Science & Engineering of Cyber Security by Uncertainty Quantification and Rigorous Experimentation (SECURE) HANDBOOK

Authors: Ali Pinar, Thomas Tarman, Laura Swiler, Jared Gearhart, Derek Hart, Eric Vugrin, Geraldo Cruz, Bryan Arguello, Gianluca Geraci, Bert Debusschere, Seth Hanson, Alexander Outkin, Jamie Thorpe, William Hart, Meghan Sahakian, Kasimir Gabert, Casey Glatter, Emma Johnson, and She'ifa Punla-Green

This is report SAND2021-11459 R.

This work has been supported by the LDRD Program at 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.

This PDF contains a set of sections that are proposed for an on-line Handbook for the SECURE project.

Overview

The following text will be on the main page for the handbook:

This website documents the methods, tools, and case studies that were created under the SECURE project funded by Sandia National Laboratories. Securing cyber systems is paramount, but cyber defenders lack evidence-based techniques for making decision about high-consequence cyber systems. The 2016 Federal Cybersecurity R&D Strategic Plan [1] states: "Most [cybersecurity] techniques are domain- and context-specific, often not validated as mathematically and empirically sound, and rarely consider efficacy and efficiency. Thus, the state of the practice consists of heuristic techniques, informal principles and models of presumed adversary behavior, and process-oriented metrics." This plan emphasizes a need for evidence-based approaches to cybersecurity, which employ principled and rigorous measurements and models.

To help address this need, SECURE developed techniques for evidence-based cybersecurity that build upon the cyber experimental foundation provided by emulation-based testbeds, which provide scalable, virtualized environments for modeling cyber systems. Specifically, this research integrated emulation models, mathematical models, optimization, and uncertainty quantification

into workflows that enable evidence-based risk assessment and risk mitigation. This supports the science of cyber security by providing a foundation to produce quantitative knowledge concerning a target system, understand the limitations of available data, estimate cybersecurity risks, and identify defensive strategies.

The SECURE Handbook consists of the following sections.

- A collection of the key *research terms and concepts* from this work.
- A recommended *workflow for cyber experimentation*.
- A summary of the *tools* that were used and/or developed under SECURE
- A set of *case studies* demonstrating the SECURE workflow and the application of rigorous cyber experimentation approaches such as verification and validation, optimization, and uncertainty quantification.
    - An *Enterprise Command and Control (C2)* study that that analyzes a C2 malware attack on an IT system.
    - A *SCADA network* study where an attacker aims to discover vulnerable remote terminal units (RTUs) on a utility grid's cyber system.
    - A *power grid impact study* that connects attacks on the associated cyber to system to consequences on the grid and identifies key vulnerabilities.
    - A *threat study*, where a Markov chain-based model it used to analyze a multi-step attack that includes the C2 and SCADA attacks listed above.

[1] National Science and Technology Council, "Federal Cybersecurity Research and Development Strategic Plan." 2016. https://www.nitrd.gov/pubs/2016-federal-cybersecurity-research-and-development-strategic-plan.pdf

SECURE Research Terms and Concepts

The purpose of this section is to provide the reader with an overview of the research terms and concepts explored in the SECURE project. This section is not necessarily meant to be read linearly: the reader can go directly to the topic of interest.

Terms

There are many types of models used in cyber analysis. The following summarizes our use throughout these documents.

- *Cyber Model* – a generic term that can apply to any methods (or combinations of methods) used to assess cyber systems
- *Cyber Testbed* - the hardware platform and software framework used to run a cyber model or combination of cyber models.
- *Physical Model* - Cyber models that run real software on a representative hardware platform to model the actual system in full fidelity.
- *Emulation Model* – Cyber models that run real software in real time on a computing cluster, using hardware abstractions such as virtual machines (VMs) and/or containers to represent individual nodes, and virtual networking technologies such as Virtual Local Area Networks (VLANS) to interconnect VMs or containers.
- *Emulation Testbed* – (also known as "virtual testbed") The resources used to instantiate emulation models. E.g. computing cluster, virtualization technologies, and experimentation/orchestration software.
- *Simulation Model* – primarily discrete event simulators (e.g. OMNET++ [1] or ns-3 [2]), which run abstract representations of software and hardware. These models can run faster than real time.
- *Mathematical Model* – Mathematical formulas that capture dynamic and/or steady-state values of a quantity of interest and can be solved using mathematical analysis tools such as Matlab or Mathematica.
- *Optimization Model* – A math-based model that can be used to efficiently identify worst-case attacks and/or optimal defense strategies using simplified representations of a cyber or cyber/physical system.

[1] https://omnetpp.org

[2] https://www.nsnam.org

Concepts

Several modeling and analysis concepts were used and expanded upon to develop our cyber experimentation methodology. The following summarizes these areas.

Threat Modeling

Threat modeling begins by first identifying the various steps an attacker will make in a particular attack (e.g. reconnaissance, privilege escalation, credential access, script execution, command and control, etc.). The MITRE ATT&CK framework (https://attack.mitre.org/) outlines many techniques that can be used to achieve success at each attack step.

The second phase of threat modeling is to quantitatively analyze the attack success at each step and then identify optimal mitigations. Under the SECURE project, we developed a threat modeling approach that utilizes a game theory approach called GPLADD [1] to first represent the attack success conditions as attack graphs and then to quantify the attack success metrics as a function of the attacker and defender strategies. We then translate the GPLADD to a Markov model representation of the attack and populate the transition matrix in the Markov model with attack steps from the MITRE ATT&CK model [2]. Because the framework allows for multi-step attacks with different times per step, we can examine multiple results, including the attack state over time, the Markov-chain steady-state distribution of being in various attack states, the time-to-success distribution, and the impact of various defenses on the graph and resulting probabilities.[2]

[1] A. V. Outkin, B. K. Eames, M. A. Galiardi, S. Walsh, E. D. Vugrin, B. Heersink, J. Hobbs, and G. D. Wyss, "GPLADD: Quantifying trust in government and commercial systems, a game-theoretic approach," ACM Trans. Priv. Secur., vol. 22, no. 3, pp. 18:1–18:27, Jun. 2019. [Online]. Available: http://doi.acm.org/10.1145/3326283

[2] Defender Policy Evaluation and Resource Allocation against MITRE ATT&CK Data and Evaluations. Alexander V. Outkin, Timothy Schulz, Thomas D. Tarman, Patricia V. Schulz, Ali Pinar. SAND2021-7713. https://arxiv.org/abs/2107.04075,

Mathematical Modeling of Attack Steps

Under SECURE, we developed two mathematical models of attack steps for the purposes of having fast, approximate models for multifidelity modeling and for examination of the assumptions about attack progress. Each mathematical model is governed by a set of equations that formally represent the state of the system and its evolution over time. For example, the state might be represented by the number of ports (closed, open, and filtered) that have not yet been identified by a scanning tool. These mathematical models can also include probabilistic representations, such as Poisson arrival rates for malicious and benign traffic. Both mathematical models were validated by comparison with cyber emulation models.

The two mathematical models developed for SECURE are:

1. *Port scanning with Nmap and detection with Snort*. This model describes the rate of port-scanning progress by an attacker and intrusion detection by the network defender. We validated this model with a set of emulation experiments conducted with a virtual testbed. The model is documented in the following paper:
    - Eric Vugrin, Jerry Cruz, Christian Reedy, Thomas Tarman, and Ali Pinar "Cyber Threat Modeling and Validation: Port Scanning and Detection," in *Proceedings of the 7th Annual Hot Topics in the Science of Security (HoTSoS) Symposium* (2020). Sept. 2020. https://doi.org/10.1145/3384217.3385626

2. *Detection of command and control traffic.* This model determines the probability that a number of alerts will be generated by the intrusion detection system (IDS) at each time step (given the arrival rates of malicious and benign traffic, the probability that packets will be dropped by the IDS, etc.). Ultimately, this model can be used to detect the probability the attacker will be seen and the rate of false positives.

Verification

Verification refers to ensuring the correctness of the model implementation: is the model implemented and working as intended? [1] Part of verification involves software testing and quality assurance. A unique aspect of cyber emulation involves assessing the performance of the emulation running in the virtualized environment: are there sufficient virtualized resources to properly handle the scenario that is being run, or are the virtualized components producing experimental artifacts and behavior that may result in the experimental outcomes to be unrepresentative or incorrect?

In our work, we have focused on the use of telemetry metrics (metrics collected from the virtual machines or host running the emulations) [2-5] to identify anomalous behavior. The telemetry metrics examined include stolen cycles, load, throughput, context switches, and user time. We have developed a series of experiments where we repeat the same emulation on one host using an increasing number of namespaces, where a namespace refers to an experiment that is isolated on its own VLANs. The deployment of the experiment to increasing number of namespaces tends to oversubscribe resources, which is seen in the telemetry metrics and output quantities of interest from the experiment.

1. Oberkampf, W.L. and C.J. Roy. Verification and Validation in Scientific Computing. Cambridge University Press, 2010.
2. https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-common-metrics
3. https://cloud.google.com/network-telemetry
4. https://www.sumologic.com/insight/what-is-telemetry/
5. https://www.intel.com/content/www/us/en/cloud-computing/telemetry.html

Validation

Validation refers to the adequacy of a model for an intended application of the model. Typically, one compares model predictions with observational or experimental data (the "benchmark" data) in validation to determine this adequacy: is the model "close enough" as measured by some metric? In the context of validation in SECURE, we have focused on reproducibility in cyber experimentation. Reproducible results are foundational in science because they provide evidence that discoveries are completely documented and provide assurance that reported results are not biased.

We performed a reproducibility study by first running a cyber experiment of scanning and detection scenario on Sandia's `minimega` emulation platform. Then, we compared the replications with the same experimental configuration conducted by Texas A&M University using their CORE testbed environment, another cyber emulation framework. The Sandia `minimega` results were considered the benchmark data we used for comparison: we validated the Texas A&M CORE emulation results against the Sandia results.

The process of reproducing the original experiment inspired us to question how we should compare the experimental results, which metrics might best evaluate the similarity of the results, and, ultimately, when it is possible to define a cyber model as "reproducible."

The metrics used, the two sets of experimental results, and the comparisons of those results are described in [1]:

> [1] T. D. Tarman, T. Rollins, L.P. Swiler, J. Cruz, E. Vugrin, H. Huang, A. Sahu, P. Wlazlo, A. Goulart, and K. Davis. Comparing reproduced cyber experimentation studies across different emulation testbeds. *USENIX 14th Cyber Security Experimentation and Test (CSET) Workshop*. Aug. 9, 2021. SAND2021-5696C.

Uncertainty Quantification (UQ)

Uncertainty quantification refers to characterizing input uncertainties and propagating them through a model (e.g. a cyber simulation or emulation model) to obtain the resulting uncertainties on the output quantities of interest. Uncertainty analysis can be used to assess the likelihood of typical or extreme outputs, determine the mean or median performance, understand the variability in the responses, and find probability of failure. A related activity to UQ is sensitivity analysis, which is the identification of the most important variables affecting the response. It involves understanding how model outputs vary as the inputs vary.

In SECURE, we studied three areas supporting UQ. *Dimension reduction* identifies the most important components of a high-dimensional space, allowing uncertainty analysis to focus only on the important components, thus helping tractability. *Discrete polynomials* are an example of a surrogate model, which serve as a "surrogate" or proxy for the computationally expensive simulation or emulation. Surrogate models are used extensively in UQ and optimization of computational models because they are fast to evaluate. However, the accuracy of the surrogate

approximation must be determined. *Multifidelity UQ* is another area of UQ which attempts to improve efficiency of sampling by incorporating samples from both low and high fidelity models.

Dimension Reduction

In monitoring the behavior of physical or emulated computer experiments, the number of certain events that occur in a given timeframe can be highly significant. Thus, recording these quantities at some frequency (e.g. every second) creates useful time-series data, although that data may be inherently stochastic (due to randomness in timings of initializations, small changes in orderings of system calls, etc.). The challenge is to understand how much of the inherent randomness observed in time series vectors of quantities from cyber experiments can be explained by a few underlying components (i.e. reducing the dimensionality of the data while retaining as much of its variability as possible).

In this work, we examined Principal Component Analysis (PCA) on cyber experiment time-series and compared with a discrete version of PCA called XPCA. We studied XPCA because the Nmap port discovery results are discrete values: 1, 2, 3, etc. ports found. We applied PCA and XPCA to several datasets involving 1000 replicates of port scanning results. Our main finding of this work is that PCA performs better than XCPA with respect to variance explained but worse with respect to reconstruction error on these discrete time series data sets. This is due to the discrete nature of the port discovery time series. The full results are described in [1].

[1] "Time Series Dimension Reduction for Surrogate Models of Port Scanning Cyber Emulations." Erin C.S. Acquesta, Laura P. Swiler, and Ali Pinar. SAND20-10617.

Discrete Polynomials

Uncertainty quantification is often accomplished via computationally expensive Monte Carlo sampling. However, less costly stochastic expansion methods can approximate the functional dependence of the simulation response on uncertain model parameters by expansion in a polynomial basis. The polynomials used are tailored to the characterization of the uncertain variables. Polynomial chaos expansion (PCE) is based on orthogonal polynomials. The goal of PCE is to construct a more efficient and accurate estimate of the uncertain response distribution than would be obtained from Monte Carlo sampling.

In this research, we investigated the use of discrete orthogonal polynomials for constructing polynomial chaos expansions to build a response approximation of the results from cyber experiments. One unique feature of the work is the presence of replicates (replicated data points) from the cyber emulations. Reference [1] discusses how samples are chosen in input space and presents an analysis of "best practice" approaches for constructing stochastic expansions based on data one might obtain from a cyber experiment.

[1] Bert J. Debusschere, Gianluca Geraci, John D. Jakeman, Cosmin Safta, and Laura Swiler, "Polynomial Chaos Expansions for Discrete Random Variables in Cyber Security Emulytics Experiments", SIAM CSE 2021 presentation, March 1, 2021. SAND2021-2270C.

Multifidelity UQ

Often, uncertainty quantification (UQ) is challenging to perform because of the large number of samples that must be run through a cyber model, which can be computationally expensive. However, lower-cost multifidelity UQ methods run many samples from one or more low-fidelity models (such as a mathematical model or a network simulator like NS-3) that are fused with a few runs of a high-fidelity cyber model (e.g. actual software run on real or virtualized hardware) to decrease the estimator variance and obtain more reliable statistics. Reference [1] presents the theory behind multifidelity UQ. While the theory for multifidelity UQ existed before SECURE, we are the first group to demonstrate it on cyber emulation uncertainty problems, to our knowledge. Reference [1] presents several network problems of increasing difficulty and demonstrates that the multifidelity estimator demonstrated increased efficiency with respect to Monte Carlo sampling.

[1] Geraci, G., Crussell, J., Swiler, L.P. and Debusschere, B. J. "Exploration of Multifidelity UQ Sampling Strategies for Computer Network Applications." International Journal of Uncertainty Quantification, 2021. Pp. 93-118. DOI: 10.1615/Int.J.UncertaintyQuantification.2021033774

Adversarial Optimization

Another focus of the SECURE project was the use of adversarial optimization to model the interactions between cyber defenders and attackers. Standard optimization models aim to identify a solution that maximizes or minimizes a given function, subject to a collection of mathematical constraints. Adversarial optimization extends standard optimization methods by embedding optimization models within other optimization models. These methods are of particular interest to SECURE because they provide a means of finding worst-case attacks against a system.

The adversarial optimization work on SECURE had two main focuses. The first was developing a toolkit to express and solve adversarial optimization problems. While there is a large body of published literature on adversarial optimization algorithms, there are few general-purpose tools available to write and solve these types of problems. In practice, this means that applying these methods typically requires custom solutions. To address this, the SECURE team developed the Python Adversarial Optimization (PAO) toolkit [1-3], which contains both a modeling language for expressing adversarial problems and algorithms for solving them. The second focus was on developing adversarial optimization models to address cyber-physical security problems. The remainder of this section describes each of the adversarial optimization models developed under SECURE.

[1] GitHub repository: https://github.com/or-fusion/pao

[2] Online documentation: https://pao.readthedocs.io/en/latest/

[3] Hart, W. E., A. Castillo, E. S. Johnson, and S. Punla-Green (2021). PAO 1.0: A Python Library for Adversarial Optimization. Tech. rep. SAND 2021–6720. Sandia National Laboratories.

N-k DC-OPF Model

The first optimization capability developed under SECURE was the *N-k* DC-OPF. This model considers worst-case attacks on a DC optimal power flow (DC-OPF) representation of a power grid, which approximates AC power flow. In this example, we assume that the power grid has *N* components and the attacker can disable *k* of those components. The attacker aims to find the set of components to attack so that unmet demand is maximized. Once the grid operator observes the attack, they will update how their system is being operated to minimize load shed. To begin, we implemented an existing model [1]. A key feature of this model is that it does not make any assumptions about how the *k* components on the system are disabled. For example, it could be from either a physical or a cyber-attack. This is useful because this capability can be used to bound the damage that can be caused for a wide variety of threats, without having to model the specific threat. SECURE utilized the *N-k* model in the following two research thrusts:

- First, it was coupled with a cyber-physical emulation to better understand the impact of a CrashOverride malware attack on a notional electric system. More details on this work can be found in [2].
- Second, we explored methods to speed up solution times. While the *N-k* model is a powerful capability for finding worst-case attacks, it can be difficult to solve for large attack budgets, even for networks with a few hundred buses. This difficulty further increases with the number of buses in the network. To address this challenge, we created a simplified version of this model. Analysis and experiments showed that in certain regimes, the results from the simplified model are often as good or nearly as good as the original DC-OPF formulation. The details of this approach can be found in reference [3]:

[1] A. L. Motto, J. M. Arroyo and F. D. Galiana, "A mixed-integer LP procedure for the analysis of electric grid security under disruptive threat," in *IEEE Transactions on Power Systems*, vol. 20, no. 3, pp. 1357-1365, Aug. 2005, doi: 10.1109/TPWRS.2005.851942.

[2] A. Castillo, B. Arguello, G. Cruz and L. Swiler, "Cyber-Physical Emulation and Optimization of Worst-Case Cyber Attacks on the Power Grid," 2019 Resilience Week (RWS), 2019, pp. 14-18, doi: 10.1109/RWS47064.2019.8971996. SAND2019-12468C.

[3] Emma S. Johnson and Santanu S. Dey, "A Scalable Lower Bound for the Worst-Case Relay Attack Problem on the Transmission Grid," arXiv, 2021, 2105.02801. SAND2021-10211O.

Network Segmentation

As an extension of the *N-k* DC-OPF model, the power grid cyber-physical network segmentation model was developed under SECURE to improve grid resiliency to SCADA cyber-attacks. The model assumes a three-tier SCADA system where an attacker must start attacks from balancing authorities, the first tier. Attacks must then pivot to control centers to reach substations. Once a substation has been infiltrated, all grid components at that substation are disabled by the attacker to damage the grid and cause loss of power to customers. A network designer can segment networks within each tier a pre-determined number of times to restrict possible attack vectors, with anticipation of the worst possible attack on the segmented SCADA system. By segmenting key

functions in the cyber layer, the network design can limit the scope of attacks and improve systems resilience. This network segmentation model and solution is documented in [1].

[1] B Arguello and E.S. Johnson and J.L. Gearhart, "A Trilevel Model for Segmentation of the Power Transmission Grid Cyber Network", arXiv.2108.10958: https://arxiv.org/abs/2108.10958. SAND2021-10208O

Sensor Placement

The sensor placement optimization model was developed to identify where sensors should be placed in a cyber network to maximize the probability that attacks are detected, knowing that an attacker will aim to evade detection after the sensor are placed. This model uses attack graphs, derived from the threat modeling work described above, as the "game board" where attackers and defenders interact.

Robust Optimization

SECURE also developed methods to incorporate robustness into multi-level adversarial optimization problems. In their standard form, optimization models use constraints that are parameterized by known values. However, in practice uncertainties can exist in the parameters used by the model. When distributional information on these parameters is available, approaches like stochastic programming can be used to account decision making under uncertainty. When distributional information is not available, robust optimization methods offer an alternative approach for dealing with uncertainty. Robust optimization assumes that parameters are not fixed but are instead constrained to take values within some uncertainty set. When robust models are solved, the solutions that are generated are guaranteed to perform well over all parameter values in the uncertainty set.

Under SECURE, these approaches were applied in the context of sensor placement on networks, such as the attack graphs shown in the previous section. In a cyber setting, the sensor model focuses on placing sensors to maximize the probability of detecting an attack. As sensors are placed, the attacker may alter their path to minimize the probability that they are detected. One potential issue with this model is that the sensors that are placed on the network might not perform as expected or advertised. Given this, the robust version of this model helps ensure that the placement decisions guard against some amount of sensor failure or degradation.

Cyber Experimentation Workflow

Given the variety of tools that can be used to assess cyber systems, experimentalists might be tempted to dive right into a study. However, an analysis rigorous enough for use in high-consequence cyber systems requires a carefully thought-out experimental design. This section describes the experimentation workflow developed and used by the SECURE research team while conducting its studies of power grid cyber effects.

The workflow presented in this document is primarily focused on emulation testbed modeling, although it may be employed for other types of cyber models. Thus, to facilitate the discussion, we define the following terms:

- *Cyber Model* – a generic term that can apply to any methods (or combinations of methods) used to assess cyber systems
- *Cyber Testbed* - the hardware platform and software framework used to run a cyber model or combination of cyber models.
- *Physical Model* - Cyber models that run real software on a representative hardware platform to model the actual system in full fidelity.
- *Emulation Model* – Cyber models that run real software in real time on a computing cluster, using hardware abstractions such as virtual machines and/or containers to represent individual nodes, and virtual networking technologies such as Virtual Local Area Networks (VLANS) to interconnect VMs or containers.
- *Emulation Testbed* – (also known as "virtual testbed") Resources (e.g. computing cluster, virtualization technologies, and experimentation/orchestration software) used to instantiate emulation models.
- *Simulation Model* – primarily discrete event simulators (e.g. OMNET++ [0] or ns-3 [0]), which run abstract representations of software and hardware. These models can run faster than real time.
- *Mathematical Model* – Mathematical formulas that capture dynamic and/or steady-state values of a quantity of interest and can be solved using mathematical analysis tools such as Matlab or Mathematica.

Figure 1 shows a spectrum of testbeds employed in the modeling of cyber systems and associated tradeoffs in terms of realism vs. cost.

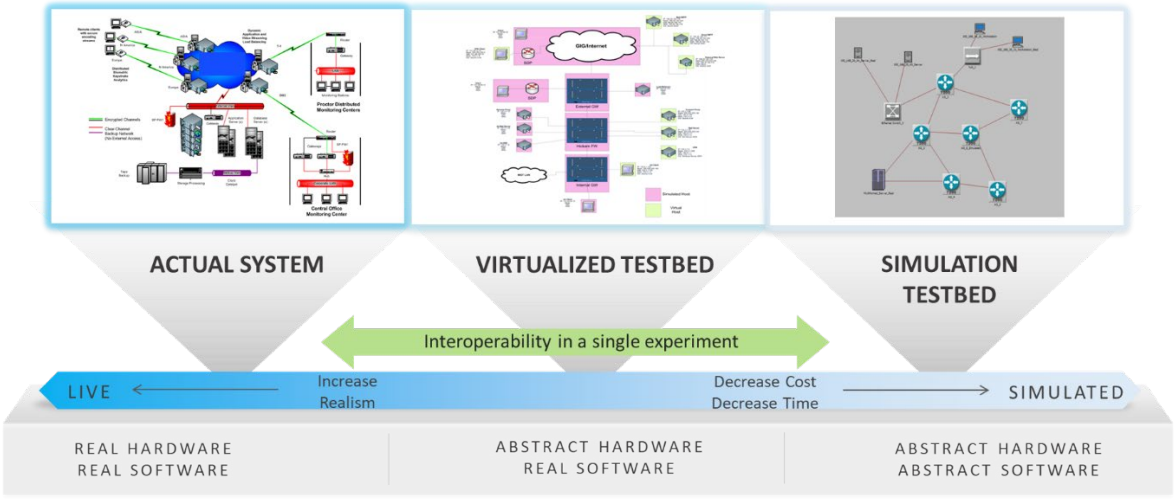


Figure 1. Spectrum of cyber model fidelity, ranging from actual system to simulation testbeds.

Because the topic of experimental design for emulation models is an active area of investigation in the cyber-security research community, several frameworks have been developed to help facilitate sound experimental practices and generate reproducible results. For example, DEW (Distributed Experiment Workflows) [3] provides a generic descriptive language to encode the scenario and topology for an experiment. Likewise, DARPA's National Cyber Range [4], Emulab [5], and DETER [6] are cyber testbeds that can be used for research and experimentation on networks. Reference [7] also examines how platform variations affect emulation models, using carefully structured experiments and statistical analysis. Although these tools exist and work well for experiments, methods for using them rigorously to provide comprehensive evidence to answer questions about high-consequence systems have not been developed and characterized. For example, reproducibility in cyber experiments remains a challenge, due to small timeframes, implementation differences, and differences in platform configurations. Therefore, to facilitate the achievement of reproducible, unbiased results and methods that may be readily applied in other contexts (e.g. on other cyber testbeds with differences in operating systems, software and hardware, kernels, system resources, etc.), the SECURE project developed the following workflow to help guide future studies, as shown in Figure 2. We acknowledge that this workflow was designed for an experimental model (to study sensitivity and uncertainty analysis) but note that it can be applied more generally to generate ensembles of runs that can support optimization studies or other studies. Further detail and a description of SECURE's experimental design (especially the design of experimental runs) can be found in "Design of Experiments for Cyber Emulation" [8].

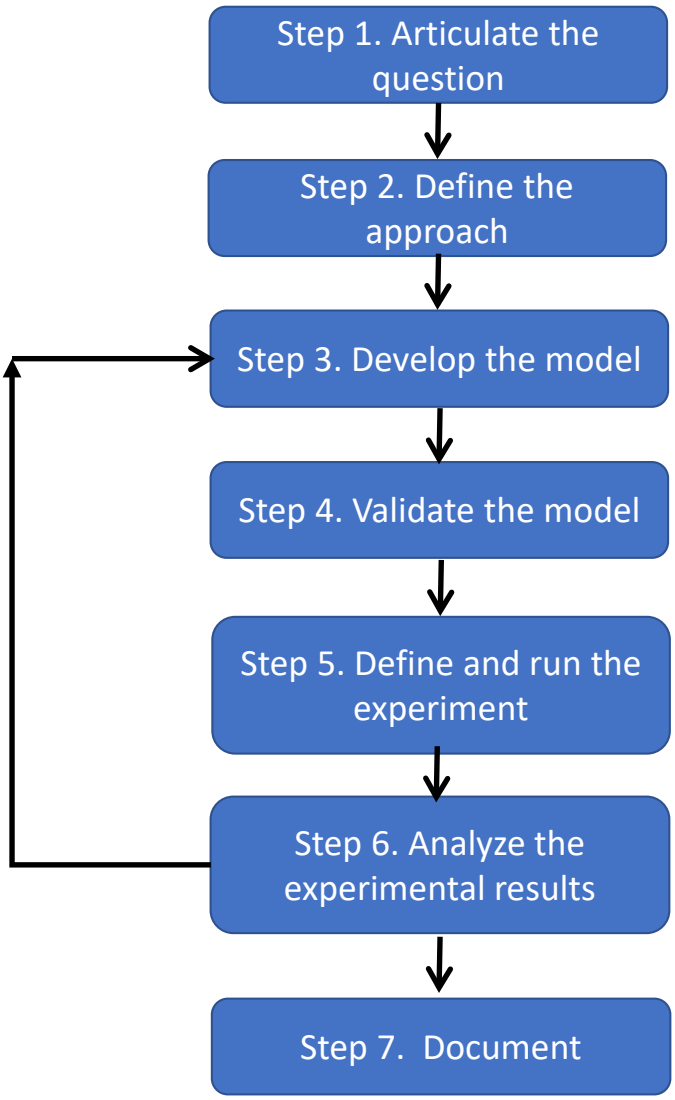


Figure 2. Recommended workflow for cyber modeling suggested by the SECURE project

When performing cyber modeling experiments, we recommend that the following workflow be used:

1. **Clearly articulate the question.** Be specific. (e.g. "If an attacker uses port scans and a given configuration of the Nmap scanning tool, how many alerts will our intrusion detection software identify in a 60-second window?" NOT "Will our intrusion detection software work efficiently?") If possible, identify what statistics are of interest (e.g. the average number of alerts in a time window, the probability that there will be more than 10 alerts, or the full distribution of alerts).

2. **Define the approach that best answers the question.** Scope the problem, identify inputs and outputs, and consider your modeling options.
    a. Identify your requirements (e.g. fidelity, scale, size of parameter space, desired variance in outputs, time per replicate, number of replicates). Most cyber models,

require multiple runs per model configuration setting (i.e. multiple replicates), because there is inherent variability or stochastic behavior in each replicate, due to small timing differences, ordering of various events happening on the system, etc.

b. Choose your modeling domain(s) (e.g. emulation, mathematical), noting that your choice of modeling domain should depend on the model requirements identified in Step 2(a), as shown in Figure 1. For example, if a large scale is required, scalable modeling technologies (e.g. emulation, simulation, or mathematical modeling) would be more practical than physical testbed modeling; however, if high fidelity is required, then physical or emulation testbed modeling would be more effective than simulation or mathematical modeling. Of course, a combination of technologies can be used to maximize outcomes (e.g. a coupled model or models at multiple levels of detail in a multifidelity modeling study).

c. Define how each modeling activity contributes to the answer.

3. **Develop the model, depending on the modeling domains.** The developmental activities for different types of models will vary by model:
    - *Mathematical models* develop equations that will be solved, typically as a function of time (e.g. traffic might be modeled with a Poisson arrival rate distribution to calculate the expected number of packets arriving in a particular time step).
    - *Simulation models* use discrete event network simulators, which often have simulation examples and model libraries (e.g. with different routing protocols, network traffic, etc.) that can be used as building blocks; however, the configuration of the simulation must typically be customized for the scenario of interest to the study.
    - *Emulation models* bear some similarity to simulated models, but the actual software components and virtualized hardware components (e.g. routers, servers, workstations, NIC cards, etc.) must be explicitly identified. The emulation platform we used for SECURE was minimega [9]. Below we specify steps that are fairly general and need to be customized for a particular emulation platform and experiment.
        i. Define or import the topology
        ii. Develop the application components, if needed
        iii. Define the experimental behaviors that will be investigated
        iv. Develop a data collection strategy
        v. Set up and verify the configuration
        vi. Obtain the resources to run the model

4. **Validate the model.** Compare the model to higher fidelity representations and/or to independently developed models of similar fidelity, to assess the degree of agreement between your model and the benchmark. Choose the comparison metrics that best expose the statistics of interest (e.g. differences due to virtual machine artifacts). A high-fidelity model (e.g. simulation or emulation) should ideally be benchmarked against an actual physical system, as in [13]. However, lower-fidelity models (e.g. mathematical) might be

benchmarked against higher-fidelity models. Any large and/or systemic differences between your modeled data and the benchmark data should be investigated before the experiment progresses.

At present, there is no standard for benchmarking cyber emulations; the current best-practice is a hierarchical validation, which occurs in stages, as shown in Figure 3. First, the components and/or attack steps are validated individually, then larger groupings or components are validated, and then the entire system is validated. Figure 3 depicts the validation of a cyber attack model, but a similar validation process could be applied to any kind of performance issue or behavior.

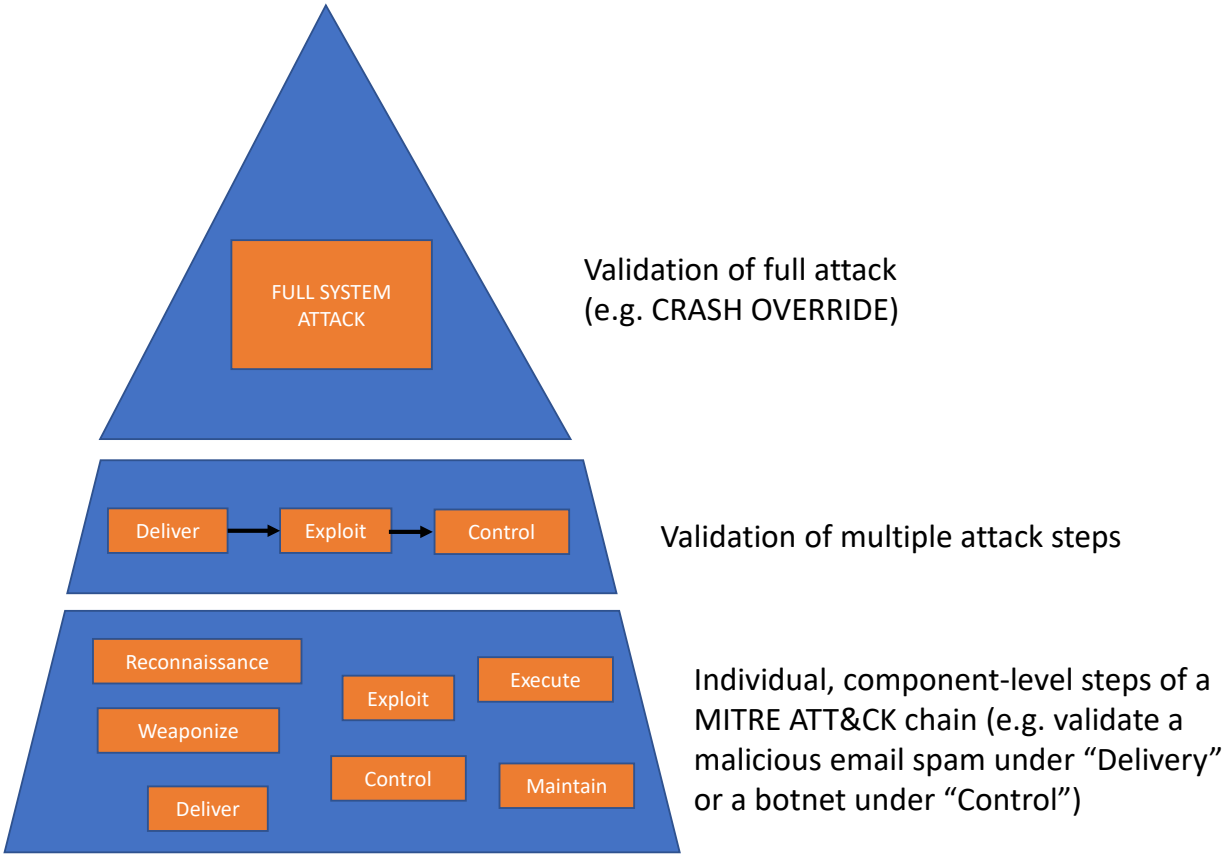


Figure 3. Hierarchical validation for a cyber system, starting with validation of individual attack steps at the bottom and proceeding to validation of the full attack at the top.

5. **Define and run the experiment.** Define the inputs/outputs for your model and specify them in a configuration file for an experimental orchestrator (e.g. Scorch or Dakota [10]). Choose an experimental design that will produce an appropriate list of input/output parameter settings:

a. Define the inputs that will be varied in the experiment and specify the distribution of possible values for each input (e.g. discrete bandwidth values, uniformly distributed traffic generation rates between upper/lower bounds, etc.). Each input that will be varied in the experiment should have a specification of its distribution in a parametric or empirical distribution form.
b. Define the outputs that will be extracted from the experiment. These outputs can take the form of detailed experimental data (e.g. packet captures and logfiles sent to an Elasticsearch/Logstash/Kibana (ELK) data collection node [11]), and/or summarized experimental outputs calculated within the experiment as it executes (e.g. the time at which an intrusion detection system generates an alert).
c. Develop the experimental design. This can be done in a variety of ways [8]. If the number of inputs is small (1-5) and each input has only 2 or 3 levels, a full factorial design can be run involving all combinations of input parameter levels. If the inputs are specified with continuous distributions, Monte Carlo sampling or more efficient alternatives such as Latin Hypercube sampling or quasi-Monte-Carlo space-filling methods can be used to generate samples. In each of these cases, the number of samples should typically be at least 10x the number of input parameters.
d. Define the number of replicates per design point. At each point in the experimental design space (e.g. input 1 is at value A, input 2 is at value B, etc.), it may be necessary to run the model multiple times, where each model run is a replicate. If the model is deterministic (e.g. running at one setting of parameter inputs always gives the same results), then it is only necessary to run the model once per parameter setting. However, many cyber models are stochastic due to slight variations in timings of processes and order of operation executions. In this case, one setting of the parameter inputs should be run with replicates to obtain statistics on the response for that parameter setting.
e. Run the model. Once the experimental design is identified, it produces a list of input parameter settings at which the cyber model should be run. This list is given to the experiment orchestrator (e.g. Scorch, Dakota). The next step is to run the cyber model at these settings. For each parameter setting, the model may be run once or some number of times (multiple replicates), depending on whether the model is deterministic or stochastic.

6. **Analyze the experimental results.** Use your data to generate a table (as an Excel spreadsheet, a data structure in a Python analysis script, a table in Elasticsearch, a table in Minitab [12], etc.) and organize the results (where the rows are each run of the cyber model, the first set of columns are the input parameters, and the second set of columns are the outputs) for further analysis.
   a. Verify results. Depending on the experimental design and the available benchmarks, choose the most appropriate validation method (e.g. scatterplots of inputs v. outputs, calculation of basic statistics on the outputs, etc.).
      i. (Optional) If the values obtained in Step 6(a) are orders of magnitude different from the benchmark values, revisit Step 3.

b. Assess convergence
    i. (Optional) If the values obtained in Step 6(b) are orders of magnitude different from the benchmark values, revisit Step 3.
c. Determine conclusions/insights. Employ statistical analysis methods appropriate to the experimental design (e.g. main effects analysis for full factorial designs with discrete input levels, correlation analysis, standardized regression analysis, and/or Sobol variance-based indices for designs with continuous input distributions). Statistical tests (e.g. t-tests or Kolmogorov-Smirnov tests) can be used to compare the results gathered from different tests, scenarios, platforms, or emulators.

7. **Document.** Document your results comprehensively so that they will be fully useful and reproducible for subsequent researchers.
    a. Question(s). List the question(s) addressed in the study.
    b. Methods. Define each step of the methodology, with enough detail that the study can be easily replicated.
    c. Analysis. Describe the analyses performed.
    d. Results. Report the complete results, including tables of raw data.
    e. Conclusions/Insights. Highlight the conclusions/insights gained from the study.

References for Workflow

1. https://omnetpp.org
2. https://www.nsnam.org
3. Mirkovic, J., Bartlett, G. and J. Blythe. *DEW: Distributed Experiment Workflows.* USC Information Sciences. Proceedings from USENIX/CSET 2018 Conference.
4. Bernard Ferguson, Anne Tall, and Denise Olsen. 2014. National cyber range overview. In 2014 IEEE Military Communications Conference. IEEE, 123–128.
5. Christos Siaterlis, Andres Perez Garcia, and Béla Genge. On the use of emulab testbeds for scientifically rigorous experiments. IEEE Communications Surveys & Tutorials, 15(2):929–942, 2012.
6. Jelena Mirkovic, Terry V Benzel, Ted Faber, Robert Braden, John T Wroclawski, and Stephen Schwab. The DETER project: Advancing the science of cyber security experimentation and test. In 2010 IEEE International Conference on Technologies for Homeland Security (HST), pages 1–7. IEEE, 2010.
7. Maricq, A., Duplyakin, D., Jiminez, I., Maltzahn, C., Stutsman, R. and R. Ricci. *Taming Performance Variability*. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2018
8. Swiler, L., Stickland, M, and T. Tarman. Design of Experiments for Cyber Emulation. Sandia National Laboratories Technical Report SAND2019-5640C. May 2019.
9. https://minimega.org/
10. https://dakota.sandia.gov
11. https://www.elastic.co/what-is/elk-stack
12. https://www.minitab.com/
13. Stephen T Jones, Kasimir G Gabert, and Thomas D Tarman. "Evaluating Emulation-based Models of Distributed Computing Systems." Technical Report SAND2017-10634, Albuquerque, NM (United States).

SECURE Tools

This page documents the key software tools developed and/or used as part of the SECURE project.

SCORCH

SCORCH is an automated scenario orchestration framework for emulation-based models that also utilizes minimega. The key benefits of SCORCH are that 1) it will configure the experiments and 2) it is able to collect and store the outputs, thereby speeding up analysis time and reducing manual error.

minimega

minimega (https://minimega.org/) is an open source distributed Virtual Machine (VM) management tool used for launching and managing virtual machines locally or across a cluster. minimega is fast, easy to deploy, and can scale to run on massive clusters with virtually no setup. It is scalable and able to support studying both small and very large VM networks. minimega is designed to give you low-level control of all the fine details when it comes to setting up and running VMs and has now been pulled into other tools, e.g. SCEPTRE, to take care of the low-level features of spinning up VMs.

SCEPTRE

SCEPTRE is an application that uses an underlying network emulation and analytics platform to model, simulate, emulate, test, and validate control system security and process simulations. Traditionally, tools and techniques for simulating and emulating control system field devices have been limited because the physical processes being monitored and controlled are omitted. SCEPTRE leverages proven technologies and techniques to integrate the end device and process simulations, with control hardware-in-the-loop (HITL), providing an integrated system capable of representing realistic responses in a physical process as events occur in the control system, and vice versa. SCEPTRE is a proven control system environment platform, having been fielded for many R&D applications, operational joint tests, and exercises supporting testing, training, validation, and mission rehearsal.

SCEPTRE is comprised of simulated control system devices, such as remote terminal units (RTUs), programmable logic controllers (PLCs), protection relays, and simulated processes, such as electric power transmission systems, refinery processes, and pipelines. The simulated control system devices are capable of communicating over Internet Protocol (IP) networks using standard SCADA protocols such as Modbus, DNP3, IEC 61850, and others. SCEPTRE also includes support for HITL, wherein real field devices under study (i.e. a specific model of PLC) can be connected to and interact with the physical process being simulated. This allows the user to include high fidelity systems where they are needed without sacrificing scalability. SCEPTRE provides an analysis capability for assessing and improving the cyber security of control systems used in the energy sector and DoD. The SCEPTRE platform provides an environment where hardware and software upgrades and new mitigations can be evaluated before installation in an operational environment.

Elasticsearch

Elasticsearch (https://www.elastic.co/elasticsearch/) is an open source tool for storing large amounts of data in a highly searchable way that is amenable to a variety of data types and structures. Under SECURE, Elasticsearch was leveraged for data storage and retrieval during the Validation and Verification studies. These studies required large amounts of data to be stored, sorted, and easily searchable. Using Elasticsearch allowed for storage of varied data types and structures, easy conversion of data to and from JSON format, and simple querying.

Dakota

Dakota is a suite of iterative mathematical and statistical methods that interface to computational models or simulations (https://dakota.sandia.gov). Dakota's goal is to make parametric explorations of models practical to support design, analysis, or test cycles. Dakota is an open-source software toolkit and has algorithms to enable design exploration, model calibration, risk analysis, and quantification of margins and uncertainty with computational models. Dakota seeks to enhance the use of computational models with a variety of iterative analyses (running the model multiple times depending on the objective of the study) so that models may be used not just for single-point solutions, but also achieve broader impact in the areas of credible prediction and optimal design.

Related to SECURE, there is an extensive suite of uncertainty analysis methods in Dakota, including a variety of sampling methods (Monte Carlo, Latin Hypercube Sampling, quasi-Monte Carlo methods, design of experiments, fractional and full factorial designs), sensitivity analysis methods, reliability methods, stochastic expansion methods such as polynomial chaos, epistemic uncertainty approaches including interval analysis and Dempster-Shafer evidence calculations, and Bayesian calibration methods, and multifidelity uncertainty methods. These are summarized in [L. P. Swiler, B.M. Adams, and M.S. Eldred, "Dakota: Bridging Advanced Scalable UQ Algorithms with Production Deployment." In Springer Handbook on Uncertainty Quantification, Ghanem R., Higdon D., Owhadi H. (eds) (2015). https://doi.org/10.1007/978-3-319-11259-6_52-1.].

PAO/Pyomo

PAO is a Python-based package for Adversarial Optimization. The goal of this package is to provide a general modeling and analysis capability for bilevel, trilevel and other multilevel optimization forms that express adversarial dynamics. Many planning situations involve the analysis of a hierarchy of decision-makers with competing objectives. For example, the cyber-grid applications developed in the SECURE Grand Challenge consider the behavior of attackers and defenders, where defenders wish to protect their cyber infrastructure and execute power grid operations to meet expected energy demands, and attackers wish to maximally disrupt grid operations. Thus, these cyber-grid applications can be naturally modeled as bi-level and tri-level optimization problems, where decision-makers need to account for the behavior of adversaries at a lower-level.

SECURE researchers developed tailored optimization solutions for cyber-grid applications using the Pyomo modeling environment, which are analyzed with commercial and open source

optimization solvers. Concurrently, PAO was developed to automate these tailored solutions to future applications that share similar structure. PAO extends the modeling concepts in the Pyomo algebraic modeling language to express problems with an intuitive algebraic syntax. Additionally, PAO supports compact problem representations that simplifies the implementation of solvers for bilevel, trilevel and other multilevel optimization problems. PAO currently includes four solver interfaces that are applicable to different classes of adversarial optimization problems.

- **Pyomo**
  - GitHub repository: https://github.com/Pyomo/pyomo
  - Online documentation: https://pyomo.readthedocs.io/en/latest/
  - Bynum, M., G. Hackebeil, W. E. Hart, C. Laird, B. Nicholson, J. Siirola, J.-P. Watson, and D. L. Woodruff. (2021) Pyomo: Optimization Modeling in Python. 3rd. Springer.
- **PAO**
  - GitHub repository: https://github.com/or-fusion/pao
  - Online documentation: https://pao.readthedocs.io/en/latest/
  - Hart, W. E., A. Castillo, E. S. Johnson, and S. Punla-Green (2021). PAO 1.0: A Python Library for Adversarial Optimization. Tech. rep. SAND 2021–6720. Sandia National Laboratories.

Command and Control (C2) Handbook

Overview

Over the last few decades, a variety of emulation tools have been developed to perform cyber experimentation. Despite this progress, relatively little attention has been devoted to developing methods that ensure the quality of experiments based on these capabilities. In this article, we demonstrate how the mathematical modeling, verification, validation, and uncertainty quantification methods, developed under SECURE, can be used in combination with emulation modeling to perform rigorous experimentation for a Command and Control (C2) cyber-attack. To our knowledge this exemplar demonstrates a level of experimental rigor and detail that has not been previously done for this kind of case study.

Recall that the full end-to-end exemplar studied in SECURE considers a multi-stage attack in which an attacker attempts to access a power utility's cyber control network and ultimately disrupt operations by causing load shed using the attack stages shown in Figure 4. Here we focus on the second step where an attacker aims to maintain C2 communications between an infected host and C2 server in order to pivot to other hosts and/or the ICS network. To counter this, the system owner uses an intrusion detection system (IDS) to identify malicious C2 traffic and take steps to remediate the infection to prevent disruption of physical operations.

Figure 4: Multi-stage attack considered by SECURE

The goals of this study fall into the following two categories: *application objectives* related to analyzing malicious C2 traffic in a cyber system and *SECURE research objectives* related to methods for cyber experimentation. Given this, we consider the following:

- Application objectives:
  - How long does it take to detect a C2 channel?
  - How does background traffic affect detection?
  - Which factors have the largest impact on the performance of an IDS system?
- SECURE research objectives:
  - What emulation capabilities are required to adequately represent this scenario?
  - Can we develop an approximate mathematical model of the emulation to analyze this scenario?
  - How can we validate the math model against the emulation?
  - What is the benefit of a math model?
  - Can the emulation and math model be used in conjunction to support analysis?

Analysis Scenario

In this study, we focus on detecting C2 malware traffic within the enterprise network portion of an electrical power utility. Figure 5 illustrates the system being analyzed. We assume that one or more hosts within the network have been infected and are communicating with an external C2 server. The internal network contains both benign and malicious network traffic, all of which is sent through a single router and switch. An IDS that monitors traffic to and from the network. The IDS performs packet inspection and issues an alert if the contents of an individual packet appears suspicious, according to one or more of its rules. We assume that it is possible that benign traffic may cause the IDS to issue an alert (i.e., a false-positive). In instances where there are large packet flow rates, the IDS may not have sufficient capacity to scan all packets [1]. In this case, unscanned malicious packets will still reach their destination without causing an alert.

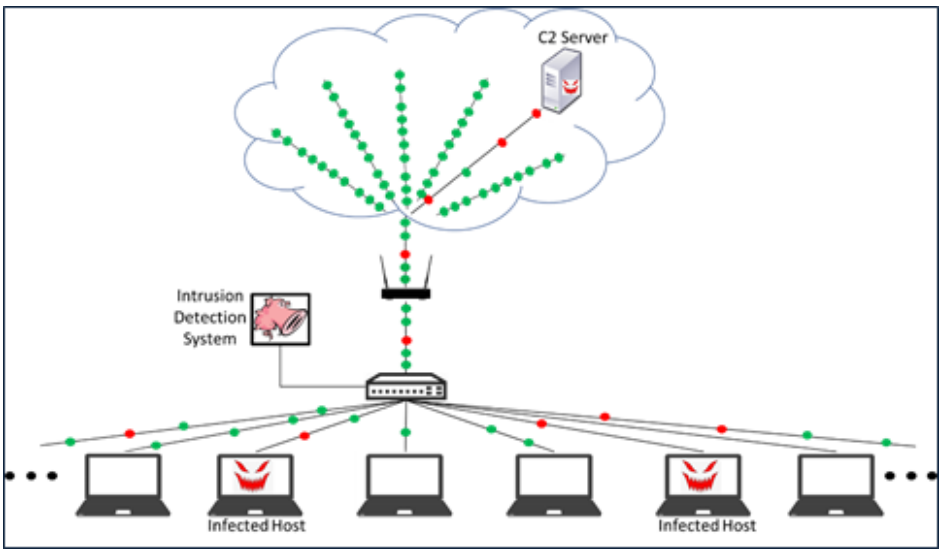


Figure 5: Notional C2 exemplar system representation

For this study we analyze C2 communication from the Emotet malware and its detection by the Snort IDS. Emotet was first discovered in 2014 as a banking Trojan. Since its initial discovery, Emotet has infected more than 1 million computers and caused hundreds of millions of dollars in damage [2]. Most antivirus and IDS programs have some sort of mechanism to detect an Emotet infection. For the Snort IDS alone, dozens of rules have been written to detect Emotet.

Though this study is motivated by and focuses on specific Snort and Emotet features, the work discussed below is not unique to this IDS or malware. Rather, we believe the capabilities presented below could be generally applicable to any IDS and malware combination in which the IDS generates alerts based on individual packet inspection. Consideration of different IDSs and alerts would merely require alternate parameterizations.

Given the goals of the attacker and the defender, the key Quantities of Interest (QoIs) are the alert rates (i.e., number of alerts issued at a point in time) for both malicious and benign traffic, under various network, attack, and IDS configurations. We recognize that issuance of an alert does not

necessarily equal detection; detection generally requires a combination of alerts and human recognition that the alerts are indicative of a problem. Modeling the human element of detection is beyond the scope of this work, so, instead, we assume that a detection occurs when a large enough number of alerts are issued that network administrators would reasonably determine that the anomalous traffic is malicious. Hence, the primary focus of this work is accurately modeling alerts over time and not establishing detection thresholds.

The remainder of this article provides an overview of how the SECURE experimentation methodology was applied to the C2 malware problem. The following summarizes the process that we used to analyze the C2 problem. For each of steps described, detailed tutorials and technical documentation are also available.

1. Emulation model development: Create a high-fidelity "ground truth" model using emulation.
2. Emulation model verification: Build confidence that the emulation models are working as intended.
3. Mathematical model development: Create a low-fidelity statistical model surrogate for the emulation model.
4. Mathematical model validation: Assess the validity of the low-fidelity model using statistical tests for discrete, time-series data to ensure that the inexpensive mathematical model can be used as a proxy for the more costly high-fidelity emulation model.
5. Analysis and Uncertainty quantification:
    1. Efficient sampling: Use Polynomial Chaos Expansion (PCE) to efficiently sample the input parameter space using the mathematical model to identify which input parameters have the largest effect on the QoIs.
    2. Multi-fidelity uncertainty quantification: Integrate results from low- and high-fidelity models to improve the accuracy of the QoIs with minimal experimentation costs, for the key parameters identified using PCE.

C2 Emulation Environment

We model the C2 environment using emulation, a capability primarily used to model distributed communication networks. As the name implies, emulation models aim to replicate high-level functionality of target networks using emulated hardware components. Abstraction of the hardware layer serves to facilitate implementation of these “logical network replicas” at reduced costs. A typical emulation environment consists of a set of virtual machines that are networked together using virtual switching. The entire environment is supported by a cluster of hardware servers. Emulation environments serve a variety of purposes such as testing, evaluation, training, and experimentation. Because of their heavy use of virtualization, large network environments can be deployed, torn down, and redeployed to an original state with relatively little effort. This makes emulation environments particularly well-suited for repeatable and reproducible experimentation of distributed communication networks. There are several tools available for creating, deploying, and managing emulation environments, including two created at Sandia National Laboratories: `minimega` and SCORCH. Sandia's `minimega` tool is used for launching and managing virtual machines locally or across a cluster. SCORCH is an automated scenario orchestration framework for emulation-based models that utilizes `minimega` to deploy and instrument experiments.

We created the emulation model for this study using `minimega` and SCORCH. The environment model is comprised of the following primary components, as shown in Figure 6: a malware traffic generator (attacker), an IDS (defender), and the background traffic generator (environment). Each component has parameters that can be adjusted and tuned for various experiment iterations. The malware traffic is generated via custom Python code that enables researchers to modify the message features, size, and frequency of the generated packets. Rather than represent each machine with an individual host, we use a single device to generate "aggregate traffic" representative of the total traffic we would see from multiple hosts. For this scenario, the malware traffic generator is calibrated to mimic the packet structure of the Emotet malware message format, encrypted structure, and C2 timing (using the 2018/2019 variant of Emotet). The signature of the Emotet network traffic has been previously researched and captured in detection rules [2,3]. Snort is used as the IDS and implements Emotet-specific detection rules to alert on Emotet-based packet signatures. The IDS component can be tuned for different detection algorithms/rules, memory availability, and processing speed. To increase the scenario's fidelity and provide a realistic network for experiments, background packets are created and sent from a client to a server via a custom Python script. The background traffic message format, packet size and frequency can be modified per experiment.

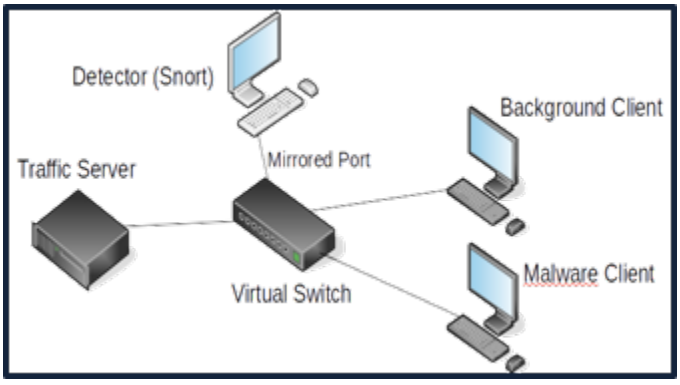


Figure 6: C2 Exemplar Emulation Environment

For this study, we focus on the parameters shown in Table 1. These parameters can be binned into four groups. The *general parameters* describe basic parameters of the test environment. The *IDS parameters* define the capacity and characteristics of the IDS. The *background traffic parameters* specify the intensity of the background traffic and the false-positive rate. *The malicious traffic parameters* specify the intensity of the malware traffic and the false-negative rate. For each of these parameters, we indicate the value or the range of the values that the parameter can take. For those values that are uncertain, we assume they follow a continuous or discrete probability distribution, as indicated in the Distribution column. Even for this relatively modest sized problem, many parameter configurations can be explored. Note that some of parameters listed in Table 1 cannot directly be controlled in the emulation environment, as specified in the Comments column.

Table 1: Key variables of interest for the C2 study.

Parameters	Units	Value	Distribution	Comments
General Parameters				
Total number of workstations	No units	10	Fixed	Variable type: input parameter Basis: selected to represent "moderately" sized portion of a corporate network
Average packet size	Bytes	150-250	Continuous uniform	Variable type: observed quantity Basis: packets observed in the experiments had an average size of 200 bytes in experiments; +/-50 bytes is selected to permit variability across experiments
IDS Parameters				
Snort capacity	Bytes per second	1e5, 2e5, 5e5, or 1e6	Discrete with equal probability	Variable type: input parameter to emulation model Basis: selected to represent "moderately" sized portion of a corporate network
Number of CPUs	No units	8	Fixed	Variable type: input parameter Basis: expert judgement and known hardware configurations
Number of CPUs to maximize Snort	No units	1-8	Discrete with equal probability	Variable type: input parameter Basis: positive integers bounded by total # of CPUs
CPUs running other (non-Snort) processes	No units	0-7	Discrete with equal probability	Variable type: input parameter Basis: positive, integers bounded by total # of CPUs

Drop rate multiplier	No units	0.9-1.1	Symmetric continuous triangular distribution	Variable type: observed quantity Basis: expert judgment used to assess the actual drop rate, which could be +/- 10% difference from the calculated rate
Background Traffic Parameters				
Benign traffic per host	Packets per sec	5-100	Continuous log-uniform	Variable type: input parameter Basis: 100 pps per host (with 20 hosts) results in 2000 pps for total traffic. This amount represents the upper limit on the traffic generator's capacity and is comparable to (and may exceed) congested TCP traffic conditions used in other IDS evaluation literature (e.g., [4] and [5]). The lower bound was selected to represent a minimal level of traffic for evaluation.
Fraction of benign packets with Emotet signatures.	Fraction of packets per sec	1e-5-1e-3	Continuous log-uniform	Variable type: input parameter Basis: expert judgment because published values were not available; selected values are relatively small to indicate the small probability that the Emotet signature would occur due to spurious conditions
Detection rate for signatures in regular, benign traffic (if signature is present)	No units	0.9-0.99	Continuous uniform	Variable type: observed quantity Basis: we observed an average detection rate of 0.95 when we used the Snort rule to evaluate actual Emotet traffic packet captures (pcaps) and simulated Emotet traffic in emulation experiments; range was expanded to 0.9-0.99 to permit variability across experiments

Malicious Traffic Parameters				
Number of infected workstations	No units	0-10	Discrete with equal probability	Variable type: input parameter Basis: non-negative integer, bounded by total number of hosts
Malware traffic per infected host	No units	4-10	Continuous uniform	Variable type: input parameter Basis: published observations and analysis of actual Emotet traffic pcaps
Fraction of malware packets with Emotet signatures	No units	0.1-0.2	Continuous uniform	Variable type: input parameter Basis: analysis of Emotet traffic pcaps and structure of TCP traffic
Detection rate of signatures for malware traffic (if signature is present)	No Units	0.9-0.99	Continuous uniform	Variable type: observed quantity Basis: we observed an average detection rate of 0.95 when we used the Snort rule to evaluate actual Emotet traffic pcaps and simulated Emotet traffic in emulation experiments

Emulation Verification using Telemetry

An important aspect of using emulation is verifying whether the emulation environment is working as intended. For this study, we approach the verification problem using the same strategy that was employed in the SCADA study. The core idea of this approach is to monitor performance metrics while intentionally stressing the emulation environment to identify potential issues. This monitoring process is called telemetry [2-5], which includes metrics like server load and availability, disk space usage, memory consumption, performance, etc. Though many aspects of the emulation could be verified, we focused on determining whether sufficient virtualized resources are available to support the scenario because insufficient resources can cause experimental outcomes to be unrepresentative or incorrect.

In this study, we run the C2 scenario under various levels of over-subscribed resources. We start with a baseline scenario where there is only one namespace running on a physical host. We then consider five scenarios where an increasing number of namespaces (2, 5, 10, 20, and 40) are run in parallel on the same physical host. As the level of over-subscription increases, we aim to identify metrics that can signal that a particular emulation experiment is unreliable. An experiment is

unreliable if an output QoI is likely to have been affected by the emulation configuration. In our case, the QoIs are the number of alerts present a four timesteps (1, 5, 10, and 16 seconds). If sufficient resources are unavailable when an experiment is run, the resulting QoIs cannot be trusted, and the data should be excluded. Table 2 shows the six cases considered in this study. In all cases, a total of 200 replicates are performed, where a "replicate" represents a single iteration of the scenario running in emulation. In the baseline, the 200 replicates are run in series on the single namespace considered. In the two-namespace case, two replicates are run simultaneously over 100 iterations, to obtain the 200 replicates. The same idea is used in the other scenarios to ensure that the product of the number of parallel namespaces and iterations is equal to 200 replicates.

Table 2. Summary of the Six Analysis Scenarios in the C2 Verification Study

Number of Namespaces Running in Parallel	1 (Baseline)	2	5	10	20	40
Iterations per Analysis Scenario	200	100	40	20	10	5

To assess the reliability of the emulation environment we focus on both the QoIs, the number of alerts at timesteps 1, 5, 10, and 16, and the telemetry metrics. While several telemetry metrics were considered during the original analysis, we will focus on the load metric. Load 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. Our threshold for acceptability is instances where the load in an experiment stays below 32.

The results of the baseline set of replicates are used to determine the acceptable range of values for the QoI. This process helps to determine whether the results of a particular replicate are likely to have been affected by the emulation configuration and should therefore be discarded. We treat each timestep as its own QoI when determining acceptability, so there are four metrics for each replicate. We test the statistics by first filtering out any replicates in which a chosen threshold was violated for that statistic. We can then examine the distribution of the QoI for the remaining replicates, using a statistical t-test to compare these distributions between different emulation configurations. The better the metric is, the more closely we expect the distributions to all match, and the higher the p-values from the t-test should be. Because the C2 scenario has four QoIs per replicate, the t-test is performed for each QoI individually, and then the resulting p-values are aggregated. We found the mean to be the best aggregation, but we include the minimum in the results below for comparison.

Table 3 shows the verification results for the C2 scenario when no replicates are filtered and when replicates are filtered for instances where load exceeds 32. A common threshold for rejecting a

null hypothesis, in this case that the hypothesis that two experiments are the same, is when the p-value falls below 0.05. Given the p-values in Table 3, we could not say that running the C2 experiment with 2, 5, or 10 namespaces in parallel causes the results to deviate significantly from the baseline. In the 20- and 40-namespace cases, the p-values indicate that the distribution for the number of alerts generated is different from the baseline case. For these scenarios, 20 and 147 replicates did exceed the load threshold of 32 and were therefore filtered out of results.

Table 3. Results of the analysis scenarios (baseline compared to the five remaining scenarios).

Telemetry	Threshold	Metric	2	5	10	20	40
None (include all replicates)	N/A	Number of Replicates	200	200	200	200	200
		Mean (Min) p-Value vs. Baseline	0.9 (0.9)	0.9 (0.9)	0.9 (0.9)	0.007 (0.001)	0.001 (0.001)
Load	< 32	Number of Replicates	200	200	197	180	53
		Mean (Min) p-Value vs. Baseline	0.9 (0.9)	0.77 (0.39)	0.9 (0.9)	0.001 (0.001)	0.001 (0.001)

These results suggest that oversubscription is potentially an issue in the 20- and 40-namespace cases, and that telemetry data related to load might be a useful indicator that oversubscription has occurred. While these results demonstrate a methodology for performing verification, further research is needed to answer a variety of remaining questions. For example, which load threshold is best and, in addition to load, are there other telemetry metrics that could be considered (stolen cycles, number of context switches per second, etc.). There may also be opportunities to combine metrics into a single indicator, as well as to use real-time metrics to discard unreliable experiments while they are running.

Mathematical Model

Emulation testbeds provide a safe, high-fidelity environment for conducting cyber experiments. However, since these testbeds run real software and protocols, the experiments typically need to be executed in real-time. This can be time-prohibitive in instances where:

- Scenarios evolve over long time periods.

- Analyses include features of the system may be unknown or vary, or in which the analyst aims to characterize a potentially wide range of possible outcomes.
- Analyses consider stochastic behaviors and thus require many experiments to suitably characterize the relevant statistics.

Given the potential number of parameter setting that could be explored (see Table 1), we developed a low-fidelity statistical model that can be run significantly faster than the real-time emulation model. The model can be most easily described through an analogy, as depicted in Figure 7. Consider a water contamination sensor system that receives flows from various sources across a water transportation network. The flows may contain benign or beneficial matter like fluoride (normal network traffic) and also toxins like lead (malicious C2 messages). Water containing both good and bad matter flows into a reservoir tank and passes through a filter (IDS) that removes the toxic particles. The "cleaned" water is then distributed throughout the system. The filter may fail to catch some portion of the toxins (false negatives); it may also remove benign materials (false positives). The filtration system is rate-limited and has a finite reservoir capacity (memory). If the inflow rate exceeds the capacity of the sensor system, a bypass valve is activated, permitting the unfiltered water to circumvent the filter and pour directly into the system without filtration. See [11] for a full description of the mathematical model.

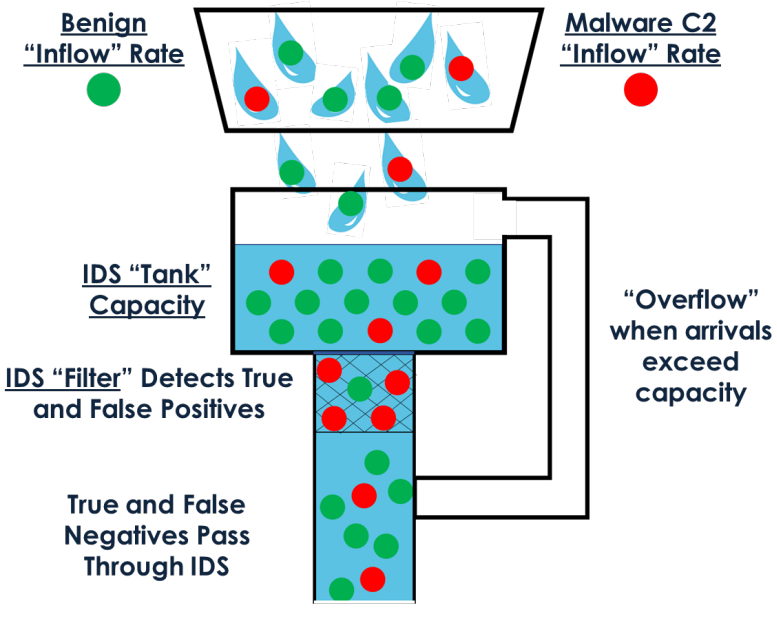


Figure 7: Mapping between water filtration and intrusion detection systems.

The mathematical model of the IDS builds on the flow/filter concepts to represent network traffic as an influx of packets from various hosts (flow) and the detection of C2 traffic by an IDS (filter). Most of the hosts are not infected with the malware, so the packets in their traffic is benign. Some hosts are infected by malware and generate packets that contains malicious C2 traffic. All packets are routed through a device running an IDS, whose signature-based rules act as a filter: if the rule identifies the malware signature from a malicious packet (true positive), the IDS issues an alert. Detection of malicious traffic is not perfect, so some malicious packets pass through without an

alert being issued (false negative). In some instances, the IDS may issue an alert for a benign packet (false positive), but most benign packets result in no alert (true negative).

The IDS is rate-limited in its capacity to process network traffic (i.e., the IDS has a threshold measured in packets/bytes per second) within a set time period. Hardware characteristics (e.g., number of CPUs, memory available), software features (e.g., types of detection rules being used by the IDS, computing requirements for individual rules, number of rules being used, degree of parallelization), and the number of other processes being run on the device (and computational requirements for the processes) all affect the IDS's capacity. In the most extreme cases (when network traffic rates far exceed the IDS's capacity), the IDS may eventually stop issuing alerts altogether until the memory buffer is cleared. In these instances, all packets will pass to their destinations without being inspected by the IDS, including any malicious C2 packets; because they are dropped, alerts are not generated for these packets, resulting in universal negatives (false and true).

The mathematical model integrates these concepts into a probabilistic, discrete-time representation to describe the C2 traffic and detection by the IDS. The key model inputs include:

- Packet arrival rates at the IDS for both benign and malicious traffic
- True and false positive rates (on a per-packet basis) for the IDS's signature-based rule
- Average packet size
- IDS capacity

When specific values are assigned to these inputs, the model produces the following two primary outputs: the average number of alerts that are expected over time (Figure 8), and the probability that at least N alerts will be registered by a point in time (Figure 9). Results can be produced for non-rate limiting (Figure 8 and Figure 9) and rate-limiting scenarios (Figure 10). Observe that in the latter case, the number of alerts levels off as the IDS reaches capacity. The total alert results can also be separated into false positive alerts and true positive alerts.

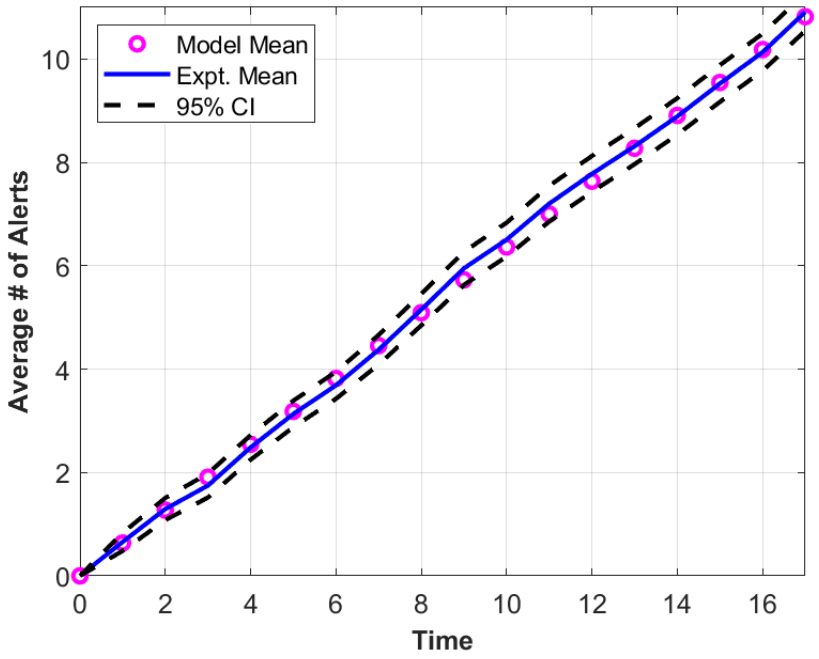


Figure 8: Average number of alerts over time, for the emulation and mathematical models (non-rate limited case).

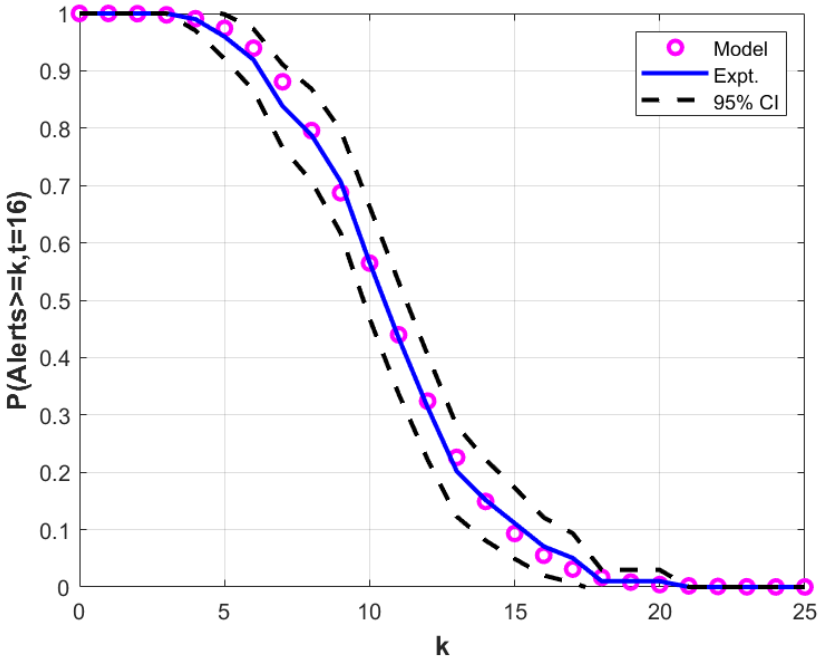


Figure 9: Probability of having at least k alters by time period 16, for the emulation and mathematical models (non-rate limited case).

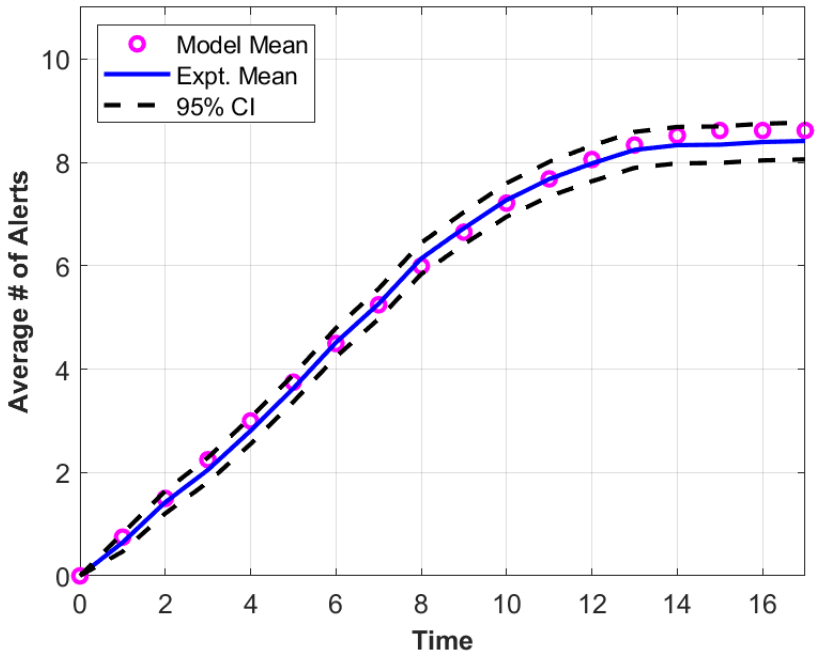


Figure 10: Average number of alerts over time, for the emulation and mathematical models (rate limited case). Note how the number of alerts levels off.

Comparison of Mathematical Model and Emulation Model Results

The mathematical model is validated against the results generated by the emulation model. Figure 8, Figure 9, and Figure 10 show the results for both models. For these particular results, a visual inspection shows a strong level of agreement between the two models, with the mathematical model results generally falling within the 95% confidence intervals of the mean value from the emulation model results. The data generated by the emulation model is both discrete (number of alerts triggered) and time-series (number of alerts per time-step). For example, a particular run might have 0 alerts triggered in the first second, 3 alerts after 5 seconds, and 7 alerts after 10 seconds. After enough of these emulation runs are collected, we can generate a cumulative distribution function (CDF) at each time step on the number of triggered alerts. In other words, we have a curve representing the probability that more than k alerts are generated by a given point in time. We compare the CDFs from both models using a more rigorous, statistical approach than visual comparison. We do this by using the Kolmogorov-Smirnov (K-S) test, a standard statistical test for comparing two distributions. Figure 11 shows an example of the model and experimental CDF curves at time period 9 for a particular C2 scenario.

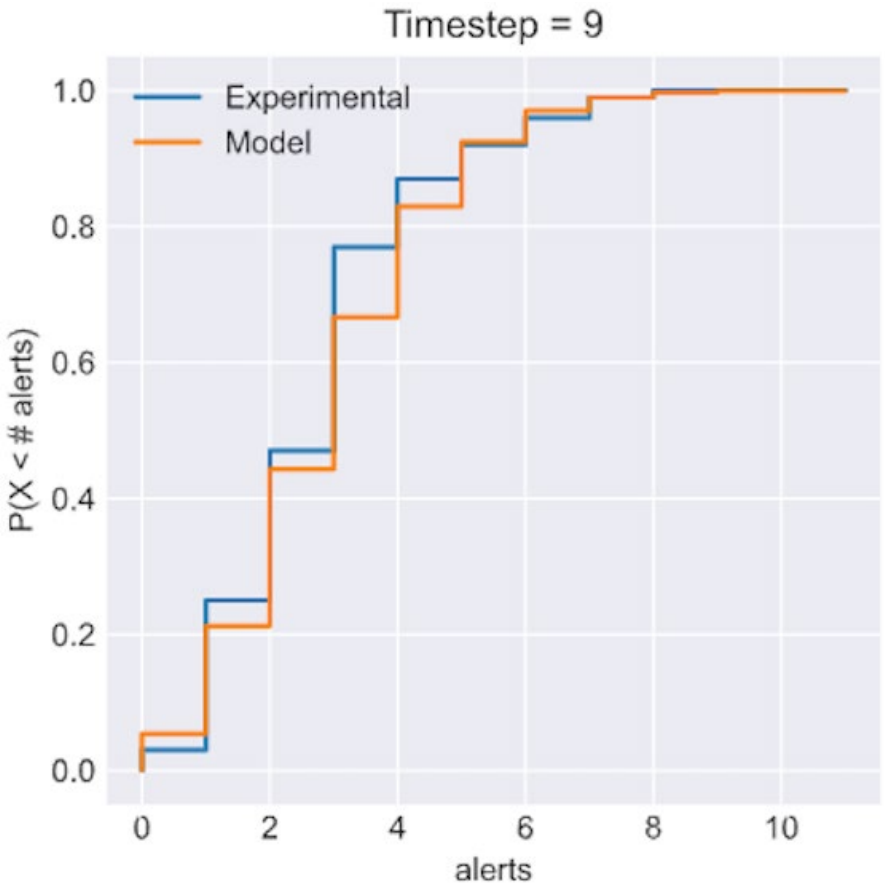


Figure 11: Comparison of the emulation and mathematical models CDFs for the probability of exceeding a given number of alerts by time period 9.

Using the K-S test, we can calculate a p-value for each time period, as shown in Figure 12. Observe that for time period 9, the p-value is about 0.2. A high p-value indicates that the null hypothesis, that the two CDFs are statistically similar, cannot be rejected. While the p-value dips around time period 9, it is still above 0.1 even at its lowest point. Given this, we would not reject the null hypothesis in this example.

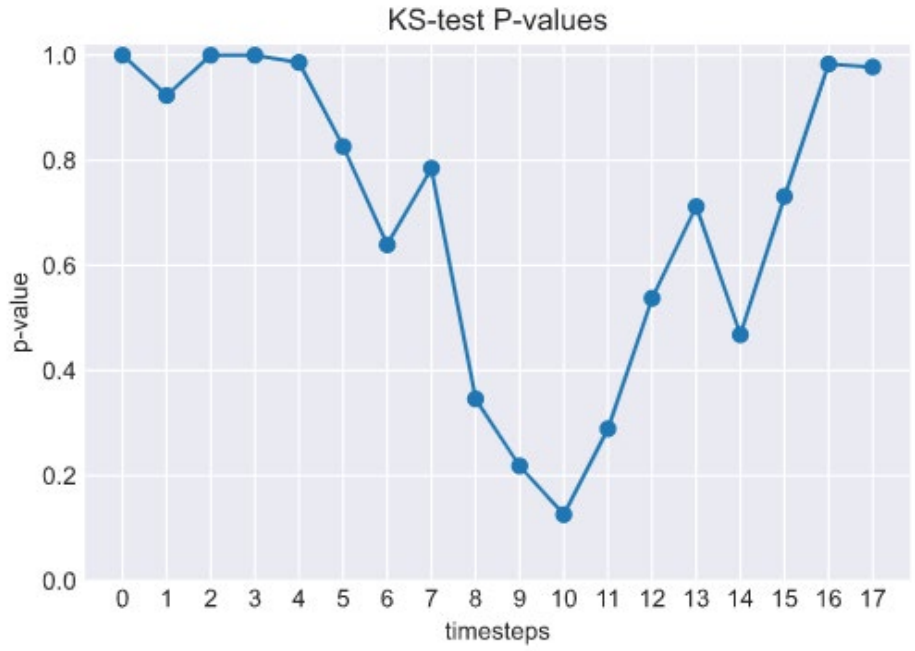


Figure 12: p-values for the K-S by time period.

In addition to the results shown above, we have compared the emulation and mathematical model results across a variety of parameter combinations. Though the results may not be perfectly identical, the combination of visual inspection and statistical comparisons provide confidence that the mathematical model is a reasonable proxy for the actual system and that it can provide reasonable estimates of alert statistics for the C2 scenario under consideration.

Analysis and Uncertainty Quantification

Given the high- and low- fidelity models, we next focus on uncertainty quantification (UQ) to understand how uncertainty in the input parameters propagates to the QoIs. We do this using two analysis methods: polynomial chaos expansion (PCE) and multi-fidelity UQ (MFUQ). We first use PCE to screen the 12 uncertain parameters shown in Table 1 to determine which parameters are the most important for more detailed study. The screening is done using the low-fidelity mathematical model to avoid the computational costs of using the emulation model. Once the key parameters are identified, MFUQ is used to analyze the QoI using a combination both models.

PCE Sampling

In the UQ community, QoIs are commonly represented as a polynomial function of the uncertain inputs; this approach is referred to as a Polynomial Chaos Expansion (PCE) of the QoI. Provided that a QoI is a smooth function of the inputs, the smoothness in the polynomial representation can give an accurate representation with fewer samples than would be required with a Monte Carlo (MC) approach. Once a PCE is constructed, it can be used to determine the mean, variability, or

other moments of the QoI. PCEs can also be used to perform a Global Sensitivity Analysis (GSA) of the QoI with respect to each of the inputs. In other words, it can tell us which inputs contribute the most to the variability in the output.

One of the challenges of applying the PCE approach to cyber security experiments is that many of the input variables are discrete. For example, the number of infected nodes on a network, the number of CPUs on the host that runs an IDS, or the nominal network bandwidth of the node connections are all discretely valued. Therefore, we employ PCEs that have been tailored to discrete random variables and their probability masses. These tools have been implemented in PyApprox, a Sandia open source software package for uncertainty quantification [10].

We applied this approach to the QoIs of total alerts and false positives at time period 5, for the parameter distributions as in Table 1. This corresponds to a case with 12 uncertain parameters, 5 of which are discrete in nature. A third order PCE was trained on random samples of the QoI that were obtained with the C2 math model. Table 4 shows the main effect indices for both QoIs for the 12 uncertain parameters.

Table 4: Main effects from PCE analysis for the number of total alerts and false positives at time period 5.

Parameters	**Total Alerts, t = 5 sec.**	**False Positives, t = 5 sec.**
Number of infected workstations	**0.87**	0.00
Fraction of benign packets with Emotet signatures	0.00	**0.51**
Benign traffic per host	**0.01**	**0.20**
Malware traffic per infected host	**0.05**	0.00
Fraction of malware packets with Emotet signatures	**0.03**	0.00
Snort capacity	**0.01**	**0.01**
Other CPU Processes	**0.01**	0.00
Number of CPUs to maximize snort	0.00	0.00
Average packet size	0.00	0.00
Detection rate for signatures in benign traffic	0.00	0.00
Detection rate of signatures for malware traffic	0.00	0.00
Drop rate multiplier	0.00	0.00

Based on these results, the main parameter that impacts the value of total alerts is the number of infected hosts, with lesser contributions from the amount of malware traffic per infected host and the fraction of malware packets that show the Emotet signature. The number of false positive alerts is most sensitive to the amount of benign traffic per infected host and the fraction of benign traffic packets that show the Emotet signature.

Multi-Fidelity UQ

Next, we explore the use of MFUQ to make optimal use of the emulation model which has high fidelity but is expensive to run and the lower-fidelity mathematical model which can be evaluated quickly. MFUQ estimator is built starting from the single fidelity MC results ($Q_{minimega}$) and adding a weighted unbiased term which involves the lower-fidelity math model (Q_{math}). The benefit of this additional term is that it can reduce the variance of the QoI (see [12] for the technical details of this approach). Using this approach many samples from the low-fidelity mathematical model can be combined a relatively small number of high-fidelity emulation model results to decrease the estimator variance and obtain more accurate and reliable statistics, with reduced computational costs.

$$\hat{Q}^{MF} = \frac{1}{N}\sum_{i=1}^{40} Q_{minimega}^{(i)} + \alpha\left(\frac{1}{N}\sum_{i=1}^{40} Q_{math}^{(i)} - \frac{1}{r\times 40}\sum_{j=1}^{r\times 40} Q_{math}^{(j)}\right)$$
$$= \hat{Q}_{minimega} + \alpha\hat{\Delta}_{math},$$

Figure 13: C2 MFUQ estimator.

Based on the screening results from the PCE analysis, we focus on the five parameters shown in Table 5. A total of 40 samples of these parameters was used for this study. The emulation model required 18 hours (plus additional processing time) to perform a total of 400 emulation runs (the 40 unique parameter combinations with 10 iterations each). In contrast, the mathematical model required less than 1 second total for all 40 of the parameter combinations (0.4 s for all the samples). We note that the mathematical model is able to provide statistics for the QoI without being affected by any stochastic noise; therefore, we will compare the average from 10 emulation model replicas with the values from the mathematical model.

Table 5: Key parameters of interest for MFUQ study.

Parameters Varied in Experiment	**Units**	**Value**	**Distribution**
Aggregate Benign traffic rate	Packets per sec	100-3000	Continuous log-uniform
Fraction of benign packets with Emotet signatures.	No units	1e-5 to 8e-4	Log-uniform
Aggregate malware traffic rate	Packets per sec	10-20	Uniform
Fraction of malware packets with Emotet signatures	No units	0.01-0.025	Uniform

RAM assigned to the 1 CPU running SNORT	Mbytes	128, 256, 512, 1024	Discrete with equal probability

For this study, we consider the total number of alerts at time periods1, 5, and 10. We begin by performing a pilot study to compare the total number of alerts generated from both models. From Figure 14, we note that the correlation between both models is high, as confirmed in Figure 15 which shows the estimated squared correlation between the models at the time steps considered.

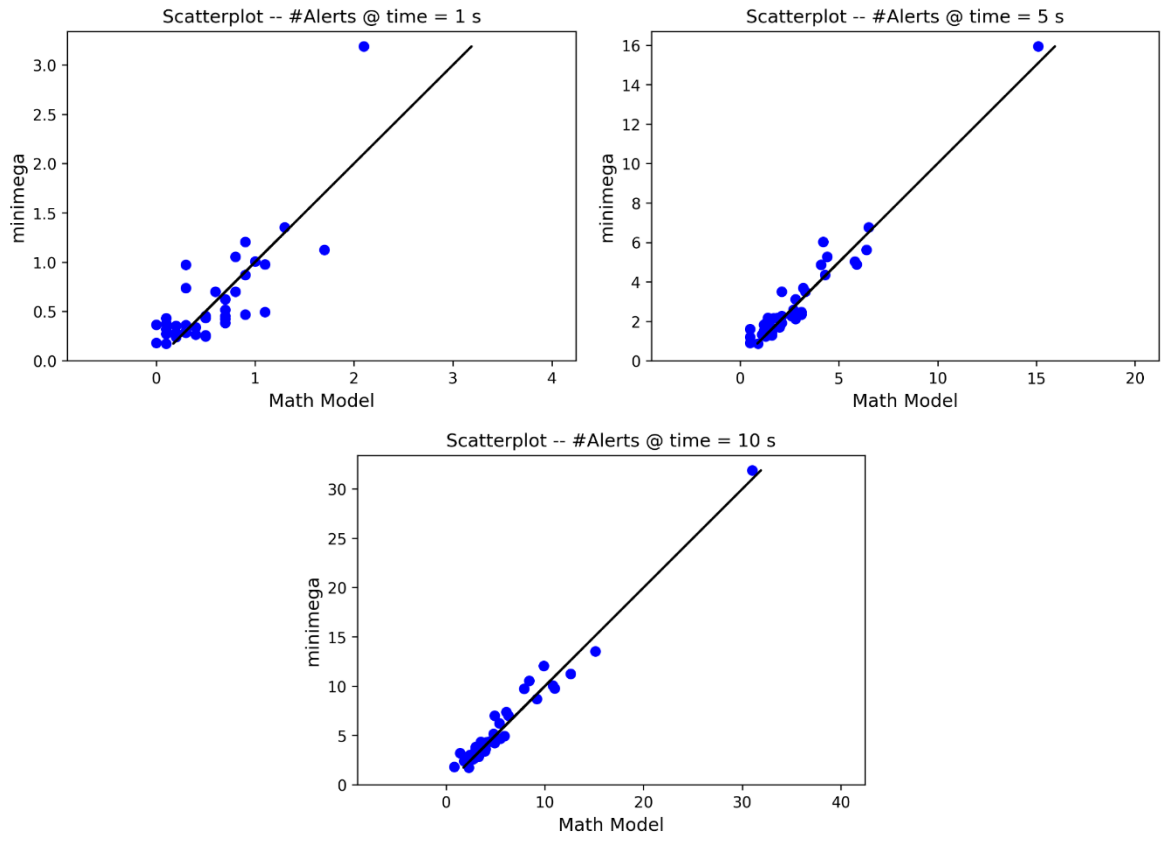


Figure 14: Scatterplots of total number of alerts at timesteps 1, 5, and 10 for 40 parameter samples for the emulation and mathematical models.

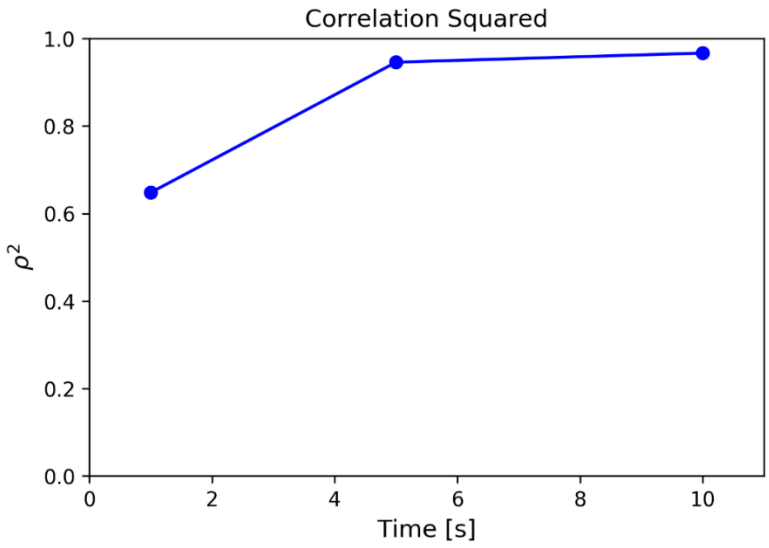


Figure 15: Correlation squared between the emulation and mathematical models at time steps 1, 5, and 10.

From the pilot study, it is possible to estimate the variance of the number of alerts, which is reported, along with the coefficient of variation, in Figure 16. We note that the variance of the number of alerts increases with time (as expected), and that the coefficient of variation (defined as the ratio between the standard deviation and the mean) approaches a value of 92%. By leveraging this information (relative to the computational costs of the two models and their correlation), we obtained the optimal number of mathematical model replicates that would be required to minimize the estimator variance for a fixed number of emulation model experiments. Due to the increase in variance with time, the most restrictive condition is obtained for the time of 10 seconds. At this time, the optimal estimator is obtained by using a total of 86840 mathematical model samples. By adding samples to the original 40 samples from the emulation model, we obtain an estimator with a total cost of 40.53 equivalent emulation model runs. It follows that we can reduce the variance of the estimator by only adding a fraction of the cost of a single emulation model run (0.53).

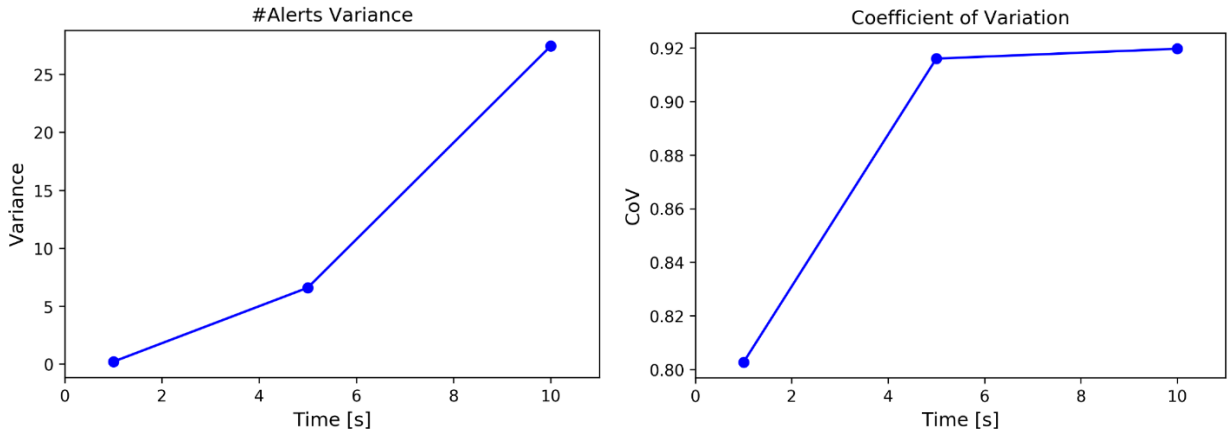


Figure 16: Variance (left) and coefficient of variation (right) for the total number of alerts.

In Figure 17, we report the mean number of alerts and the associated 99.7% confidence interval for the MFUQ estimator and the single-fidelity MC estimator. From the experiments, we can also evaluate the estimator variance, which was used to calculate the confidence intervals. We note that the variance reduction that the MFUQ estimators attains increases with time since the Multi-Fidelity estimator can maintain a high variance reduction with respect to MC, thanks to the increasing correlation between the models. The single MC estimator is not able to compensate for the increase in variance over time, and consequently, its confidence intervals grow more rapidly with progressively less accurate estimation for the mean number of alerts.

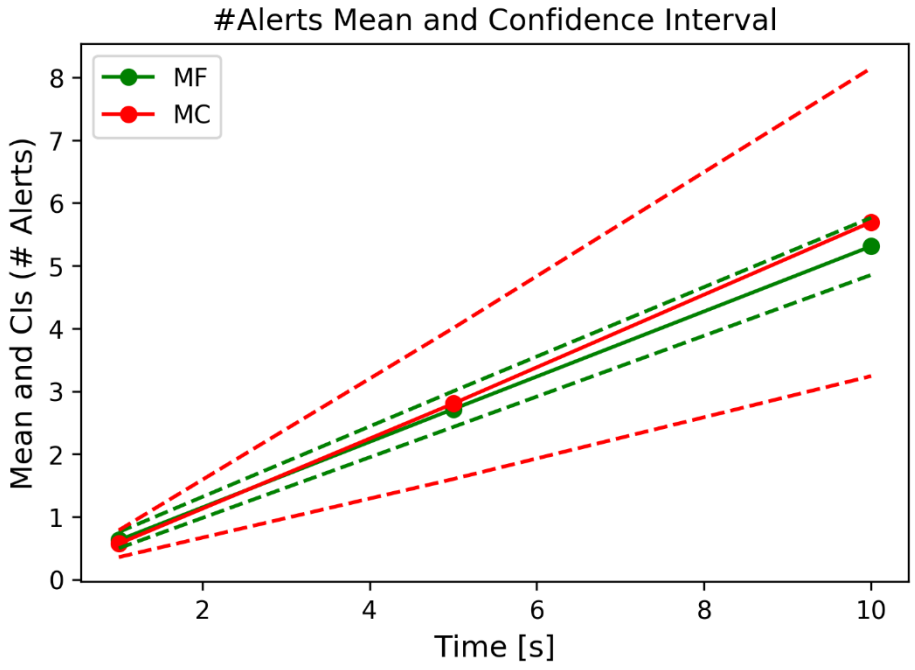


Figure 17: Prediction of mean number of alerts and associated confidence interval for single (MC) and multi-fidelity (MF) estimators.

Conclusions

This exemplar demonstrates how the capabilities developed under SECURE can be used to support rigorous cyber experimentation. Specifically, it shows:

- How experiments and metrics can be used to verify the behavior of emulation models.
- How to develop low-fidelity models to approximate high-fidelity models and how to validate the outputs of these models.
- How UQ methods can be used to efficiently explore input and output uncertainty.
- How high- and low-fidelity models can be combined to effectively utilize the experimentation budget.

References for C2 Case Study

1. Karim I, Vien Q-T, Le TA, Mapp G. A Comparative Experimental Design and Performance Analysis of Snort-Based Intrusion Detection System in Practical Computer Networks. *Computers*. 2017; 6(1):6. https://doi.org/10.3390/computers6010006
2. US CERT (2018). "Alert (TA18-201A) Emotet Malware." accessed October 21, 2020 at https://us-cert.cisa.gov/ncas/alerts/TA18-201A
3. "Snort Rules." accessed October 21, 2020 at https://snort.org/downloads/#rule-downloads
4. S. A. R. Shah and B. Issac. "Performance comparison of intrusion detection systems and application of machine learning to Snort system, *Future Generation Computer Systems,* 80(2018), 57-170.
5. W. Bul'ajoul, A. James, and M. Pannu. "Improving Network Intrusion detection system performance through quality of service configuration and parallel technology". *J of Computer and System Sciences,* 81 (2015), 981–999.
6. https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-common-metrics
7. https://cloud.google.com/network-telemetry
8. https://www.sumologic.com/insight/what-is-telemetry/
9. https://www.intel.com/content/www/us/en/cloud-computing/telemetry.html
10. https://github.com/sandialabs/pyapprox
11. Vugrin, E., J. Cruz, C. Reedy, T. Tarman, and A. Pinar. "Cyber Threat Modeling and Validation: Port Scanning and Detection," *Proceedings of the 7th Annual Hot Topics in the Science of Security (HoTSoS) Symposium*. Sept. 2020. https://doi.org/10.1145/3384217.3385626
12. Geraci, G., Crussell, J., Swiler, L.P. and Debusschere, B. J. "Exploration of Multifidelity UQ Sampling Strategies for Computer Network Applications." *International Journal of Uncertainty Quantification,* 2021. Pp. 93-118. DOI: 10.1615/Int.J.UncertaintyQuantification.2021033774. SAND2021-1221J

Scanning and Detection on a SCADA network

Overview

This section discusses scanning for vulnerable RTUs (remote terminal units) and the detection of scanning activity within the SCADA network. This activity is part of the end-to-end threat scenario, depicted in the purple box in Figure **1**.

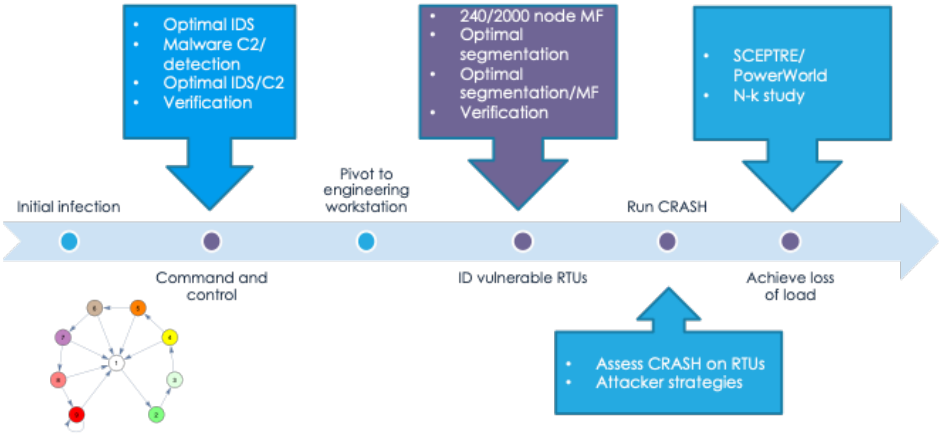


Figure **1**: SECURE end-to-end threat scenario, with SCADA network studies highlighted

In this scenario, when the attacker lands on an engineering workstation in the power grid control center, it doesn't know the IP addresses of RTUs that are vulnerable to the CRASHOVERRIDE malware, so it must scan for them. However, as the attacker is scanning, the defender is monitoring SCADA network traffic and examining it using an intrusion detection system (IDS). One method used by IDS to detect scanning activity is to look for network packets that might indicate such activity, and when these packets are received with an intensity above a certain threshold, the IDS signals an alert. This detection approach guides an attacker's strategy: it can attempt to run slowly "below the radar" of IDS detection (at the expense of launching its attack later), or it can run quickly (at a higher risk of detection).

The following sections describe the mathematical modeling, the ns-3 simulation, and the emulation-based experimentation that were applied to model this step in the attack timeline.

Scenario

The scenario addressed in the emulation, simulation, and mathematical models assumes the attacker uses Nmap to scan for vulnerable RTUs, and the defender uses Snort (with the sfportscan module) to detect scanning activity. Both tools were selected for these models because they are commonly used, open source, and familiar to the experimental team. In particular, the fact that these tools are open source means that the experimental team can better understand how these tools

work "under the hood," which is especially important when developing simulation and mathematical models. However, it's important to emphasize that, although these specific tools were selected for the studies, the methodologies (and, in some cases, the results) are generalizable to other scanning and IDS tools.

Topology

The topology studied in the emulation, mathematical, and simulation models is shown in the following Figure 2.

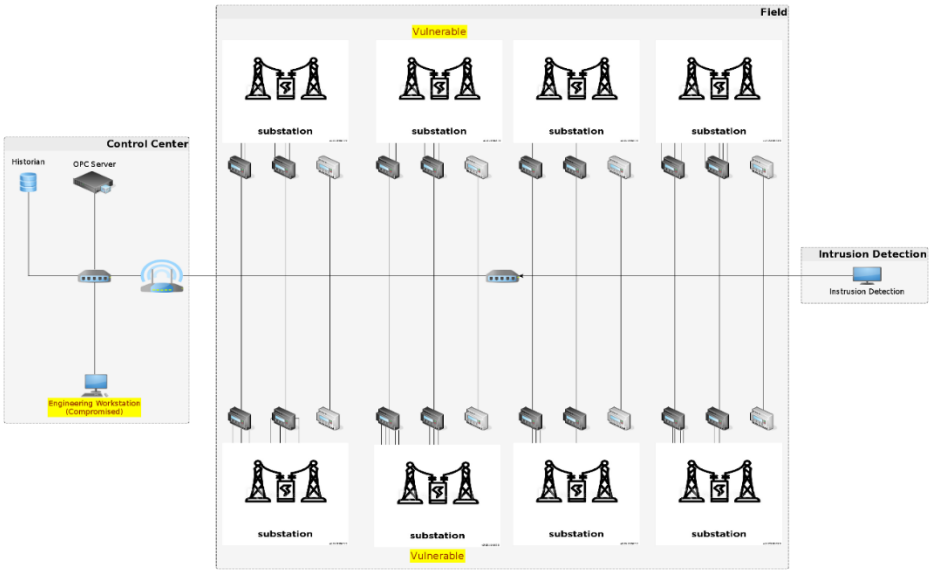


Figure **2**: Notional SCADA network topology for scanning/detection study

This topology (which does not reflect a particular real-world SCADA/ICS network, but is meant to be representative) consists of the following components:

- An engineering workstation in a control center network that represents the attacker's current location, from which it scans the SCADA network for vulnerable devices;
- A router that separates the control center IP subnet from the SCADA network IP subnet;
- An IDS that listens to all traffic on the SCADA network IP subnet;
- 8 SCADA substations, all on the same IP subnet; and
- 24 hosts, distributed across the SCADA substations, configured as follows:
    - 4 hosts are vulnerable to CRASHOVERRIDE,
    - 8 hosts are not vulnerable, but are discoverable,
    - 12 hosts are neither vulnerable nor discoverable.

Nmap

As described earlier, in our modeled scenarios we configure the attacker node to use Nmap to scan for and find vulnerable nodes. Nmap performs its scan using the Transmission Control Protocol (TCP) connection establishment protocol to look for active IP addresses with open ports, as shown in the following Figure 3:

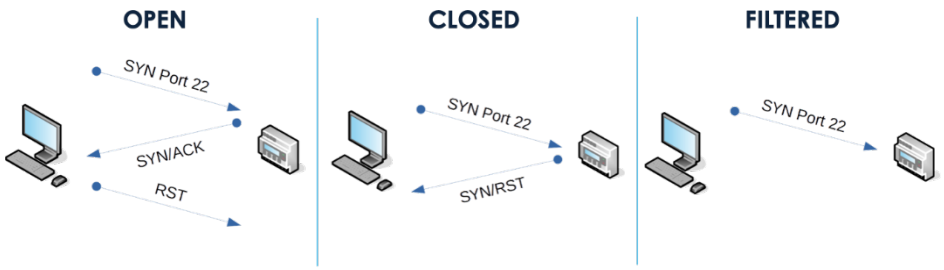


Figure **3**: Nmap protocol operations while scanning open, closed, and filtered hosts

In our scenarios, we model "vulnerable" hosts (see previous section) as hosts that have a particular port in the "open" state, which represents a vulnerable application. When Nmap scans a host on an active IP address with an open port (i.e. an application listening on that port), Nmap sends a TCP SYN (synchronization) packet to that host IP/port combination, and the host responds with a SYN/ACK (acknowledgement). Normally the initiator would acknowledge the connection with a third message, ACK; however, Nmap does not want to maintain an open connection, so it responds with an RST (reset). If Nmap receives a SYN/ACK from a remote host, then it knows two things: that the IP address is valid, and that an application is listening on that port.

Our scenarios model non-vulnerable but discoverable hosts as hosts that have that particular port in the "closed" state (meaning that these hosts are not running the vulnerable application). When Nmap scans a host on an active IP address with a closed port, Nmap sends a TCP SYN packet to the host IP/port combination, and the host responds with a SNY/RST message. Therefore, if Nmap receives a SYN/RST from a remote host, it knows that the IP address is valid, but there is no application listening on that port.

Hosts that are neither vulnerable nor discoverable are modeled as hosts with the IP address/port combination that are "filtered." In this case, when Nmap sends a TCP SYN message to these hosts, there is no reply back to the Nmap host, meaning that the host either does not exist or chooses not to reply.

Intrusion detection systems (IDS) will observe these connection request/response packets and use them to determine whether a scanning attack is occurring, as described in the next section. To counter IDS, Nmap has a couple of command line configurations that can be used. To reduce the scanning traffic intensity, Nmap allows the user to increase the delay between scanning probes (the “delay” parameter) and decrease the number of hosts that are probed in each attempt (the “host group” parameter). By default, Nmap scans hosts in sequence by IP address; however, that approach could tip off an IDS, so Nmap has a command line parameter to randomize the sequence in which the hosts’ IPs are probed.

In our studies, we varied parameters related to attacker strategy (i.e. “fast” vs. “slow”) and randomness (i.e. “sequential” vs. “random”). In addition, we also configured our experiments to allow random packet drop (i.e. “no drop” vs. “drop”), to determine the effect of imperfect packet transfers on results. The combination of the randomness order and random packet drop parameters are organized into two formulations: a *deterministic* formulation (i.e. sequential ordering, no packet drop) and a *stochastic* formulation (i.e. random ordering, random packet drop). The plots shown later in this section show results from both formulations.

Detection

Our scenarios assume intrusion detection using Snort [2]. Snort is a very flexible IDS framework that uses signature definition files and rules to identify traffic as malicious. In this example we use the “sfportscan” rule to detect Nmap scanning traffic using the technique identified in the previous section. As shown in Figure **4**, the sfportscan rule looks for SYN/RST traffic from "closed" (i.e. non-vulnerable, but discoverable) hosts, which is indicative of a scanning attack. If Snort/sfportscan counts five or more SYN/RST packets within a 60 second window, then it generates an alert. Our models and scenarios consider two attacker strategies: a "fast" strategy where the attacker attempts to discover as many vulnerable nodes as quickly as possible, and a "slow" strategy where the attacker attempts to stay within the 5 SYN/RST packets within a 60 second threshold. The results shown later in this section account for both strategies.

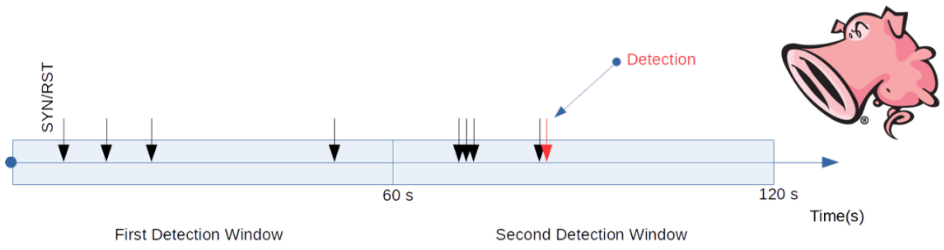


Figure **4**: Snort “sfportscan” rule

Tools

Mathematical model

We developed a mathematical model to assess the port discovery process. The model describes the stochastic state transitions that occur within the Nmap protocol that occur over time during the scanning process. This mathematical model is described in detail in [1] and summarized below and in Figure **5**:

1. The model states (illustrated in Figure **5**) are defined by the progress that Nmap makes scanning the nodes. The initial state at time 0 (indicated in the state on the far left of Figure **5**) contains key model parameters provided to the model. Each state consists of three lists that track the nodes that have yet to be scanned (topmost list in the state figure), the nodes that are being actively scanned (middle list), and the nodes that have already been scanned (bottom list). Furthermore, the color of the dots in the lists indicates the scanned nodes' status - magenta for filtered (inconclusive), green for closed (secure),

and red for open (vulnerable). All nodes begin in the first "To Scan" list in the initial state.

2. The model describes the transition from the initial state to subsequent states (in the second column in Figure **5**). The transition probabilities Pr{# filtered, # closed, # open} are determined by the number and type of nodes that have yet to be scanned and the probability that combinations of nodes are selected for scanning.
3. The third step the model consists of a third set of states (third column) that describe which nodes have been discovered (i.e. TCP SYN/RSTs occurred) and which ones timed out. The transition probabilities are conditioned on the current (second) state and depend on which nodes have been discovered so far. That is, the transition probability is Pr{# filtered_to_scan, # closed_to_scan, # open_to_scan | # filtered, # closed, # open}
4. If timeouts occurred, steps 2 and 3 are repeated.
5. Steps 2-4 are repeated until all nodes are moved to the Scanned list.

The steps in the model are implemented to effectively create a probability tree that lists the probability of discovering open, closed, and filtered nodes at each time step.

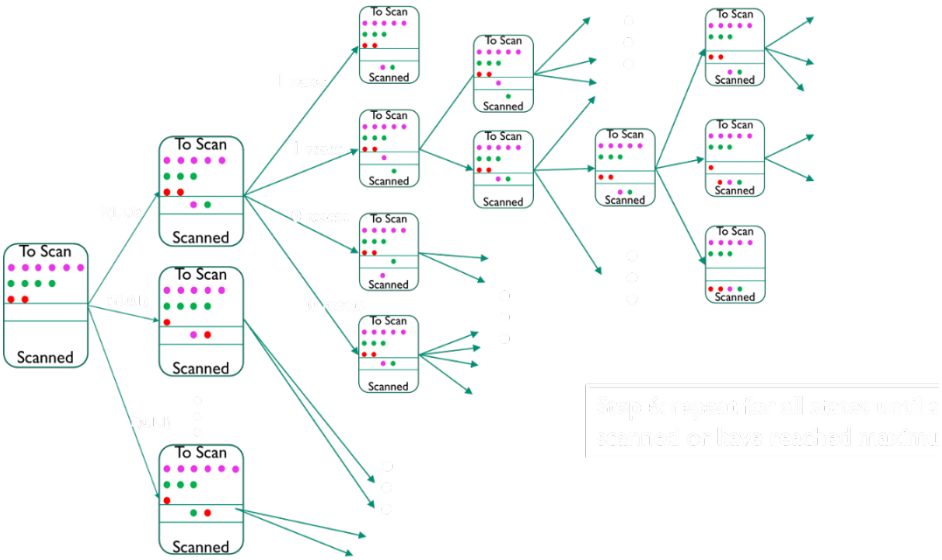


Figure **5**: Mathematical state transition diagram

We use the model results to compute the statistics of port discovery. Figure **6** shows the open port discovery process. The magenta stars represent the mean number of open ports discovered, as calculated with the math model. The blue line represents the mean number of open ports discovered from 1000 runs of the `minimega` emulation model, and the dashed black lines represent the 95% confidence intervals on the emulation means. The plot shows the mathematical results tracking the mean of the `minimega` runs and falling within the 95% confidence interval of these runs. This agreement validates the predictive value of the mathematical model, which, for small topologies, can run more quickly than the emulation model, making it more suitable for more widely evaluating the effect of configuration parameters (e.g. host group size and delay) on the results.

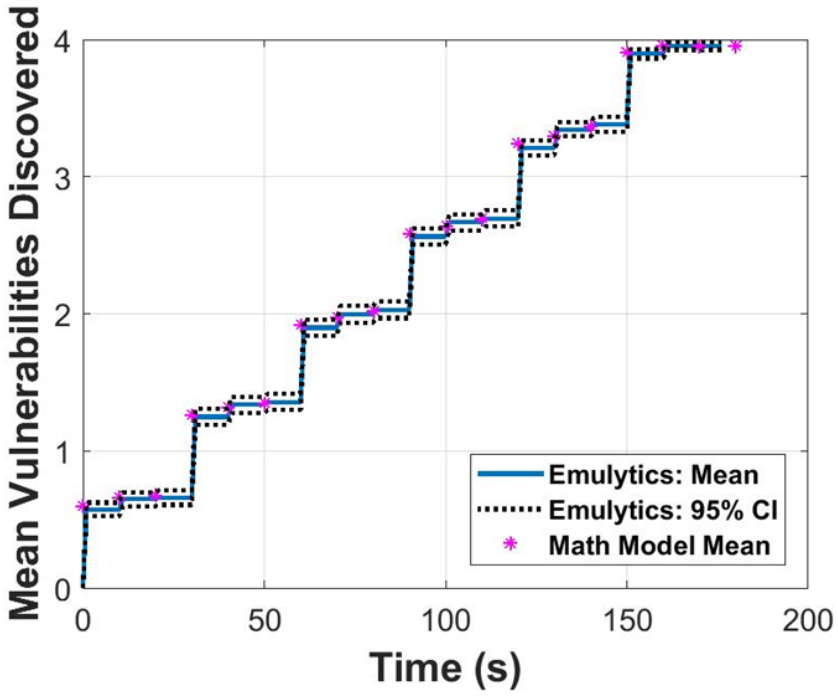


Figure **6**: Port discovery analysis (mathematical model and `minimega` emulation)

The model results were also processed to determine when and if detection would have occurred using the logic in the Snort sfportscan algorithm. These times were compared against the detection times that were experimentally determined using the `minimega` topology. The mathematical results, shown in Figure **7**, also closely track the results from the emulation runs and, again, validate the mathematical model's predictive ability.

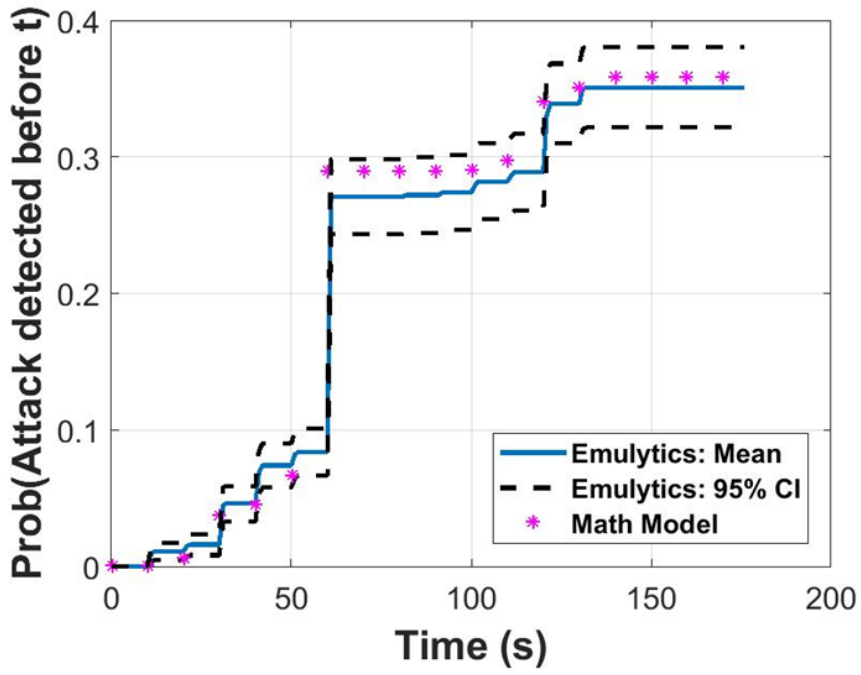


Figure **7**: Detection times

ns-3

Ns-3 [3] is a discrete event simulator that is used for network simulation and has an extensive model library for various network links, devices, and applications. Because it is a simulation, the components are abstracted objects and it does not run real implementations of applications and protocols. However, ns-3 simulations can run much more quickly when compared to emulations because discrete event simulations are event-driven rather than time-driven and can run faster than real time. This makes an ns-3 simulation particularly useful for serving as the low fidelity model in multi-fidelity modeling studies because it is much more efficient, and if implemented correctly, well correlated with emulation runs.

The Nmap ns-3 model developed in this work implements two major components - a topology and an Nmap application simulation model. The topology, shown in Figure **8**, corresponds to the SCADA network topology described earlier, but is different from the emulation model topology in a couple of ways:

- The ns-3 simulation topology has each SCADA device on its own subnet:
  This design choice is an artifact of how the example ns-3 star topology code does subnetting, and should not appreciably affect packet timings and results. Nevertheless, it does affect scalability of the topology because the subnetting schemed used in the model only allows up to 255 subnets (and with one host per subnet, 255 hosts).
- Different mechanisms are used to implement closed and filtered nodes:
  Whereas the emulation uses iptables filtering to implement closed and filtered nodes, the

ns-3 model does not install a packet sink on closed nodes, and causes Nmap to scan unused IP addresses for filtered nodes.

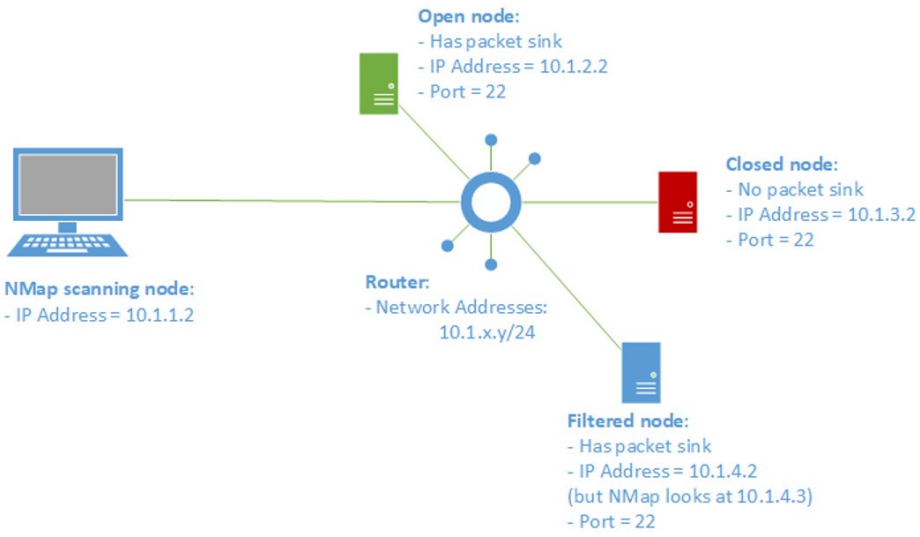


Figure **8**: ns-3 model for scanning/detection

The Nmap application running on the scanning node functions similarly to the real Nmap application running in the emulation. Also, the ns-3 model implements packet dropping using a similar mechanism that is used in the emulation model.

Emulation experiments using Scorch

The name SCORCH comes from the terms SCenario ORCHestration. It is primarily an automated scenario orchestration framework for emulation-based models, where a scenario is a specification of high-level experimental behaviors for a given experimental goal. Concretely, SCORCH is implemented as a python package that interfaces with `minimega` to run experimental scenarios on and collect data from emulation-based models (EBMs) managed by `minimega`.

At a high-level, basic SCORCH usage is as follows. First, a *scenario configuration file* is created that defines a scenario (experimental behaviors), model parameters, and output parsing. This file describes the "what" of the experimental scenario. The scenario is defined in terms of modular scenario *components* which represent re-usable experiment primitives. The code implementing components describes the "how" of the experimental scenario.

Secondly, a `minimega` topology is deployed on a hardware cluster (or single machine). This is the EBM to which the experimental scenario will be applied. This step highlights a degree of separation between structure and function of the experiment. The `minimega` topology represents the structure of the experiment while the SCORCH scenario represents the function. This separation enables efficiency in experimentation by, for example, enabling the user to apply the same scenario to a variety of topologies without the need to re-create the scenario for each

topology, or enabling the user to apply a variety of scenarios to the same topology without having to tear down the topology.

In this study, the SCADA network topology is deployed within `minimega` where each virtual machine (VM) receives the necessary software and model parameters to execute the scanning/detection scenario. For example, the scanning VM includes Nmap and a list of parameters such as: number of IP addresses and ports to scan, specific port number to scan, time to wait between scans (delay), etc. This set is subsequently used to scan the SCADA network. Each time a port is scanned, the metadata associated with the scan is logged to an Nmap.out file. To counter the adversarial scanning VM, the detector VM runs snort and its configuration parameters capable of sensing the syn packets used in Nmap probing. If snort notices a packet that aligns with criteria in one of its rules, it will signal an alert and append all such to an alert file. During this reciprocal exchange, tcpdump captures all traffic on the network by way of a port mirror residing on the `minimega` virtual LAN hosting the SCADA network. This data is saved as a PCAP file.

Data collection

Input/output to and from the live virtual network is handled by the individual components as facilitated by the framework. Here, the `minimega` command and control agent, miniccc, handles the data input and output process, in tandem with the snort, tcdump, and filebeat components. During SCORCH execution, the Nmap and snort components call miniccc to signal that their respective model parameters and other supporting data, be added to the model. This occurs during EBM setup, where miniccc copies the data from the hardware cluster node to the respective VM within the `minimega` topology. After the experiment has completed, each component initiates an exfil process where it again calls the miniccc agent to extract any logging data accrued by Nmap, snort or tcpdump. This data is then written to the host cluster node for analysis. If enabled, SCORCH interfaces with Filebeat to push the collected experimental data and artifacts to a specified Elasticsearch server.

Following data collection, post processing scripts run against the PCAP and snort alert files to derive the time delta (in seconds) between the 1st packet captured and the 1st alert instance captured, for every Nmap portsweep occurrence. If any time format discrepancies exist between the PCAP and alert file, the scripts will convert the packet time to reflect seconds since Unix epoch time (Jan 1, 1970). Once the initial alert time values have been calculated, the post processing scripts aggregate the initial alerts times for every experiment and log to a metrics.txt file. This is done for each snort sensitivity level (low, medium, high).

Experimental methods

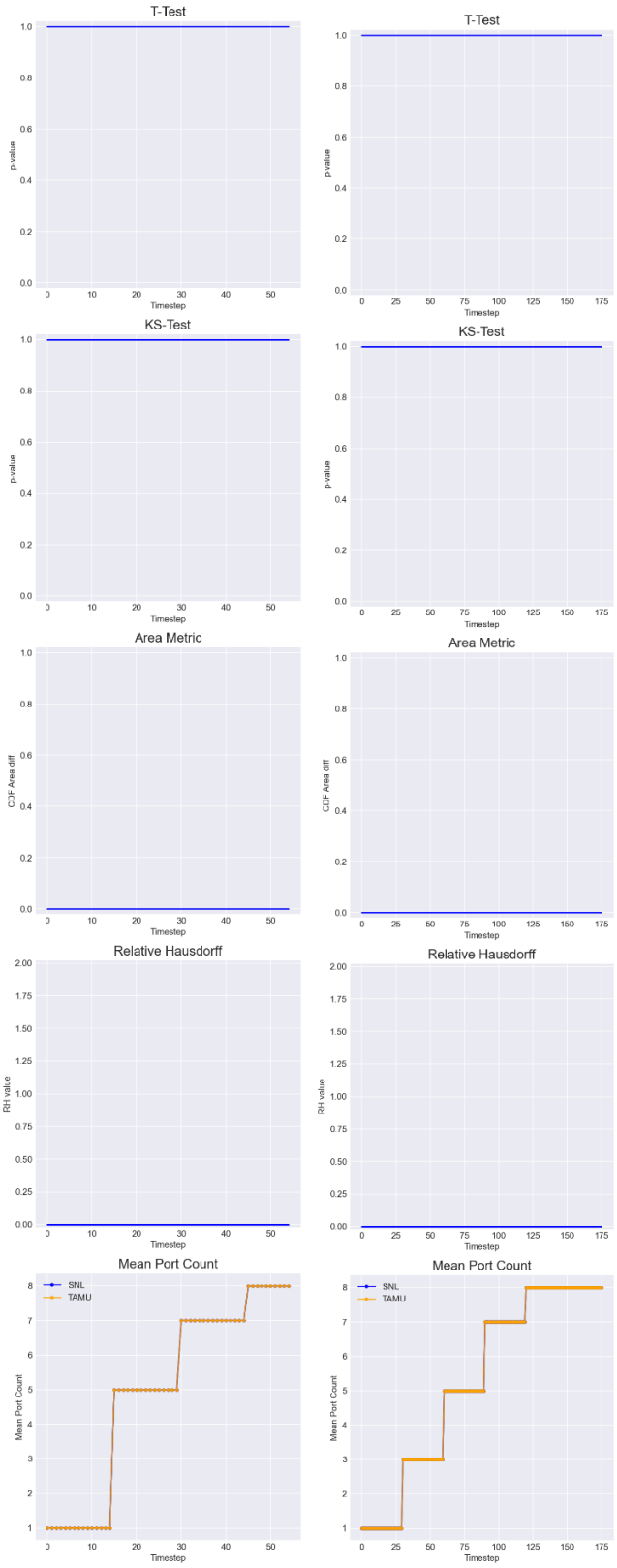
Experiment reproduction

Reproducibility is essential to science because it ensures results are not biased according to overt or hidden desires for a particular outcome. The SECURE team, working with our collaborators from Texas A&M University (TAMU), wanted to see understand the degree to which the results published in [1] can be reproduce by a research team that did not contribute to the original paper.

In the process of reproducing this study (which is described in detail in [4], the team not only considered the methods for reproducing the results, but also the metrics by which the results from Sandia and TAMU should be compared. The comparison metrics used during this study were:

- t-test: the t-test is a widely-used test for determining if there is a statistically significant difference between the means of two data sets,
- Kolmogorov-Smirnov Test: the KS-test is a non-parametric statistical test for equality of distributions, based on the maximum difference between the cumulative distribution functions (CDFs),
- Area Test: the area test also compares CDFs, but accounts for the entire difference between CDFs rather than the maximum difference, and
- Relative Hausdorff Distance: originally developed for graph analysis, the Relative Hausdorff Distance can also be used to compare distributions

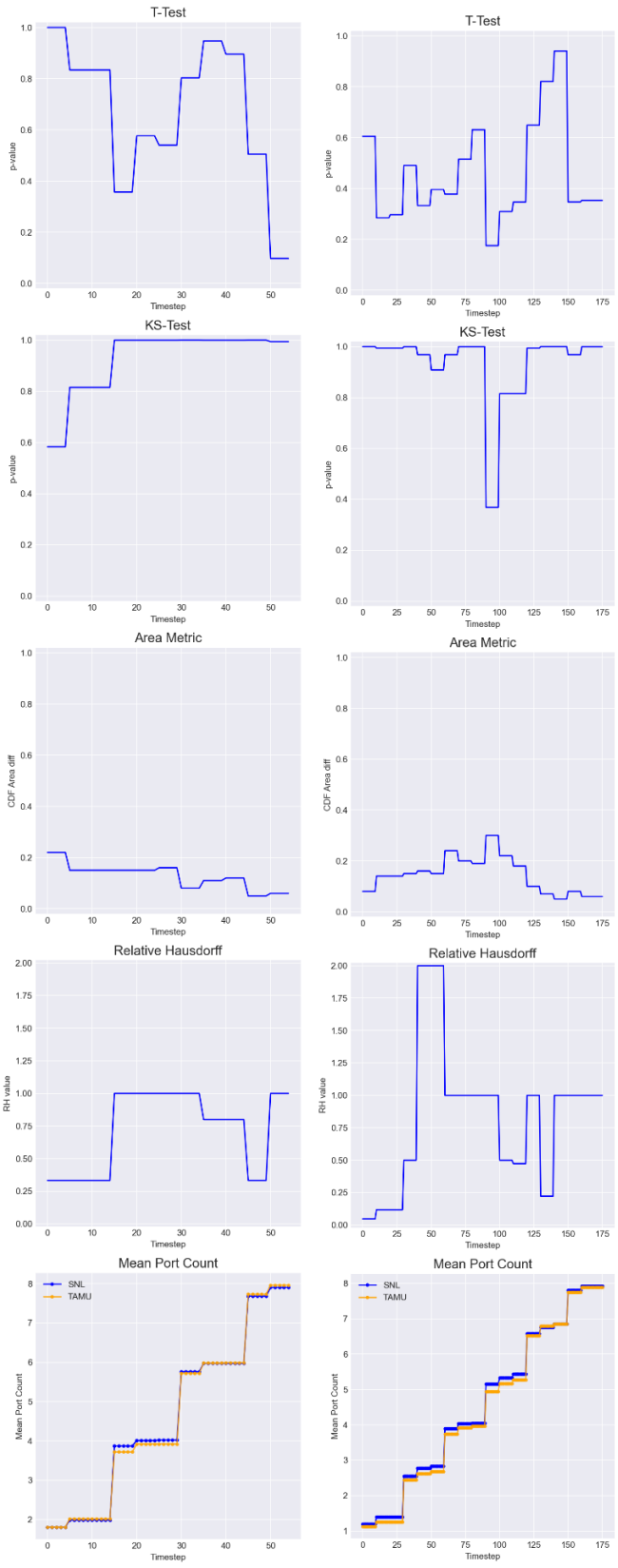
The plots in Figure 9 show the application of these metrics to compare Sandia and TAMU port discovery results in the case where there is no added randomness (i.e. deterministic formulation):



Fast, deterministic Slow, deterministic

Figure **9**. Port Discovery Statistical Test Results for Deterministic Case

The results above show perfect agreement between the Sandia and TAMU results, as evidenced by all four metrics, indicating that TAMU correctly set up the experiment for the deterministic formulation. The plots in Figure 10 show the application of these metrics to compare Sandia and TAMU port discovery results in the case where there is there is added randomness in the Nmap search order and in packet loss (i.e. stochastic formulation):



Fast, stochastic Slow, stochastic

Figure **10**. Port Discovery Statistical Test Results for Deterministic Case

From these comparisons we find that the KS Test shows good agreement between the Sandia and TAMU results, as evidenced by the p values > 0.05. The Area Metrics for all cases also show good agreement as evidenced by the consistently low area values. However, we find that the Relative Hausdorff metric does not seem to be a suitable metric for comparing results, as seen in the plots above.

Verification

An important part of using emulation is verifying whether the emulation environment is working as intended, also called verification [5]. Part of verification involves software testing and quality assurance. A unique aspect of cyber emulation involves assessing the performance of the emulation running in the virtualized environment and determining whether there are sufficient resources to properly handle the scenario that is being run. If there are not, the virtualized components may produce experimental artifacts and behavior that result in the experimental outcomes being unrepresentative or incorrect.

Under SECURE, we focused on determining whether there are sufficient virtualized resources to support the emulation experiment and whether we can identify metrics that indicate when the results of an emulation experiment are unreliable. We refer to these metrics as telemetry metrics, following the usage of this phrase from Microsoft [6], Google [7], Intel [8] and Sumo Logic [9]. We studied telemetry metrics such as system load and CPU utilization relating to the performance of virtual machines which are used in the scanning/detection scenario and the physical machine hosting that study. We ran experiments with various levels of over-subscribed resources.

In these experiments, we purposefully put more and more strain on the physical resources available to the emulation experiments. We accomplished this by forcing the physical host to do more and more work in parallel through the concept of a namespace, which is an isolated copy of the experiment environment running on its own VLAN. For the purposes of this study, we ran several iterations of the same experiment with increasing numbers of parallel namespaces. By increasing the number of namespaces, we hoped to reach a point of resource over-subscription, where the results of the experiments run are affected by emulation artifacts caused by this over-subscription. We saw evidence of oversubscription at 20 namespaces and greater.

We found that statistical tests such as the Tukey multiple mean comparison test was useful to identify anomalies in results as we increased the number of parallel namespaces running in the experiments. For scanning/detection, as we increased namespaces, we found that the alert time distributions shifted upward and became much more diffuse with longer tails. We also found that the telemetry metrics of system load and throughput were effective at filtering out replicates which had statistically significantly different results than the baseline case with one namespace.

Validation

Validation is the process of verifying that the model is correct with respect to the questions that it is intended to answer. Validation can be done in several ways; it can be performed on multiple models and compared (i.e. cross-validation), and validation experiments can be conducted in

physical testbeds and compared with models. In the SECURE project we performed two different kinds of physical experiments and compared the results with the `minimega` scanning/detection model:

1. Validation experiments on physical hosts in the Sandia computing cluster. The physical hosts used for these experiments were all identical, but configured differently to assume different roles in the validation experiment, and
2. Validation experiments using physical and virtual devices in the Texas A&M testbed. Physical relay devices were used to model vulnerable hosts (open ports) in the scanning detection scenario, however, due to limited numbers of physical devices, closed ports were modeled using the CORE virtual machine testbed, and filtered devices were modeled using firewall rules in the network switch.

The Sandia physical validation experiments utilized the same software (applications and operating systems) that was used in the `minimega` virtual machine-based experiments. The primary differences between the `minimega` and physical experiments were 1) `minimega` used KVM-based virtual machines whereas the physical experiments were run on physical hosts, and 2) a few configuration differences due to differences in networking between the virtual and physical experiments. We found the port scanning and Snort detection time results between the `minimega` and physical experiments matched up very well.

The Texas A&M University (TAMU) physical testbed experiments used a mixture of physical and virtual hosts in order to achieve the scales that were needed to conduct the validation experiment. The TAMU team used four field devices to implement vulnerable hosts with open ports, eight virtual machines running in the CORE virtual testbed environment to represent secure hosts with closed ports, and used firewall rules in the network switch to represent 12 secure hosts that are filtering inbound TCP connection requests. The TAMU physical testbed configuration used the same scanning and detection software used in the `minimega` experiment, however, because the TAMU testbed was very different from the `minimega` testbed, a number of custom scripts were written to orchestrate the experiment and collect data. These scripts required some amount of debugging, resulting in some back-and-forth between the Sandia and TAMU teams to make sure the physical experiment was producing correct validation data. Due to limitations in available time, the two teams were able to validate port discovery but did not have an opportunity to assess validation with respect to detection times.

Optimal segmentation

Network segmentation is a strategy used by the network designer to limit the scope of what an attacker may see if they are able to achieve a malware presence on the network. However, network segmentation has costs and constraints on the network design - too much segmentation will incur excessive costs and exceed the defender's budget. Therefore, a tri-level optimization formulation was developed to account for 1) network designer's budget, 2) attacker's budget (in terms of the number of networks that the attacker can compromise), and 3) the network operator's response to an attack (e.g. re-dispatching generation resources to loads). This optimization model and results are documented in [10].

References for Scanning/Detection

1. Vugrin, Eric D., Jerry Cruz, Christian Reedy, Thomas Tarman, and Ali Pinar. "Cyber Threat Modeling and Validation: Port Scanning and Detection." *Proceedings of the 7th Symposium on Hot Topics in the Science of Security*, Lawrence, Kansas, Association for Computing Machinery, 2020. https://doi.org/10.1145/3384217.3385626
2. https://www.snort.org
3. https://www.nsnam.org
4. CSET: T. D. Tarman, T. Rollins, L.P. Swiler, J. Cruz, E. Vugrin, H. Huang, A. Sahu, P. Wlazlo, A. Goulart, and K. Davis. Comparing reproduced cyber experimentation studies across different emulation testbeds. USENIX 14th *Cyber Security Experimentation and Test (CSET) Workshop.* Aug. 9, 2021. SAND2021-5696C.
5. Oberkampf, W.L. and C.J. Roy. Verification and Validation in Scientific Computing. Cambridge University Press, 2010.
6. https://docs.microsoft.com/en-us/azure/azure-monitor/autoscale/autoscale-common-metrics
7. https://cloud.google.com/network-telemetry
8. https://www.intel.com/content/www/us/en/cloud-computing/telemetry.html
9. https://www.sumologic.com/insight/what-is-telemetry/
10. B Arguello and E.S. Johnson and J.L. Gearhart, "A Trilevel Model for Segmentation of the Power Transmission Grid Cyber Network", arXiv.2108.10958: https://arxiv.org/abs/2108.10958. SAND2021-10208O

Power Grid Impacts

Overview

This section demonstrates how the methods developed under SECURE can be used to analyze the power grid impacts of the larger attack chain. Recall that the full end-to-end exemplar considered under SECURE describes a multi-stage attack in which an attacker attempts to access a power utility's corporate enterprise network, pivot to the ICS network, identify vulnerable RTUs, run the CRASHOVERRIDE malware and ultimately disrupt operations by causing load shed. The focus of this article is the power grid impacts caused by the CRASHOVERRIDE malware.

CRASHOVERRIDE

CRASHOVERRIDE was malware designed to attack power grids and was used in the 2016 cyber attack on the Ukrainian electric grid. Unlike the previous attack on the Ukrainian grid in 2015 in which attackers manually switched off power to electrical substations, the CRASHOVERRIDE attack was fully automated and could perform attacks much more quickly and with less preparation. Once the malware had infected the system, CRASHOVERRIDE could launch four payload modules. This study focuses on the module that communicates directly with grid equipment and switches breakers within the power grid. https://www.dragos.com/resource/crashoverride-analyzing-the-malware-that-attacks-power-grids/

In power systems, field devices (such as relays, RTUs and PLCs) monitor and control the power grid. CRASHOVERRIDE understands how to enumerate and discover the inputs and outputs to field devices and leverages this to open circuit breakers in the power system. Additionally, CRASHOVERRIDE can force the field devices into an infinite loop thus continually opening the circuit breakers even if operators are dispatched to re-close them.

In our multi-stage attack, Nmap is used to scan the network for vulnerable RTUs. CRASHOVERRIDE will then target only those RTUs and open the breakers associated with those RTUs. The power grid impacts of this CRASHOVERRIDE attack will highly depend on the identification of vulnerable RTUs.

CRASHOVERRIDE Configuration

CRASHOVERRIDE modules were designed to be used with configuration files specifying various parameters of the attack. This section focuses on the configuration associated with the module that targets the protocol payload. In this configuration, a set of stations are specified for an attack. Each targeted station has the following configuration options:

- target ip - specifies the IP address of the targeted field device
- first action - specifies the first action (on or off) used to switch grid components

- change - specifies whether to continually toggle power grid equipment (1) or only change once (0)
- interval - specifies the time interval in between toggles

TAMU Topology

Power grid impact experiments were all performed on a synthetic cyber-physical topology of the Texas power grid developed by Texas A&M University's (TAMU) Cyber Physical Resilient Energy Systems (CyPRES) project. https://cypres.engr.tamu.edu/test-cases/

This topology consists of both cyber and physical components. The cyber model shown in Figure 1 has three main sets of components: (1) balancing authorities, (2) utility control centers, and (3) substations. The primary and secondary balancing authorities are responsible for managing the flow of electric power among the utilities. The utility control centers are responsible for monitoring multiple substations and contain networking equipment, a demilitarized zone, and SCADA software. The substations are responsible for monitoring and controlling the power grid and contain networking equipment, relays, as well as corporate devices such as PCs, security cameras, phones, and card readers. The relays in each substation are mapped to busses and branches of a synthetic 2000-bus power model of the Texas grid shown in Figure 2. Overall, this topology contains 2 balancing authorities, 150 utility control centers, and 1251 substations.

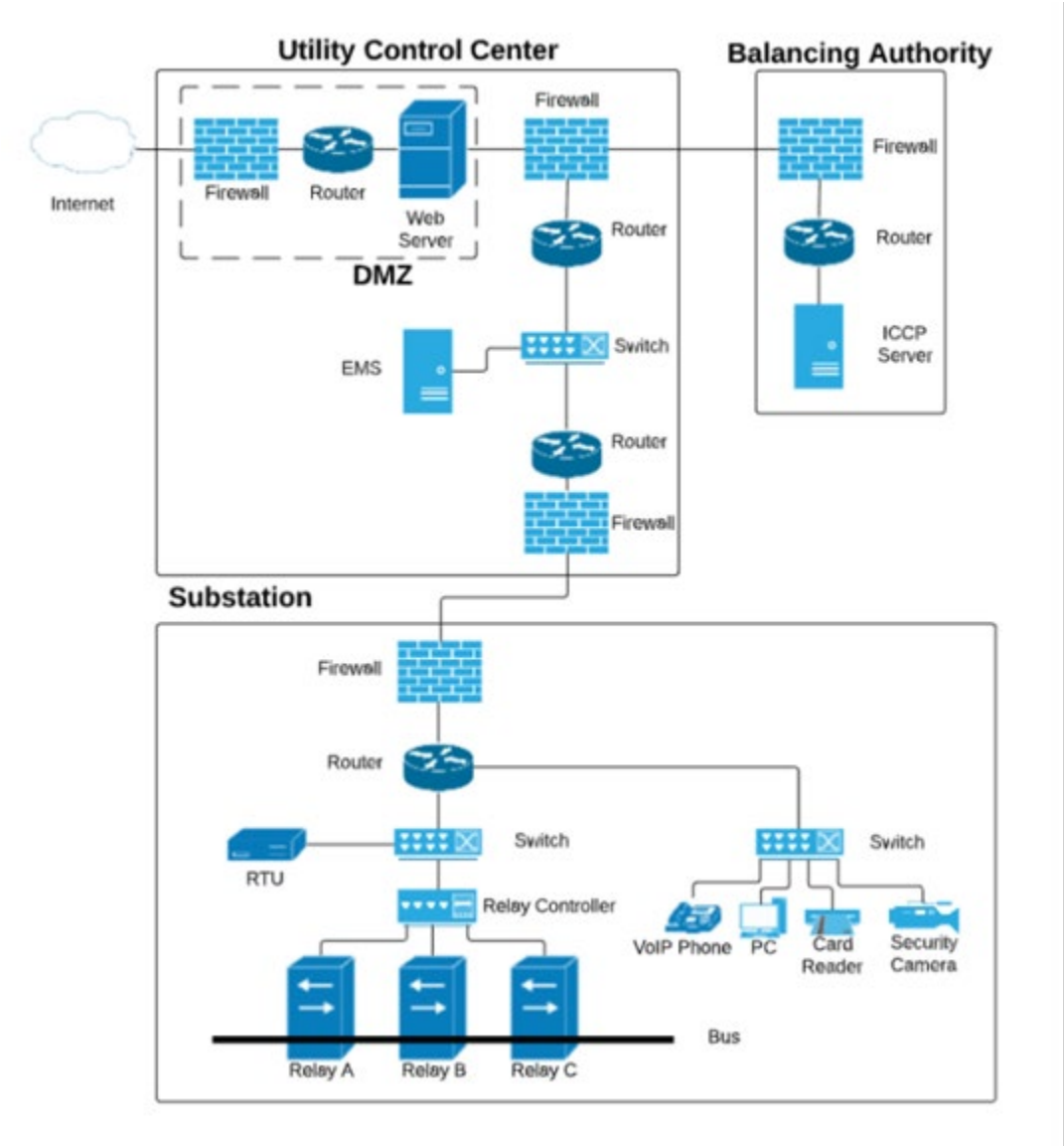


Figure 1: TAMU cyber topology

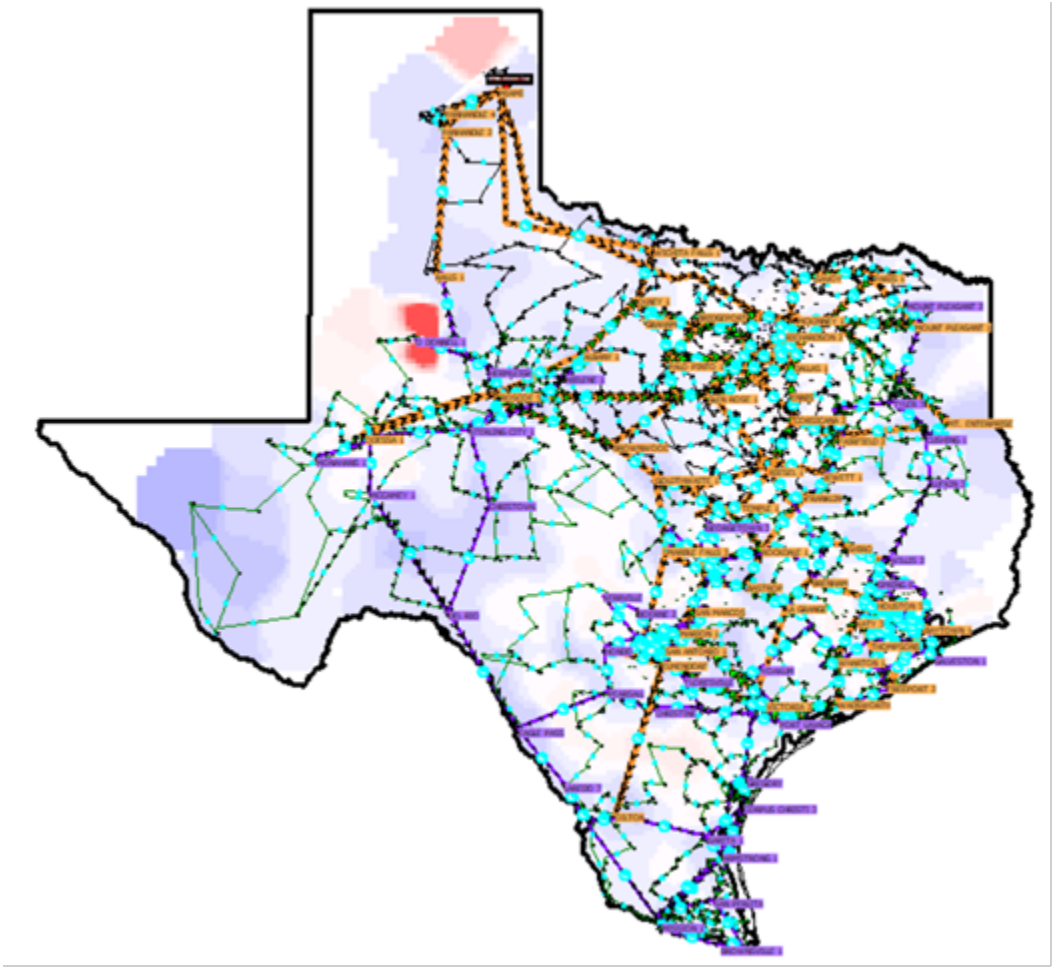


Figure 2: 2000-bus power model

Power Grid Impact Studies

The CRASHOVERRIDE malware and the TAMU topology were used for two main studies: an uncertainty quantification study and an optimal segmentation study.

UQ Study

A workflow was developed for the UQ study that leverages both traditional UQ tools and emulation tools. Dakota provides a means to sample CrashOverride parameters and generates a CrashOverride configuration file. For each sample of parameters, Scorch then injects the new CrashOverride configuration file into the SCEPTRE experiment, runs the CrashOverride malware in SCEPTRE, collects physical process data from the power model, and then resets the SCEPTRE emulation. Dakota then chooses the next sample and the process repeats. The data is then post processed and can then be further analyzed.

The UQ study was performed on a small subset of the TAMU topology consisting of 1 balancing authority, 2 control centers, and 11 substations. All protections on relays in the topology were disabled so that the effects of CRASHOVERRIDE could be clearly identified. 800 experiments were run sampling the parameters in Table 1. The overall timing of each experiment was 150s; the first 30 seconds of each experiment was normal operations. CRASHOVERRIDE was executed at the 30s mark and was run for an additional 2 minutes. The physical process data was post-processed to calculate loss of load for each experiment.

Parameter	Values
target ip	set of 49 relay IPs
first action	[off]
change	[0, 1]
interval	[10, 11, 12, ..., 60]

Table 1 - Parameters of UQ Study 1

Figure 3 shows results of the UQ study. Each point on the plot shows the loss of load results for a single experiment. The red line shows the mean regression line. the green line shows the median regression line while the black lines show the regression lines for the 0.05, 0.1, 0.25, 0.75, 0.9, 0.95 quantiles respectively.

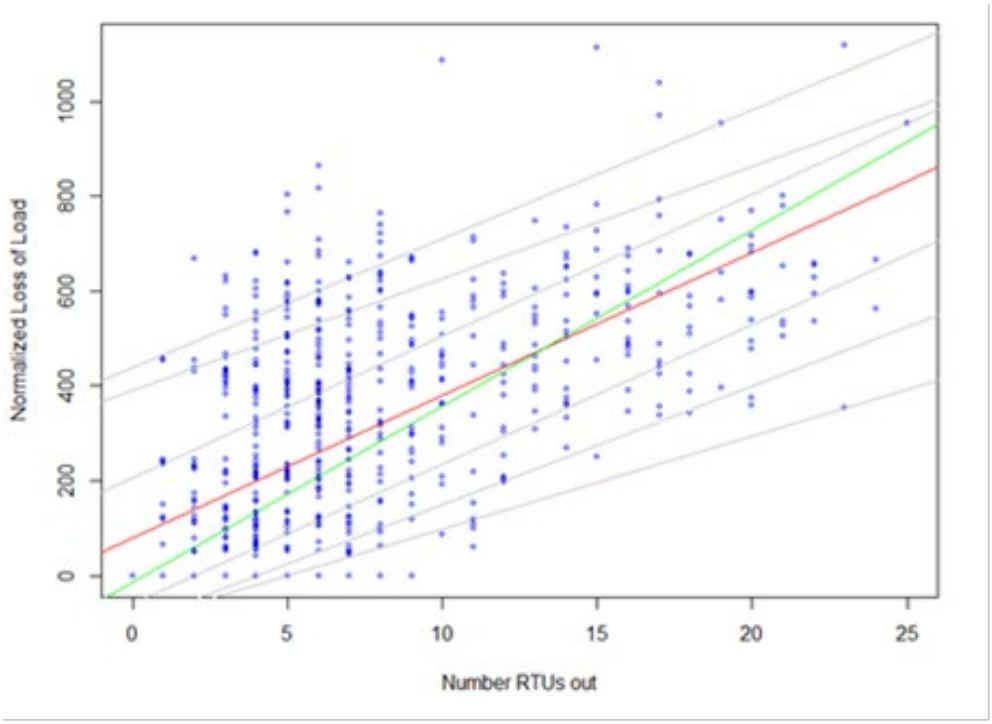


Figure 3: UQ Experiment Results with Quantile Lines for Normalized Loss of Load

For a given number of RTUs out (such as 4), there is a huge spread in the loss of load based on which four RTUs are targeted. This variance makes it hard to get a good regression model: the regression captures the mean trend but does not capture the variance well. If we instead look at the quantile regression lines, there is a better trend than with the mean regression line. Each quantile regression also gives us an analytic formula for a tail probability of normalized loss of load. For example, the 95th quantile = 440.18+27.10*RTUs_out. This formula can be used in end-to-end CRASH studies, where we want to couple upstream attack uncertainties to a tail probability loss of load (instead of worst case).

Future studies are planned, to increase complexity of the model by scaling the size of the topology as well as reimplementing the relay protections. However, due to the large variability of results present in the small topology, future work will first include more analysis of the current results such as worst-case analyses.

Segmentation Study

The second study using the TAMU topology and CRASHOVERRIDE malware was a segmentation study. The optimal segmentation work determined optimal segmentation of a network using mathematical optimization. This study applied the mathematical results to the TAMU topology and investigated the impacts the CRASHOVERRIDE malware would have on this new, segmented topology. We hypothesized that using the mathematical results would decrease the impact of the CRASHOVERRIDE malware since optimal segmentation would force the attacker to pivot more within the network to deliver the CRASHOVERRIDE payload to specific relays.

A workflow was designed that interfaces emulation with mathematical optimization for network segmentation. The workflow starts with an initial SCADA network implemented in SCEPTRE. The design of the topology (i.e. current network segments) is input to the mathematical optimization. The mathematical optimization then does two things. 1) identifies the worst-case attacker on the original topology and 2) identifies a new optimally segmented network topology along with the worst-case attack for this new topology. The SCEPTRE topology is then updated with the new segmented topology. Theoretically speaking, this is done by re-subnetting and applying new firewall rules. However, for our example, we wanted to investigate the effects of the CrashOverride malware against the optimal and non-optimal network topologies. So practically speaking, we investigated this by simply changing the potential targets of CrashOverride based on the segmentation that came from the optimization.

To gather results, the worst-case attacker (specific to each topology) was used to identify the set of RTUs that CrashOverride would target. The CrashOverride malware was implemented and for each topology, 100 experiments were run varying the other parameters of CrashOverride. Figure 4 shows results of this study. The results show that the optimal segmentation of the network lowered the cumulative loss of load for the scenario.

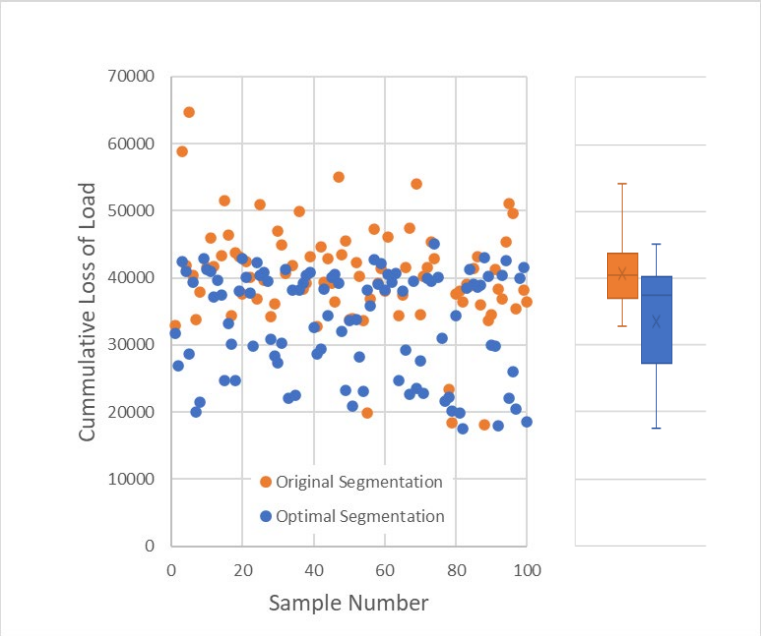


Figure 4: Segmentation Results

Moreover, this study shows the value of coupling mathematical optimization with emulation. Determining an optimal segmentation in emulation is usually SME driven and would require full enumeration or brute force to determine a true optimal solution. This is infeasible in emulation so practically heuristics would normally be used, but these do no guarantee an optimal or even near optimal solution. By coupling the emulation with mathematical optimization, most of the burden is done by the lightweight mathematical model.

Beyond this initial study, other questions we want to answer about this study are:

1. Can we use emulation to show that the mathematical result is better than SME design?
2. Can we use emulation to explore the robustness of the mathematical abstraction?
3. Does the incorporation of other real-world parameters (such as scanning and detection probabilities) affect the optimality of the segmentation?
4. What are the tradeoff costs between cost to implement segmentation versus benefit the segmentation provides against an attacker?

Markov Modeling of End-to-End Attack

The SECURE project used Markov analysis to assess attacker/defender performance relative to the end-to-end scenario, answering questions regarding an attacker's probability of successfully performing a power grid attack, and the time required for an attacker to traverse all of the steps required to reach this state. The process starts with translating the end-to-end scenario to a Markov state transition diagram, as shown in Figure 1.

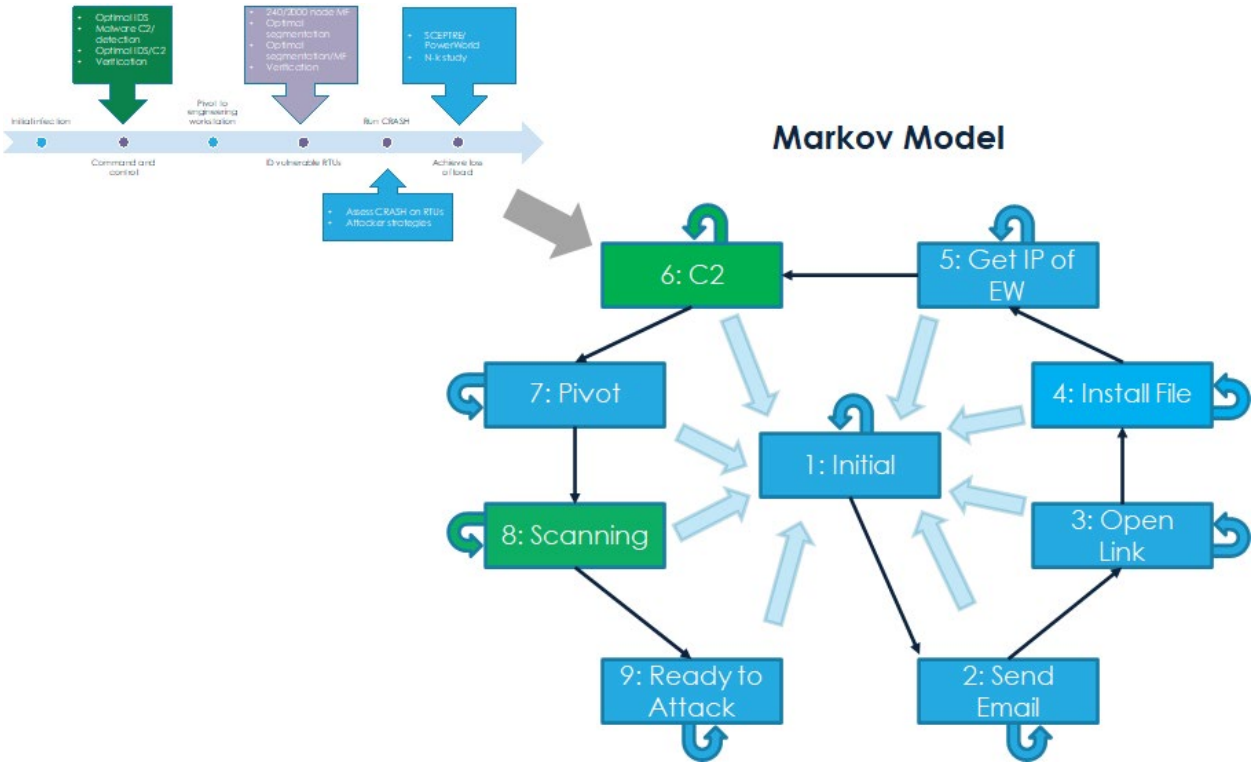


Figure 1: Translating the end-to-end threat scenario to a Markov state transition diagram

Once the state transition diagram is constructed, the task shifts to populating the model with transition probabilities. These transition probabilities can be determined via a number of means: through data collected from the MITRE ATT&CK evaluations, through subject matter expert (SME) judgment, or through cyber experimentation. In this study, we used cyber experimentation (i.e. emulation-based modeling and mathematical modeling) to calculate transition probabilities for both the "Command and control" (Markov state 6) and the "ID vulnerable RTUs through Scanning" (Markov state 8) steps highlighted in green in Figure 1. These transition probabilities are shown in Table 1 and Table 2.

Table 1. C2/Markov state 6 transition probabilities (from emulation and mathematical models)

Snort condition	Timestep value	Detection probability	Next state transition probability	Same state transition probability
Unstressed	16 s	0.565	0.435	0.0
Stressed (dropping packets)	16 s	0.372	0.628	0.0

Table 2. ID RTUs/Markov state 8 transition probabilities (from emulation and mathematical models)

Attacker scanning strategy	Timestep value	Detection probability	Next state transition probability	Same state transition probability
Fast	30 s	0.69	0.31	0.0
Slow	61 s	0.70	0.30	0.0

An example of an analysis using the experimental and MITRE ATT&CK transition probabilities is shown in Figure 2.

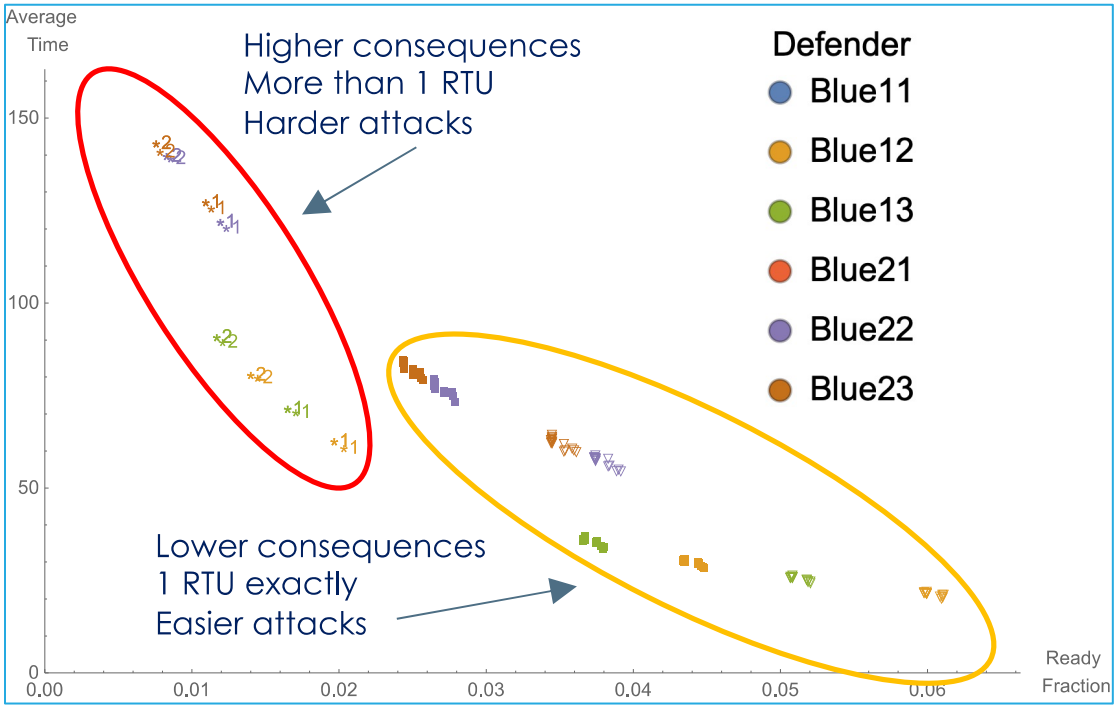


Figure 2: Markov analysis results showing attacker time to success and success probabilities, depending on defender capabilities

This analysis assumes a set of different defender (blue team) capabilities denoted B_{ij}, depending on the specific MITRE ATT&CK tactics employed by the attacker (denoted by subscript i), and the defender's ability to handle increasing levels of ambiguity in attack indications (denoted by Table 3).

Table 3. Defender capabilities

Defender name	Level of ambiguity	Detection capabilities
$\mathbf{B_{i,1}}$	None	Indicators of compromise (IOC)
$\mathbf{B_{i,2}}$	Medium	IOC, specific alerts
$\mathbf{B_{i,3}}$	High	IOC, specific alerts, general alerts

Figure 2 shows the mean time it takes an attacker to transition from state 1 (initial state) to state 9 (ready to attack state) in the Markov chain on the Y axis, and the steady state probability of the attacker residing in state 9 on the X axis. These results are collected into two sets, indicated by the ovals, with the yellow oval indicating results if the attacker only needs to discover exactly one RTU to proceed, and the red oval indicating results if the attacker needs to discover more than one RTU. In cases where the attacker must find more than one RTU in order to continue with its attack, the probability of success is lowest and the time to success is longest (as shown in the set surrounded by the red oval). This makes intuitive sense, since the criteria are more difficult than in the other set, where the attacker only needs to find one RTU.

Within each set there are two arcs: one arc (green and orange points) is for attacker i=1, and the other arc (dark red and purple points) is for attacker i=2. Recall that each of these attackers is distinguished by the particular MITRE ATT&CK tactics that the attacker employs. As can be seen in Figure 2, attacker i=1 appears to use tactics that do a better job of evading detection than attacker i=2. Details regarding the tactics used by both attackers can be found in [1].

Within each arc there are two groups. In one group, denoted by triangles and *1 markers, the intrusion detection system is stressed by the volume of network data, and is dropping packets as a result. In the other group, denoted by squares and *2 markers, the intrusion detection system is able to process every packet. As the results show, when the C2 intrusion detection system is stressed and dropping packets, the attacker's time to success decreases and its ready fraction increases, indicating that the attacker is more likely to achieve its object more quickly, which makes intuitive sense.

Within each group, the data points are classified according to defender capability. Orange and purple markers represent a defender of Medium capability ($B_{i,2}$), and green and dark red markers represent a defender of High capability ($B_{i,3}$). As Figure 2 shows, when the defender's capability increases from j=2 to j=3, the attacker's time to success increases and its ready fraction decreases, indicating that it becomes harder for the attacker to achieve its objectives, which also makes intuitive sense. It should be noted that defender j=1 is not shown in this figure because the results are off the scale of the plot at Ready Fraction = 1.0, meaning that the attacker is certain to succeed against defender j=1.

[1] Defender Policy Evaluation and Resource Allocation against MITRE ATT&CK Data and Evaluations. Alexander V. Outkin, Timothy Schulz, Thomas D. Tarman, Patricia V. Schulz, Ali Pinar. SAND2021-7713. https://arxiv.org/abs/2107.04075