SAND2022-2639C

WiP: Verification of Cyber Emulation Experiments Through Virtual Machine and Host Metrics

Jamie Thorpe and Laura P. Swiler and Seth Hanson and Gerardo Cruz and Thomas Tarman
{jthorpe,lpswile,shanson,gcruz,tdtarma}@sandia.gov
Sandia National Laboratories
Albuquerque, New Mexico, USA

Trevor Rollins and Bert Debusschere
tkrollins6@gmail.com,bjdebus@sandia.gov
Sandia National Laboratories
Livermore, California, USA

ABSTRACT

Virtual machine emulation environments provide ideal testbeds for cybersecurity evaluations because they run real software binaries in a scalable, offline test setting that is suitable for assessing the impacts of software security flaws on the system. Verification of such emulations determines whether the environment is working as intended. Verification can focus on various aspects such as timing realism, traffic realism, and resource realism. In this paper, we study resource realism and issues associated with virtual machine resource utilization. By examining telemetry metrics gathered from a series of structured experiments. These experiments involve large numbers of parallel emulations meant to oversubscribe resources at some point. We present an approach to use telemetry metrics for emulation verification, and we demonstrate this approach on two cyber scenarios. Descriptions of the experimental configurations are provided along with a detailed discussion of statistical tests used to compare telemetry metrics. Results demonstrate the potential for a structured experimental framework, combined with statistical analysis of telemetry metrics, to support emulation verification. We conclude with comments on generalizability and potential future work.

CCS CONCEPTS

• **Security and privacy → Formal methods and theory of security**.

KEYWORDS

cyber experimentation, system emulation, model verification

ACM Reference Format:
Jamie Thorpe and Laura P. Swiler and Seth Hanson and Gerardo Cruz and Thomas Tarman and Trevor Rollins and Bert Debusschere. 2022. WiP: Verification of Cyber Emulation Experiments Through Virtual Machine and Host Metrics. In *Proceedings of Hot Topics in the Science of Security (HotSos2022).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/XXXXX.123456

HotSos2022, April 2022, Virtual Conference

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/XXXXX.123456

1 INTRODUCTION

Cyber networks are commonly modeled using one of two approaches: network simulation or network emulation. In network simulation, such as the discrete-event simulator `ns-3` [5] [15], the underlying processes, components, and their interactions are simulated in software code. In network emulation [11], actual software binaries (e.g. operating systems and applications) are run on virtualized hardware, offline and separated from real operational systems. Because emulation models (hereafter called "emulations") run the same binary software found in real systems, they provide a more realistic platform for assessing cybersecurity issues and their impact on other components in the system. However, this additional realism comes with an increased cost. While network simulators can "control the clock" and simulations can run faster than real-time, emulations need to run in actual clock time. Additionally, modeling large networks with emulation requires hundreds or thousands of virtual machines (VMs) which typically have to be managed over multiple nodes of a supercomputer. A number of cyber emulation testbeds have been developed to provide a platform for test and evaluation, cyber security investigation, research, and training. [16, 25]. Such testbeds include LARIAT [24], Emulab [26], DETER [8, 21], and DARPA's National Cyber Range [12]. Experimental frameworks such as DEW [20] and SCORCH [13] have been developed to run structured experimental scenarios on these testbeds.

An important consideration when using emulation is determining whether the emulation environment is working as intended, also called *verification* [1] [23]. Verification of systems in general typically involves software testing and quality assurance. A unique aspect of cyber emulation in particular is assessing the performance of the experiment in the virtualized environment and determining whether there are sufficient resources available to run the experiment properly. If there are not, the virtualized components may produce experimental artifacts that result in the outcomes being incorrect or unrepresentative of the system being modeled.

While there are many aspects of cyber emulation that could be verified, in this paper, we focus on determining whether there are sufficient physical resources to support the emulation experiment. To this end, the goal of this study is two-fold: (1) develop and exercise a process for verifying cyber emulation environments, and (2) identify metrics that can indicate when there are insufficient resources to reliably run an emulation.

In this work, we refer to these metrics as *telemetry metrics*, following the usage of this phrase from Google [6], Microsoft [3], Intel [4] and others [7]. These companies use telemetry in the context of network monitoring metrics (e.g. monitoring traffic to and from VMs, round trip time for TCP flows [6]), virtual machine

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

resource usage (e.g. number of system processes, thread counts, physical disk read/write time [3]), and application monitoring (e.g. CPU utilization). A formal definition from Intel states "Telemetry refers to monitoring and analyzing information about IT systems to track performance and identify issues." [4] Sumo Logic adds "Telemetry data is used to improve customer experiences, monitor security, application health, quality, and performance." [7]. In this paper, we focus on telemetry metrics relating to the performance of virtual machines which are used in a cyber emulation study and the physical machine hosting that study.

We will address resource verification specifically for an emulation tool called `minimega` [19]. This study was performed on two different scenarios, described later in this paper. These scenarios are specific examples meant to demonstrate our proposed general approach to resource verification. While there has been some related research in emulation verification and validation, there is generally no common framework, set of standards, or evaluation metrics to use when setting up an emulation; it is left to the developer and/or analyst running the emulation model to examine experimental artifacts and decide whether the emulation ran correctly. We hope to make this process more consistent and objective by providing a process for resource verification and a method for identifying metrics which can be used to detect anomalous experimental results.

The organization of this paper is as follows: Section 2 provides an overview of related research. Section 3 summarizes our approach to investigating telemetry metrics for emulation verification. Section 4 outlines the experimental configurations used in both scenarios of this study. Section 5 presents a discussion of statistical metrics used to compare telemetry metrics. Section 6 presents the results for both scenarios. Section 7 discusses future work.

2 RELATED WORK

While the focus of this work is verification, it is also important to address *validation* of emulated environments. Validation addresses the question: is the behavior of the emulated model "close enough" to the behavior of the physical system to be an acceptable and adequate representation? Crussell et al. investigate this question [10], identifying behavior at three levels of abstraction: application, operating system, and network. A unique aspect of their work was the use of Markov models to compare patterns of system call orderings. While we acknowledge validation is also an important area of research, the focus of our study is resource verification.

In his Ph.D. thesis [14], Brandon Heller developed the idea of "network invariants." These are properties that can be used to verify timing errors in the emulation. The network invariants should be universal and apply regardless of the experiment being run, the topology of the experiment, the platform upon which the emulation is run, or the emulation code being used. In this paper, we expand upon Heller's idea of network invariants in order to develop telemetry metrics. We choose to use telemetry metrics as defined above instead of network invariants because performance and health monitoring metrics will be more generalizable to larger emulation scenarios that are not bandwidth limited.

It is also important to consider whether periodic polling to track the metrics, a form of "health monitoring", affects the behavior of the emulation itself by using resources. Jia et al. [17] referred to this issue as a weak form of the Heisenberg uncertainty principle for measuring host resource usage. They studied virtualized environments where hundreds of VMs were running on one physical host. They concluded that frequent polling (sampling intervals between 0.001 and 0.1 seconds) did affect the performance of the emulated environment. In the work we present in this paper, the polling was much slower, at 5 or 10 second intervals, and our experimentation suggests this frequency of polling did not introduce artifacts into the results.

3 SUMMARY OF APPROACH

The first goal of this study is to develop and exercise a process for resource verification. The process we developed is as follows:

(1) Plan a series of experiments to stress the physical resources of your system. In this paper, we refer to a series of these experiments as "runs". Each run is defined by a different emulation configuration that will put different amounts of stress on on physical host. We achieved this by running increasing numbers of experiments in parallel. Within each run, a certain number of "replicates" will be executed in serial. A replicate is a single iteration of the emulation experiment.
(2) Collect telemetry data from the replicates. The platform may already collect and return experiment results ("quantities of interest") for each replicate. These results help us answer some question about the scenario being emulated. The telemetry metrics will help us to answer questions about the emulation itself, and to verify that the emulation had sufficient resources to run.
(3) Perform the planned experiments using emulation settings that are increasingly likely to generate emulation artifacts. For this study, we wanted to purposefully push the physical resources to oversubscription, forcing a high likelihood of emulation artifacts in the resulting data. This allowed us to address the secondary goal of the study, which was to develop indicators that could help detect unreliable experiments run in oversubscribed conditions.
(4) Observe changes to the emulated scenario's quantity of interest as resource conditions vary. If the quantity of interest changes drastically as more and more resources are utilized on the physical host, then we know that there are likely emulation artifacts affecting the results.

In addition to establishing this verification process, the second goal of this study is to identify telemetry metrics that can indicate when the results of an experiment are likely to be unreliable. In order to achieve this, we define indicators as a combination of a telemetry metric and a threshold that that metric should not cross. We wish to identify indicators where violation of the given threshold during an experiment is highly correlated with unreliable results in the quantity of interest for that experiment. We hypothesize that this indicator can then be used to detect and filter experiments with unreliable results even when the level of resource subscription is not known.

We evaluate our approach using two scenarios, described further in Sections 4.2 and 4.3.

4 EXPERIMENTAL SETUP

4.1 Emulation Infrastructure

- **minimega and SCORCH**
  The emulation platform we used to run the emulations is `minimega` [19]. We also leveraged SCORCH, a framework that handles scenario orchestration for `minimega`. SCORCH allows specific emulation experiments to be defined in a highly modular way. [13]. Modules called "components" define a variety of aspects of the experiment, including the network topology, the scenario (or attack) being run, and any added tools for data handling. These components can also be configured to feed resulting data directly to a database for later analysis.
- **Over-Committing Resources**
  We want to put increasingly more strain on the physical resources available to the emulation experiments. This will not only help test our verification process, but it will give us sufficient data to develop telemetry-based indicators of unreliable experiments. We oversubscribe resources by forcing the physical host to perform increasingly more work in parallel. When running an experiment using SCORCH, the experiment is part of a "namespace". This namespace can be thought of as an isolated copy of the experiment environment, as specified by the scenario being run. Multiple namespaces can be started in parallel. Each will have its own copy of the experiment environment and be able to run its own series of experiments without affecting the actions of the other namespaces. For the purposes of this study, we run several iterations of the same experiment on an increasing number of parallel namespaces. By increasing the number of namespaces, we hope to reach a point of resource oversubscription.
- **Data Collection**
  In order to establish telemetry metrics to indicate unreliable experiments, data must be collected from the emulations. SCORCH components can be configured to parse data and input it directly to a database. Some components that construct the emulated scenario already report data in this way. Such components may be extended to report additional data by adding configurations to parse existing data in a new way. Alternately, it may be useful to design a SCORCH component specifically for collecting data that is not already collected. These components run some command at a configured rate during the experiment and output results that can be parsed. For this study, collected data included telemetry from the emulated VMs and the physical host running the emulations, as well as the results, or "quantity of interest" of the experiment scenario.

4.2 Scanning and Detection Scenario

The scanning and detection scenario is extensively documented in [28] and [27]. It consists of an attacker conducting reconnaissance using a scanning tool called `nmap` and is detected by the intrusion detection system called `snort` [9].

The quantity of interest for this scenario is the time to detection, or "alert" time. There are two stochastic elements that can be modeled for this scenario: port scan order and rate of packet dropping. As we are interested in the effects of the over-subscription of resources on the results, both of these modes of stochasticity are removed – the packet drop rate is set to zero and the port scanning order is fixed. We then run 400 total replicates and try to capture the variability in the quantity of interest.

4.3 Command and Control Scenario

The Command and Control scenario [29] emulates a network under attack by a piece of malware with a command and control communication pattern, such as Emotet [2]. This communication involves a malicious actor leveraging a covert communication channel, such as the "cookies" field in HTTP requests, to communicate with infected hosts within a network. A `snort` intrusion detection system on this network attempts to detect and alert to this malicious communication.

The quantities of interest for this scenario are the number of alerts received by time `t = 1`, `5`, `10`, and `16` seconds. Because this scenario has multiple quantities of interest, the statistical analysis will differ slightly from the analysis of the Scanning and Detection scenario, as will be further discussed in Section 5.2.

The Command and Control scenario was emulated using a simplified network topology. This reduced emulation complexity while still allowing us to track the number of alerts over time. The emulated network consisted of a VM to generate background traffic, a VM to generate malware traffic, a single traffic server VM, and a single VM to host `snort`. There were several tunable parameters for this scenario, but all parameters were set to constant values for the duration of the verification study. There were 1000 benign packets per second, and 1 in 1000 packets contained the malware signature that `snort` would identify. There were 20 malicious packets per second, and 3 in 20 (150 in 1000) contained the malware signature.

5 METHODS

5.1 Statistical Comparisons of Experimental Results

As discussed in the scenario descriptions above, each emulation scenario has a quantity of interest that is being monitored (e.g. time to alert in the scanning and detection scenario). The telemetry metrics such as CPU utilization and system load are also being monitored as a function of time (e.g. recorded every second). The distribution of the quantity of interest is used to compare similarity of experimental outcomes at each namespace case. The assumption is made that the 1-namespace case should be the expected results of these experiments (as these are least likely to oversubscribe the physical resources), and the distribution of the 1-namespace quantities is taken as the ground truth. Two statistical comparisons are the Tukey multiple comparison test and the ratio acceptance test as described below. Additional statistical tests will be the subject of future work.

5.1.1 Tukey Multiple Comparison Test. Tukey's multiple comparison test is used to determine which means amongst a set of means differ from the rest. A mean value in this context is the average of a number of experimental replicate results. For example, the Tukey test allows us to compare the alert time mean at each namespace

case to all other namespace cases. That is, for a set of r means, the Tukey test considers all possible pairwise differences:$\mu_i - \mu_j$ for $i \neq j$ and $i, j = 1 \ldots r$. For each pair, the test checks the following confidence limits to see if these lower and upper bounds encompass zero. If they do, the differences in means are not significantly different; if the bounds don't cover zero, the means are considered different:

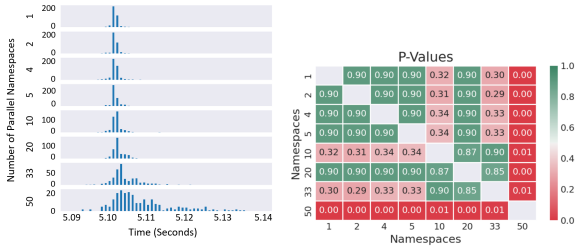
$$\overline{y_i} - \overline{y_j} \pm \frac{1}{\sqrt{2}} q_{\alpha;r,N-r} \hat{\sigma} \sqrt{\frac{2}{n}} \quad (1)$$

In this formula, $\overline{y_i}$ is the mean of group i, $q_{\alpha;r,N-r}$ is the studentized range value for a confidence interval of $1-\alpha$ with r observations (r means) and N degrees of freedom, and $\hat{\sigma}$ is the joint estimate of the standard deviation. Note that the Tukey test assumes that the groups associated with each mean are normally distributed and it also assumes that the within-group variance is the same across groups. The typical version assumes each group has the same number of samples n but there is a variation that allows for unequal groups. Finally, it assumes that the observations of the groups are independent. The Tukey test is basically a version of the t-test for comparison of means across multiple groups.

The Tukey test produces a p-value for each comparison, with a high p-value indicating that the null hypothesis: $\mu_1 = \mu_2$ cannot be rejected. Thus, a high p-value indicates that our distributions of alert times are similar, indicating that the experiment produced comparable results in the two cases. The shortcomings of this statistical test are that it tends to be overly generous when there are few samples, and overly strict when there are many samples combined with a small variance. We used the Tukey test because it is a robust multiple comparison tests. A recent survey [18] states: "Tukey's...presents a robust and widely available test for a variety of situations." We note that there are several non-parametric tests which relax the assumption of normal distributions, such as the Mann-Whitney-U test, the Dunn test, and others [22]. In future work, we plan to investigate these non-parametric versions.

An example of how the alert times change in the scanning and detection scenario is shown in Figure 1. The set of histograms in subplot (a) show the systematic change in the alert time distribution as the number of namespaces increases from one to 50. The 50 namespace experiment has a wider, longer tailed distribution than the one namespace distribution and the mean is also larger. The matrix of Tukey test values in subplot (b) of Figure 1 shows that the alert times in the tests run on 20, 33, or 50 namespaces would be considered significantly different, as evidenced by the small p-values highlighted by the red colors. Thus, the Tukey test serves to quickly summarize differences in mean alert times across varying namespaces. Note that p-values appearing as "0.00" in the Tukey figures throughout this paper indicate a non-zero p-value which is less than 0.005.

5.1.2 Ratio of Acceptability. To augment the Tukey test, a second metric, which we call the Ratio of Acceptability (RoA), is utilized. We define "acceptability" as the range of values for the quantity of interest of the middle 95% of samples in the 1-namespace case, as seen in Figure 2. This means that if the quantity of interest of an experimental run falls into this range, the replicate is deemed acceptable. The RoA is then defined as the number of samples that are acceptable divided by the total number of samples. The RoA goes beyond the mean comparison of the Tukey test: it indicates the percentage of samples falling within a central 95% region of the benchmark distribution.



(a) Alert time distribution at each namespace. **(b) Tukey multiple comparison.**

Figure 1: Overview of all scanning and detection emulation runs.

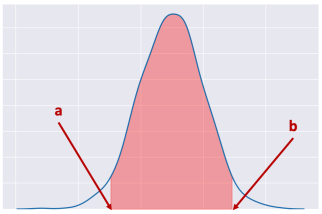


Figure 2: This plot shows how the upper and lower limit for the range of acceptable values is calculated. The shaded area under the curve represents the middle 95% of alert times for the 1-namespace case. (a) shows the lower bound of this range, and (b) represents the upper bound.

5.2 Developing Telemetry Metrics

In addition to establishing a verification process, the second goal of this study is to identify telemetry metrics that can serve indicators of unreliability for an emulation experiment. In its simplest form, defining these indicators involves selecting telemetry that can be extracted from the emulation and thresholds that the telemetry should not cross for the duration of a given experiment. If thresholds are crossed, then we believe the results of the experiment to be unreliable.

In practice, determining a good indicator can be done one of two ways: from subject matter expert (SME) input, or through extensive experimentation.

There are certain telemetry that have intuitive thresholds that shouldn't be crossed. One instance might be stolen cycles, which is a measure of the time a VM has to wait for available resources. A SME might suggest that any experiment for which there are any stolen cycles is one that should not be trusted, because the amount of time that a VM waits for resources will affect the timing-based results of the experiment. But we might still ask: Is this always the case? Could an experiment still give reliable results with some

stolen cycles? Answering these questions may be very difficult based only on SME opinion.

For this verification study, we turn to experimentation to help us determine good indicators. As previously described, we define a range of acceptable quantities of interest, determined by some baseline set of experiments. In order to identify a good indicator, we want to find a telemetry metric and threshold such that the set of experiments that cross this threshold and the set of experiments with results outside the acceptable range are highly correlated. Once we identify this indicator, we can use it to find experiment results that are likely unreliable when running future experiments. Although an ideal indicator would be useful across different experiment scenarios, the best telemetry metric and threshold may be highly scenario-dependent.

A good indicator will filter out many unreliable experiments, so that the Tukey and RoA metrics will show better agreement to the baseline compared to the set of results when there was no filtering applied. However, it is also important to look for an indicator that doesn't over-filter the experiments. Thus, the success of different indicators can be compared using the Tukey and RoA metrics described above as well as with a confusion matrix applied to the indicator's results.

The confusion matrix compares the "true" validity of the replicates to the results of filtering with a particular indicator. Validity is determined using the RoA test described above. If the quantity of interest for a given replicate falls within the range of acceptable values given by the RoA test, then the true label for that replicate is "Valid". Otherwise, the label is "Invalid". The test label is assigned to the replicate based on the telemetry and threshold for the indicator in question. The confusion matrices are then normalized over the true conditions. A high ratio of false negatives indicates that the telemetry metric is over-filtering reliable replicates. A high ratio of false positives indicates that the telemetry metric is not successfully filtering out replicates that are invalid.

Note that this process of filtering according to some indicator may need to be approached differently depending on the nature of the scenario's quantity of interest. For example, the Command and Control scenario has four discrete quantities of interest, as opposed to a single continuous quantity from the Scanning and Detection scenario. The method used to evaluate agreement to the baseline based on all or some of these quantities of interest may affect the perceived success of failure of an indicator.

6 RESULTS

6.1 Scanning and Detection Results

Recall that the method used for oversubscribing resources in this study is to run more and more copies of the experiment scenario in parallel. Experiments for the Scanning and Detection scenario were run under several different emulation configurations, each of which differed in the number of parallel namespaces used to run replicates of the scenario experiment. A summary of these settings can be seen in Table 1. Note that several replicates were removed from the study due to corrupted or insufficient data. Across all settings, there were 3038 total runs with sufficient data for further analysis.

In the scanning and detection scenario, three candidate telemetry metrics were considered independently:

Table 1: Summary of Scanning and Detection Emulation Settings

Parallel Namespaces	1	2	4	5	10	20	33	50
Replicates per Namespace	400	200	100	80	40	20	12	8

- *Stolen Cycles* – The amount of time stolen from the virtual machine waiting for the host CPU to become available. The threshold for this metric is no cycles stolen during the experiment.
- *System Load* – This is the CPU demand on the physical host in terms of the number of processes running. The threshold for this metric is that the system load will not exceed the number of logical host cores for the duration of the experiment (in this case, 64 cores). Note that system load is calculated as a minute average, and these experiments last on the order of seconds.
- *Throughput* – This is the number of bytes per second of TCP traffic averaged over the course of the experiment. 250 KB/s, the average of all 1-namespace experimental run, was chosen as the minimum threshold for this study.

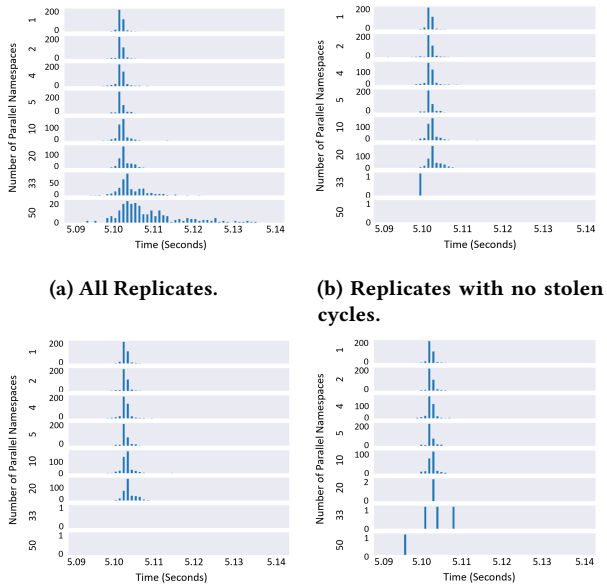
These metrics are captured every second for the virtual machine statistics, and every 10 seconds for the host statistics.

Recall that we consider the 1-namespace experiments the baseline set. In Figure 3(a), we can see the distribution of data deviates significantly from the baseline for numbers of parallel namespaces greater than 20. From these analyses, we would say that the results from 20-, 33-, and 50- parallel namespaces come from different distributions than the baseline data, and that at least some of the replicates under these conditions are unreliable. The other histograms in Figure 3 show the data distributions when certain replicates are filtered out according to the specified indicator. All three metrics appear to do a reasonable job of removing replicates with alert times dissimilar to the baseline distribution.

However, looking at the Tukey comparisons and confusion matrices in Figure 4, the indicators are giving ideal results. System load and throughput do the best job of leaving replicates with alert times that match with the 1-namespace distribution. Stolen cycles is actually not as successful as a telemetry metric, according to the Tukey results. However, the Load-based indicator does have a higher false positive rate and the Throughput-based indicator has a higher false negative rate. In fact, in all cases, each of these metrics seem to filter out many replicates that have acceptable alert times – especially those at higher namespaces. This can be seen in Figure 5.

Overall, these results could indicate that the thresholds for the telemetry could be better tuned, although this may make the indicators highly specific to this particular scenario and emulation platform configuration. For all indicators, although the Tukey and confusion matrix results are not perfect, there is a clear improvement in agreement to the baseline distributions when replicates are filtered compared to the unfiltered replicate set. Performance on this specific scenario should be weighed against the desire for generalizable indicators.

The results of the Scanning and Detection study show considerable success in applying the proposed verification procedure to an emulated scenario. We are clearly able to identify the point at



(a) All Replicates.

(b) Replicates with no stolen cycles.

(c) Replicates with a system load under 64.

(d) Replicates with TCP throughput above 250kB/s.

Figure 3: Histograms of the distribution of alert times at each namespace.

which physical resources are over-subscribed. Based on this, we can identify several promising telemetry-based indicators that could be used to help filter out experiments with unreliable results.

Now that we have seen success on one scenario, we aim to apply the same process to the Command and Control scenario and see how the results differ.

6.2 Command and Control Results

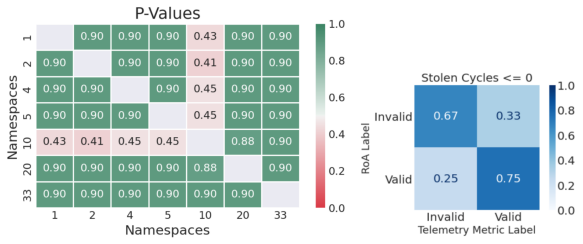
As with the Scanning and Detection scenario, Command and Control experiments were run under different emulation configurations, each with increasingly more parallel namespaces. A summary of the settings tested for the Command and Control scenario can be seen in Table 2. The total number of replicates for the command and control verification study is 1187.

Table 2: Summary of Command and Control Emulation Settings

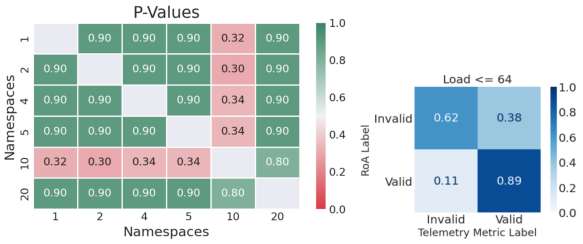
Parallel Namespaces	1	2	5	10	20	40
Replicates per Namespace	200	100	40	20	10	5

Recall that the quantities of interest for this scenario are the number of alerts by time t=1, 5, 10, and 16 seconds. Due to the multiple quantities of interest, Tukey statistics must be aggregated across all quantities of interest in order to compare result distributions. The aggregate function used in this study was the mean.

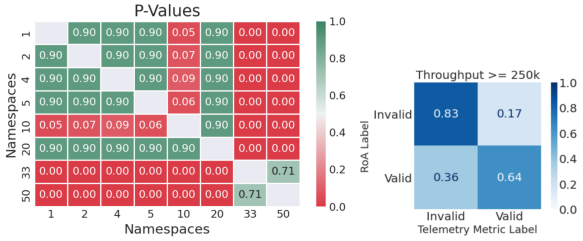
Figure 6 shows the Tukey analysis performed on all replicates for this scenario. We can see that around 20 parallel namespaces, the quantity of interest results deviate significantly from the baseline.



(a) Replicates with zero Stolen Cycles.

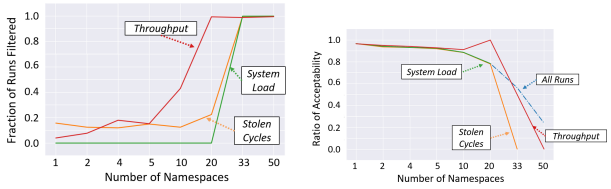


(b) Replicates with Load under 64.



(c) Replicates with Throughput above 250kB/s.

Figure 4: Tukey and Confusion Matrix Results for Selected Indicators on Scanning and Detection Scenario.



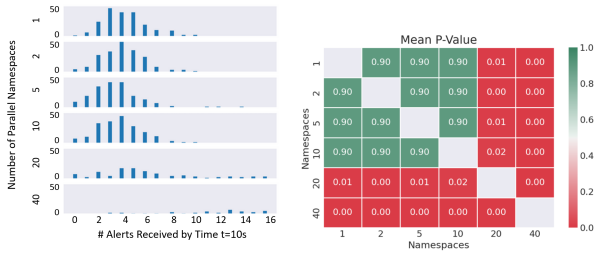
(a) Ratio of runs filtered out at each namespace.

(b) RoA at each namespace.

Figure 5: Comparing the number of filtered replicates to the improvement in the RoA metric.

This is also reflected in the histograms for the scenario. From these analyses, we would say that some of the replicates from 20- and 40-parallel namespaces are likely unreliable.

We wished to first test the exact same indicators as the Scanning and Detection scenario. If the same metrics worked approximately as well in both scenarios, it could indicate a highly generalizable metric. In addition, we tested a variety of other available telemetry and thresholds. During experimentation, telemetry is captured every 5 seconds for the VM statistics and every 10 seconds for the host statistics. The set included:



(a) Number of Alerts at 10 seconds, distribution at each namespace.

(b) Tukey multiple comparison.

Figure 6: Overview of all command and control emulation runs.

- *Stolen Cycles* – See Scanning and Detection results for telemetry description. The best-performing threshold tested was no more than one stolen cycle.
- *System Load* – See Scanning and Detection results for telemetry description. Although we tested the same threshold as the Scanning/Detection method used (64), we found that the Command and Control scenario needed a much lower threshold, so we chose no more than 14 processes.
- *Interrupts per Second* – This is the number of times in one second that the system is interrupted. The more interrupts there are, the more clock time a single process will take to complete. From looking at the resulting data, a threshold of no more than 2250 was set.

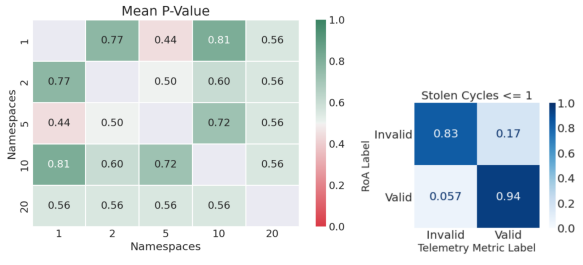
Note that although we wanted to test the Throughput metric with this scenario as well, throughput data was not successfully collected from enough experiments to utilize this metric.

The Tukey comparisons and associated confusion matrices for these indicators can be seen in Figure 7.

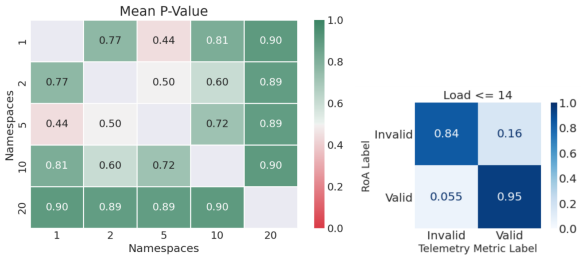
The approach to verification analysis taken with the Scanning and Detection scenario needed to be adapted for the Command and Control scenario. Even if the same telemetry has the potential to make a good indicator, the filtering thresholds often needed to be adjusted for the new scenario, as is discussed in subsection 6.2.1. Although there isn't an obvious candidate for a good universal indicator between the Scanning and Detection and the Command and Control Scenarios, we were still able to find good indicators for the Command and Control scenario specifically.

6.2.1 Setting Thresholds. In order to strike a balance between the results of the Tukey comparisons and the confusion matrices, we can tune the threshold settings for each telemetry. Figure 8 compares the Stolen Cycles indicator with a threshold of 0 (used in the Scanning and Detection Scenario) vs a threshold of 1. We see that the more forgiving threshold of 1 allows us to keep more replicates while still improving the Tukey results overall. As discussed earlier, it is important to choose an indicator that can both improve Tukey results and avoid over-filtering in order to maintain as many reliable replicates as possible.

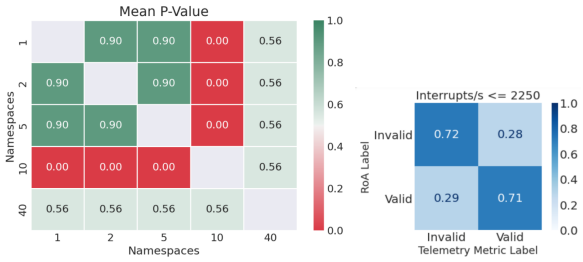
Figure 9 compares the results of the host load-based indicator when using different thresholds. For the Scanning and Detection



(a) Replicates with Stolen Cycles less than or equal to 1.



(b) Replicates with Load less than or equal to 14.

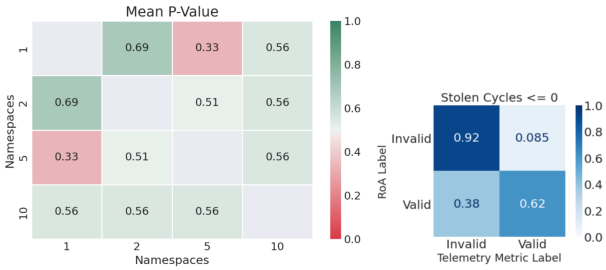


(c) Replicates with Interrupts/second less than or equal to 2250.

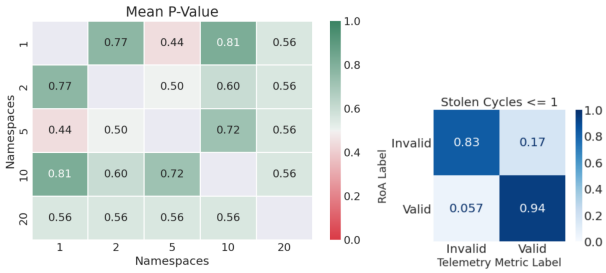
Figure 7: Tukey and Confusion Matrix Results for Selected Indicators on Command and Control Scenario

scenario, the threshold was chosen to be 64 to match the number of cores. For reasons discussed in the next subsection (6.3), the number of cores available on the host was reduced to 32 for running the command and control scenario. Therefore, we tested the load-based indicator with thresholds of 64 and 32, but found that an even lower threshold of 14 was more effective at filtering out the unreliable replicates.

Additional insight about thresholds can be gained by looking at the distribution of telemetry values collected from all replicates in each emulation setting. This is one tactic for approximating reasonable thresholds to set for our metrics, as long as we accept that the threshold will be very scenario-dependent. However, such data can also help to find telemetry that will likely not be useful indicators. Figure 10 shows two box plots, one for load and one for number of interrupts per second. For load, we see a smooth increase in load as we increase the number of parallel namespaces, with a very sharp increase as we get to 20- or 40- parallel namespaces. It is easy to conceive of a horizontal line on this plot that might linearly separate datapoints of "good" replicates from datapoints of "bad" replicates. This separability indicates that load could be a good indicator, and the plot gives us an idea of how to set the



(a) Replicates with zero Stolen Cycles. (b) Filter success.



(c) Replicates with Stolen Cycles less than or equal to 1. (d) Filter success.

Figure 8: Results using two different thresholds on "stolen cycles" telemetry.

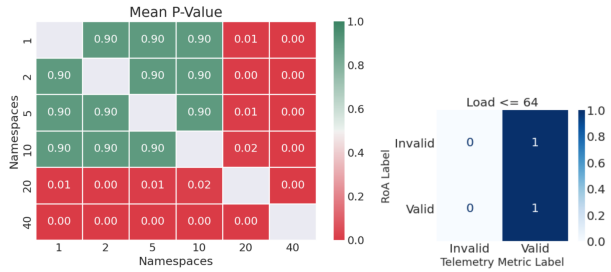
threshold. On the other hand, notice the wide range of collected values for the interrupts per second telemetry. The box plots are mostly centered around the same values, but there are also many low- and high-end outliers. This pattern persists regardless of the number of parallel namespaces. These factors could indicate that developing an indicator based on interrupts telemetry would be difficult or unsuccessful.

Threshold identification is challenging. Ideally, we want to identify thresholds which do not rely on experiments, especially because one may not be able to generate controlled conditions in larger, more complicated emulation scenarios. Threshold setting for verification testing deserves more attention in the emulation community.

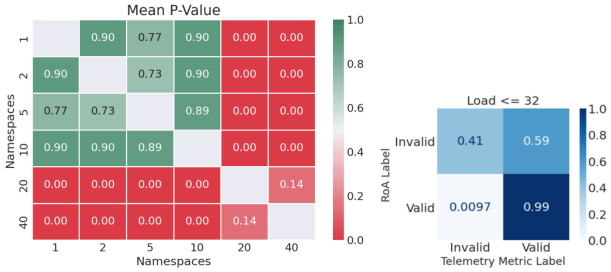
6.3 Process Generalizability

The verification process we established showed considerable promise on the Scanning and Detection scenario. However, the process was not immediately applicable to the Command and Control scenario. Additional factors had to be considered, such as the difference in resources required to run each scenario, the type and amount of data to be collected from each scenario, and how simplifying decisions in the emulation model affect our verification methods.

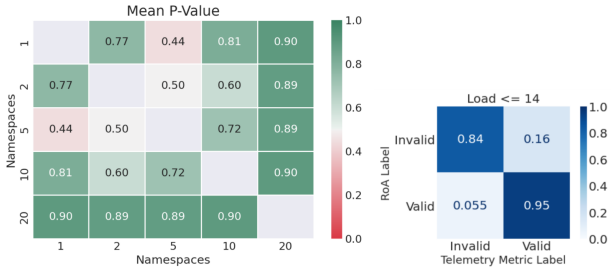
In the results for the Scanning and Detection scenario (Figure 1), there is a clear point where the physical host was oversubscribed, around 33 parallel namespaces. We did not initially see this same point in the histogram data for the Command and Control scenario - all distributions looked approximately the same regardless of the number of parallel namespaces run. In addition, looking at the Tukey analysis, no p-values were lower than 0.05. Therefore,



(a) Replicates with 64 or fewer parallel processes. (b) Filter success.

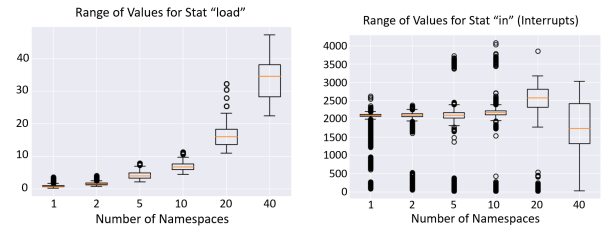


(c) Replicates with 32 or fewer parallel processes. (d) Filter success.



(e) Replicates with 14 or fewer parallel processes. (f) Filter success.

Figure 9: Tukey and confusion matrices when using three different thresholds on "load" telemetry.



(a) Load Telemetry. (b) Interrupts per Second Telemetry.

Figure 10: Boxplot of Data Collected Across All Replicates.

we could not reject the null hypothesis that all experiments were reliable.

There are a couple factors that we theorized would cause the Command and Control results to be inconclusive. Although the Command and Control scenario seemed more complex logically, the scenario implementation for emulation was simplified. The Scanning and Detection scenario involved emulating 27 VMs while the Command and Control scenario only contained four. The Scanning and Detection scenario had one host connecting to many, which requires TCP connection maintenance throughout the experiment. The Command and Control scenario simplified all network traffic emulation by having only one VM actually output the traffic. For both of these reasons, the Command and Control scenario as it was defined in this study required far fewer resources to run than the Scanning and Detection scenario.

We considered several approaches to address this. One approach was to run more parallel namespaces. However, we found during testing that our emulation infrastructure did not allow us to run 50 or more parallel namespaces (contact authors for a discussion of these issues, which were traced to Python socket handling).

Another approach to increasing physical resource demand is to reduce the amount of resources available to the host. We did so by reducing the available processing power on the physical host by half. The results described in 6.2 came from a set of experiments where resources were limited in this way. From the results shown, it is clear that this configuration did lead to resources being oversubscribed in a similar way to the Scanning and Detection scenario.

Several additional methods are left to explore in Future Work.

With the conclusion of the two scenario studies, we have shown that our proposed verification process is generalizable. A great deal of insight was gained through the process of applying our verification methods to two different scenarios.

- More consideration was required to process Command and Control data in a timely and meaningful manner due to the volume of data. Modifications to data collection methods were required to preemptively reduce the volume of output traffic data, and even then 1200 experiment replicates output approximately 20Gb of data total.
- Verification can highlight subtleties in physical host configurations. Using our local computing cluster, we found a marked difference in results depending on which physical host ran the experiments, even though each host should have similar specifications.
- The design of the emulated system model can effect verification methods. When addressing the level of fidelity with which to model a system, consider the abilities of the physical host running the emulation. Similarly, when telemetry is collected, consider the emulation model when interpreting results. It is possible that the design of the model or the emulation platform could affect the outcome of the verification experiments.

Overall, the verification process we have outlined presents a significant step forward in emulation verification. We were able to use our process to verify the capabilities of our physical host to run large numbers of parallel emulations successfully without introducing emulation artifacts to the resulting data. In addition, our process shows great promise of being a generalizable framework for emulation verification.

7 FUTURE WORK

We have only scratched the surface of what can be explored through verification. First, the Results section (6) discussed two methods we pursued to force the Command and Control scenario to stress the available physical resources of the system. One method we did not test was making the scenario itself more complicated. We could try any of the following:

- Add more VMs to the scenario without changing the functionality of any existing VMs or implementing functionality on new VM's. This could stress memory resources.
- Have the VMs perform more resource-intensive processes.
- Make the network for the scenario more complex. The physical host must maintain network connections in the emulation, so increasing traffic complexity could stress the host.

Performing any of these experiments could further inform new research questions, such as which tactics are more effective at causing stress to the system. Answering this might help to direct the selection of more informative telemetry for defining verification metrics.

Additionally, there is more than one way to think about verification. So far, we have only explored the topic of resource verification. There are other levels of realism to verify, including realistic network traffic patterns and software execution[14]. The type of verification may depend on the questions being asked about the scenario, and different questions may require different metrics.

There is also a great deal more to explore in the way of metric formulation. One path that we considered in a limited way was the application of a machine learning model to develop indicators from multiple telemetry metrics. This approach could give more of a comprehensive evaluation of experiment health and reliability as compared to a single-telemetry metric. Using a machine learning model to learn weights for various telemetry could help develop a combined metric which would be able to better determine unreliability of replicates. Additionally, high feature importance in the model could be used to drive further single-telemetry indicator studies.

Finally, there are a couple fundamental assumptions on which our verification analysis relied, and these assumptions should be further explored.

- Our metrics relied on a baseline set of data to determine the base distribution for acceptable experiment results. This was calculated from results given when only one experiment was run at a time, since this was the minimum resource utilization for our set of experiments. But how do we know that this really is an acceptable baseline? If the scenario was sufficiently large and complex, or the physical resources were sufficiently limited, only a single experiment may overwhelm the physical host, giving unreliable results. The assumption that the physical host has enough resources for one experiment is akin to a mathematical axiom. We need this assumption to help "prove" other hypotheses, but proving the axiom itself could be very difficult.
- We define "sameness" of distributions by a successful Tukey test. But is this enough? How close do two distributions have to be to each other to be considered the same? And is

this sameness of distributions sufficient to indicate reliable experiment results?

We made early, sensible assumptions at the time that allowed us to build this study. But it may be fruitful to look back and reconsider these assumptions.

8 CONCLUSIONS

Verification is a key process for ensuring that system emulation successfully models the real world. We demonstrated a proposed verification process on two different cyber attack scenarios. In the Scanning and Detection scenario, we found that we could use telemetry-based metrics to detect experiments that were likely to have unreliable results. By applying a similar method to the Command and Control scenario, we were able to show that the same verification process, while not immediately applicable, was fairly generalizable to the new scenario. The exercise also gave us key insights into the subtle differences to be aware of in physical resource configuration, the importance of fidelity and implementation decisions in the emulation platform, and differences in data collection and storage demands from scenario to scenario.

In general, experiment repeatability is key to bringing rigor and scientific process to cybersecurity. Just as reported experiments and results in the physical sciences should be repeatable and verifiable, so too should cyber experiments. Failure to reproduce a result may be the fault of emulation artifacts rather than faulty experiment design. These artifacts can come from the abilities and resources of the physical host running the emulation or from the emulation platform itself. The process of verification and use of telemetry metrics can help evaluate these factors. Repeatability and reproducibility of experiments can strengthen belief in the validity of the results. A result that holds under repetition and various modeling environments is more likely to hold in the real system as well. In this way, verification can help to give us more confidence that experimental environments can help to inform our decisions on real life cyber systems.

ACKNOWLEDGMENTS

This work has been supported by the SECURE project within the Grand Challenge LDRD Program at the Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] 2012. IEEE Standard for System and Software Verification and Validation. *IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)* (2012), 1–223. https://doi.org/10.1109/IEEESTD.2012.6204026

[2] 2018. *Alert (TA18-201A) Emotet Malware.* Techncial Report. US CERT. https://us-cert.cisa.gov/ncas/alerts/TA18-201A

[3] 2021. Azure Monitor. (2021). https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-common-metrics.

[4] 2021. Cloud Telemetry: Advancing Your IT Strategy. (2021). https://www.intel.com/content/www/us/en/cloud-computing/telemetry.html.

[5] 2021. A Discrete-Event Network Simulator for Internet Systems. (2021). https://www.nsnam.org/.

[6] 2021. Network Telemetry. (2021). https://cloud.google.com/network-telemetry.

[7] 2021. What is Telemetry? The Guide to Application Monitoring. (2021). https://www.sumologic.com/insight/what-is-telemetry/.

[8] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, and John Wroclawski. 2009. Current developments in DETER cybersecurity testbed technology. In *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE, 57–70.

[9] Cisco. 2019. Snort intrusion detection and prevention system. https://www.snort.org/.

[10] Jonathan Crussell, Thomas M Kroeger, Aaron Brown, and Cynthia Phillips. 2019. Virtually the Same: Comparing Physical and Virtual Testbeds. In *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE.

[11] Jon Davis and Shane Magrath. 2013. *A survey of cyber ranges and testbeds*. Technical Report. Cyber and Electronic Warfare Division, Defence Science and Technology Organization, Australian Government.

[12] Bernard Ferguson, Anne Tall, and Denise Olsen. 2014. National cyber range overview. In *2014 IEEE Military Communications Conference*. IEEE, 123–128.

[13] Seth Hanson, Jerry Cruzy, and Casey Glatter. 2021. *SCORCH User Guide*. Technical Report SAND2021-11504 O. Sandia National Laboratories.

[14] Brandon David Heller. 2013. Reproducible Network Research with High-Fidelity Emulation. (2013).

[15] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. 2008. Network simulations with the ns-3 simulator. *SIGCOMM demonstration* 14, 14 (2008), 527.

[16] Alefiya Hussain, David DeAngelis, Erik Kline, and Stephen Schwab. 2020. Replicated Testbed Experiments for the Evaluation of a Wide-range of DDoS Defenses. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 46–55.

[17] Quan Jia, Zhaohui Wang, and Angelos Stavrou. 2009. The Heisenberg Measuring Uncertainty in Lightweight Virtualization Testbeds.. In *CSET*.

[18] Stephen Midway, Matthew Robertson, Shane Flinn, and Michael Kaller. 2020. Comparing multiple comparisons: practical guidance for choosing the best multiple comparisons test. *PeerJ* 8 (2020). https://doi.org/10.7717/peerj.10387

[19] minimega developers. 2019. minimega: a distributed VM management tool. http://minimega.org/

[20] Jelena Mirkovic, Genevieve Bartlett, and Jim Blythe. 2018. DEW: Distributed Experiment Workflows. In *11th USENIX Workshop on Cyber Security Experimentation and Test CSET 18)*.

[21] Jelena Mirkovic, Terry V Benzel, Ted Faber, Robert Braden, John T Wroclawski, and Stephen Schwab. 2010. The DETER project: Advancing the science of cyber security experimentation and test. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 1–7.

[22] Kimihiro Noguchi, Riley S Abel, Fernando Marmolejo-Ramos, and Frank Konietschke. 2020. Nonparametric multiple comparisons. *Behavior Research Methods* 52, 2 (2020), 489–502.

[23] William L Oberkampf and Christopher J Roy. 2010. *Verification and validation in scientific computing*. Cambridge University Press.

[24] Lee M Rossey, Robert K Cunningham, David J Fried, Jesse C Rabek, Richard P Lippmann, Joshua W Haines, and Marc A Zissman. 2002. Lariat: Lincoln adaptable real-time information assurance testbed. In *Proceedings, IEEE Aerospace Conference*, Vol. 6. IEEE, 6–6.

[25] Stephen Schwab and Erik Kline. 2019. Cybersecurity Experimentation at Program Scale: Guidelines and Principles for Future Testbeds. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 94–102.

[26] Christos Siaterlis, Andres Perez Garcia, and Béla Genge. 2012. On the use of Emulab testbeds for scientifically rigorous experiments. *IEEE Communications Surveys & Tutorials* 15, 2 (2012), 929–942.

[27] Thomas Tarman, Trevor Rollins, Laura Swiler, Jerry Cruz, Eric Vugrin, Hao Huang, Abhijeet Sahu, Patrick Wlazlo, Ana Goulart, and Kate Davis. 2021. Comparing reproduced cyber experimentation studies across different emulation testbeds. *Proceedings, 14th Workshop on Cyber Security Experimentation and Test (CSET21), ACM* (2021).

[28] Eric Vugrin, Jerry Cruz, Christian Reedy, Thomas Tarman, and Ali Pinar. 2020. Cyber threat modeling and validation: port scanning and detection. In *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*. Association for Computing Machinery.

[29] Eric Vugrin, Seth Hanson, Jerry Cruz, Casey Glatter, Thomas Tarman, and Ali Pinar. 2021. Detection of command and control traffic: model development and experimental validation. *in preparation* (2021).