42.29%	7.00%	66.53%	5.58%	68.76%	0.00%
JAR (127)	0.00%	0.00%	5.51%	0.00%	0.00%	0.00%	40.16%	0.00%
MS-Office (299)	0.00%	49.16%	51.51%	47.83%	54.85%	81.94%	71.57%	79.26%
PDF (3,977)	0.00%	0.00%	59.67%	33.77%	0.00%	92.33%	96.18%	95.35%
PE (26,930)	79.40%	86.15%	58.98%	69.38%	66.97%	73.90%	89.66%	92.36%
Source-code (905)	0.00%	0.00%	31.38%	12.93%	48.07%	49.94%	54.14%	0.00%
Text (8,738)	0.00%	0.00%	34.25%	23.85%	59.21%	40.59%	56.50%	0.00%
XML (211)	0.00%	7.11%	53.08%	11.85%	0.00%	63.51%	78.20%	0.00%

we report an interesting statistic for each tool, the number of malware no other tool detected, showing that Tool 3, the Network, and the Algorithm columns have good ability to complement other detectors. Our main takeaway from Table III is that ML-based tools are much better for detecting never-before-seen malware.

We note that the cost model, as configured made no accommodation for zero-day vs. public malware, so this benefit to ML-based tools is not reflected in the costs. As shown in previous work [2], increasing the maximum attack cost for only the zero-day files penalizes false negatives and late detection of zero-days over that of public files. For this experiment, the percent of malware that is zero-day, or equivalently, the number of zero-days expected in a given year is needed for inclusion of an increased attack cost for zero-day files. As a an example, if we suppose 1% of malware are zero-days (giving 6 zero-day malware, 572 public malware, and 49,420 benignware per year) and increase the maximum attack cost from $1,000 for public malware to ~$35,500, then the ML-based tools will begin to surpass both Baseline tools in the cost simulation.

2) Results per Filetype: Bridges et al. [2] have provided the first emperical study to show that commercial malware detectors vary dramatically in detection abilities across filetypes [2], with a trend to excel in recall on PE files while yielding relatively poor recall on other filetypes. We present detection rates per filetype for all eight tools on our much larger corpus in Table IV and our evidence supports previous findings. Interestingly Tool 1 seems to only support PE and compressed filetypes while Tool3 and the Network tool have relatively good recall on nearly all filetypes. In short, this is a large discriminator for modern malware detection technologies as efficacy varies wildly.

A. Drawbacks and potential future research directions

Although we achieved a large and balanced test corpus, conditioned on filetype the malware-benignware balance was skewed. This is due to our inability to find enough benign files of certain types.

It is unclear how well our eight tested technologies represent the whole commercial market space of endpoint malware detectors. Notably, there was diversity in the types of detectors used, namely, two signature-based detectors, multiple that claimed to *only* use supervised ML (as opposed to those that leverage both a library of signatures and ML), one dynamic (albeit network-level detector), and one that leveraged a central server to assist decision making by the per-host clients.

No effort was made to ensure the malware would execute correctly during the dynamic phase of our test.

Because no internet collection was allowed in this experiment, promising and popular detectors that leverage a cloud connection for processing difficult files were not tested with their cloud connections. Lack of internet may also hurt the rate at which malware samples execute correctly.

No effort was made to validate the labels on our samples from expected malicious/benign sources. Therefore, due to the size of the dataset, it's possible on rare occasions that detectors were penalized for correctly detecting mislabeled data. We plan future analysis of our dataset to gain further insight into its true composition.

While each input cost/rate to the model is believable, the final numbers seem high. There are a number of reasons that may cause inflation of the total cost. Most notably, attack cost is hard to estimate. Many malware if undetected will cost nothing (e.g., those that fail to execute or succeed in, say, changing one's homepage) while some tiny minority of files may cause unbounded (and unestimatable) costs.

We vary the maximum attack cost parameter below to see how it affects the final costs of the tools. Tools have roughly

50% recall on average. The consequence is they incur max attack cost ($1000) for about 250 (50,000 files/year $\times$ 1.16/100 malware) which costs $250K! In addition, the remaining half of detected malware will accrue some smaller costs. This cost estimate, while perhaps accurate for the first host in the network affected by malware, likely would not scale to every host in the whole network; rather, once-seen malware would be automatically blocked using modern endpoint detection configurations. This means the cost model as configured is a worst case scenario in that it assumes novel malware per host, which is unlikely. Perhaps the malware in this experiment is harder to detect than in the real world, in particular, the percent of zero-days is 1,000/50,000 = 2%, which may be higher than in the wild. A large scale data-driven study to understand some reasonable estimates of rates of zero-day attacks would be useful.

In a real SOC, the cumulative IR and triage costs are limited by logistics, namely the time available for triage and IR by operators'; yet, our cost model does not take this into account. While the IR and triage rates are (we believe) good estimates for a single investigation, they may not be reasonable for influxes, i.e., a flurry, of alerts; in this case, likely many alerts would simply go unfilled without the operators needed to investigate all of them. Further, it is possible that many similar alerts would possibly be handled in bulk, leveraging automation capabilities of modern endpoint tools (or SOAR tools). In short, the cost model linearly adds a fixed amount for each alert/TP, but this may not be realistic in that it overestimates costs in some scenarios. Operations or perhaps qualitative research of SOCs to understand and model non-linearity of their processes is needed.

All together, these cost-model drawbacks are particular instances of the overarching problems with cost-benefit analyses, namely, untenable assumptions are needed to model real-world processes for which we have little data. Nevertheless, they provide a quantitative way to integrate and reason about many different statistics. Further, by illustrating their benefits and drawbacks we, hopefully, pave the way for refining the models to gain accuracy.

Acknowledgements

The research is based upon work supported by the Department of Defense (DOD), Naval Information Warfare Systems Command (NAVWAR), via the Department of Energy (DOE) under contract DE-AC05-00OR22725. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the DOD, NAVWAR, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

[1] A Not-So-Common Cold: Malware Statistics in 2022. [Online]. Available: https://dataprot.net/statistics/malware-statistics/#:~:text=560%2C000%20new%20pieces%20of%20malware,58%25%20of%20all%20computer%20malware

[2] R. A. Bridges, S. Oesch, M. E. Verma, M. D. Iannacone, K. M. Huffer, B. Jewell, J. A. Nichols, B. Weber, J. M. Beaver, J. M. Smith et al., "Beyond the hype: A real-world evaluation of the impact and cost of machine learning-based malware detection," arXiv preprint arXiv:2012.09214, 2020.

[3] MITRE, "ATT&CK Evaluations, APT 3," https://attackevals.mitre-engenuity.org/enterprise/evaluations.html?round=APT3, 2022, 2021-05-23.

[4] AV-TEST, "The best Windows antivirus software for business users," https://www.av-test.org/en/antivirus/business-windows-client/windows-10/april-2022/, 2022, 2022-06-23.

[5] SE Labs, "Endpoint security (EPS): Enterprise 2022 Q1," https://selabs.uk/reports/enterprise-endpoint-protection-2022-q1/, 2022, 2022-06-23.

[6] AI ATAC 1 Endpoint Malware Detection Challenge (website). [Online]. Available: https://portal.challenge.gov/public/previews/challenges?challenge=eccedd04-0c23-434e-9363-5b77266b64f4&print=true

[7] J. Nichols, K. Spakes, C. Watson, and R. Bridges, "Assembling a cyber range to evaluate artificial intelligence/machine learning (ai/ml) security tools," in ICCWS 2021 16th International Conference on Cyber Warfare and Security. Academic Conferences Limited, 2021, p. 240.

[8] M. D. Iannacone and R. A. Bridges, "Quantifiable & comparable evaluations of cyber defensive capabilities: A survey & novel, unified approach," Computers & Security, p. 101907, 2020.

[9] M. Christodorescu and S. Jha, "Testing malware detectors," ACM SIGSOFT Software Engineering Notes, vol. 29, no. 4, pp. 34–44, 2004.

[10] S. K. Pandey and B. Mehtre, "Performance of malware detection tools: A comparison," in Advanced Communications, Control and Computing Technologies. IEEE, 2014, pp. 1811–1817.

[11] Ö. Aslan and R. Samet, "Investigation of possibilities to detect malware using existing tools," in IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA). IEEE, 2017, pp. 1277–1284.

[12] W. Fleshman, E. Raff, R. Zak, M. McLean, and C. Nicholas, "Static malware detection & subterfuge: Quantifying the robustness of machine learning and current anti-virus," in 2018 13th International Conference on Malicious and Unwanted Software. IEEE, 2018, pp. 1–10.

[13] M. Abdelsalam, R. Krishnan, and R. Sandhu, "Online malware detection in cloud auto-scaling systems using shallow convolutional neural networks," in IFIP Annual Conference on Data and Applications Security and Privacy. Springer, 2019, pp. 381–397.

[14] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online anti-malware engines," in USENIX Security), 2020, pp. 2361–2378.

[15] Maven - repositories. [Online]. Available: https://www.tutorialspoint.com/maven/maven_repositories.htm#:~:text=What%20is%20a%20Maven%20Repository,be%20used%20by%20Maven%20easily

[16] P. Dreher, D. Nair, E. Sills, and M. Vouk, "Cost analysis comparing HPC public versus private cloud computing," in International Conference on Cloud Computing and Services Science. Springer, 2016, pp. 294–316.

[17] B. Li, K. Roundy, C. Gates, and Y. Vorobeychik, "Large-scale identification of malicious singleton files," in Proceedings of the 7th Conference on Data and Application Security and Privacy. ACM, 2017, pp. 227–238.