

# Cyber Threat Modeling and Validation: Port Scanning and Detection

Eric D. Vugrin, Jerry Cruz, Christian Reedy, Thomas Tarman, Ali Pinar\*

{edvugri,gcruz,creedy,tdtarma,apinar}@sandia.gov

Sandia National Laboratories

Albuquerque, NM, USA, and Livermore, CA, USA

## ABSTRACT

Port scanning is a commonly applied technique in the discovery phase of cyber attacks. As such, defending against them has long been the subject of many research and modeling efforts. Though modeling efforts can search large parameter spaces to find effective defensive parameter settings, confidence in modeling results can be hampered by limited or omitted validation efforts.

In this paper, we introduce a novel, mathematical model that describes port scanning progress by an attacker and intrusion detection by a defender. The paper further describes a set of emulation experiments that we conducted with a virtual testbed and used to validate the model. Results are presented for two scanning strategies: a slow, stealthy approach and a fast, loud approach. Estimates from the model fall within 95% confidence intervals on the means estimated from the experiments. Consequently, the model's predictive capability provides confidence in its use for evaluation and development of defensive strategies against port scanning.

## CCS CONCEPTS

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

## KEYWORDS

cyber security, port scanning, intrusion detection, cyber experimentation, model validation

## ACM Reference Format:

Eric D. Vugrin, Jerry Cruz, Christian Reedy, Thomas Tarman, Ali Pinar. 2020. Cyber Threat Modeling and Validation: Port Scanning and Detection. In *Hot Topics in the Science of Security Symposium (HotSoS '20)*, April 7–8, 2020, Lawrence, KS, USA. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3384217.3385626>

\*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 work is supported by the Laboratory Directed Research and Development program of 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.

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the United States government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

HotSoS '20, April 7–8, 2020, Lawrence, KS, USA

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-7561-0/20/04...\$15.00

<https://doi.org/10.1145/3384217.3385626>

## 1 INTRODUCTION

The recent attacks on the electrical power grid in Ukraine [1] and the discovery of the Triton malware in petrochemical plants in 2017 [2] have heightened cyber security concerns for industrial control systems (ICS). Cyber security researchers are increasingly using modeling and simulation platforms to investigate ICS risks, proposed security solutions, vulnerabilities, and other issues. These platforms typically include four main components. The first component is a representation of the software, hardware, communications, and other elements that comprise the system of interest and how it operates. ICS often include a physical process (e.g., chemical manufacturing, power generation); as such, the second component describes a physical process that interacts with the first component. The third component generally entails security features such as firewalls, intrusion detection systems (IDSs), and other features that are intended to protect the system against attacks. The fourth model component is a representation of the cyber threat; that is, the attacker, malware, or other means by which an adversary aims to steal information, disrupt operations, or cause other harm to system operations.

Representing threat elements in the cyber models and experiments can present a significant challenge. As noted in MITRE's ATT&CK framework [26], cyber attacks generally consist of multiple stages, and many tactics and tools are available to attackers. Furthermore, attacks are designed and executed by humans, implying that threat representation may need to consider strategic elements. These challenges can make validation difficult for models including the threat representations.

Discovery is an important, early stage in a cyber attack [26]. During discovery, the attacker attempts to gain knowledge about the system and determine how to proceed with the attack. Attackers can choose from a variety of techniques, depending on the type of network, the information they are attempting to discover, and their goals and strategies. To counter against these activities, network administrators often employ IDSs that monitor network traffic to identify potential scanning attacks. (Bou-Harb et al. [9] and Khraisat et al. [17] provide comprehensive surveys of cyber scanning techniques and IDS approaches, respectively.)

In this paper, we describe an approach to develop and validate predictive models of cyber threats. To manage model complexity and facilitate validation, we focus on port scanning and intrusion detection activities as part of discovery. We develop a mathematical model that provides insights into the process, and validate this model using cyber experimentation through emulation. As such, this paper makes the following contributions that add to the body of cyber scanning and detection models:

- Introduction of a novel, mathematical model that describes the rate at which an attacker scanning a network identifies vulnerable hosts. The model can also be used to calculate the probability that the attacker will be detected by an IDS over time and to evaluate attacker strategies.
- Validation of the model through comparison with cyber experiments that were conducted on an emulation testbed. Validation is demonstrated for a hypothetical scenario in which an attacker is scanning remote terminal units (RTU) to find vulnerabilities in a power grid system.
- Discussion on how the work presented herein on scanning and detection could be generalized for other cyber security analyses. We demonstrate that the combined usage of analytical, mathematical models with virtual cyber testbeds can help overcome the limitations that each technique has individually.

## 1.1 Related Work

Mathematical techniques have been used for decades to analyze network intrusions, but “most published work on intrusions covers the issue of intrusion detection rather than modelling of the intrusion process itself” [16]. Markov models have been commonly used to describe multi-stage attacks (e.g., [16]). McCarthy et al. have created partially observable, Markov decision process (POMDP) models to identify optimal detection configurations for detection and prevention of data exfiltration [18], and Miehling et al. use POMDPs to identify optimal defender strategies for halting attacker progression when network sensors are *noisy* [19]. Many efforts have explored the use of Susceptible-Infected-Recovered (SIR) epidemiology models and related variations to estimate the spread of scanning worms across the internet (e.g., [25] and [29]). Game theory has been leveraged frequently for modeling attacker-defender interactions that can occur during cyber attacks. Alpcan and Basar have developed game theoretic models to analyze distributed sensor networks and their effectiveness against attacks [5], [6], and Wang et al. use *attack-defend stochastic game Petri nets* to “model and analyze the security issues in enterprise network” [27], including scanning and detection. Sallhammar et al. use a combined Markov-game theoretic approach to predict attacker behaviors while defenders attempt to detect attacks [23]. Horak et al. develop a partially observable stochastic game (POSG) to explore defender strategies and how to affect attacker beliefs in the presence of deception networks [15], and Ahmadi et al. develop scalable solution methods for the POSGs [4].

Mathematical models are often used to evaluate attacker and defender strategies and to develop worst/best case scenarios (e.g., [27]). The high levels of abstraction common in these models can provide computational efficiencies, enabling analysts to explore large parameter spaces while only using limited computing resources. However, the levels of abstraction can limit the predictive power of these models and make validation through comparison with experiments difficult, if not impossible.

Virtual testbeds are becoming a more commonly used alternative for modeling and analyzing cyber security issues. In virtual testbeds such as the National Cyber Range [11], DeterLab [8], and LARIAT [14], virtual machines running real software and communications

protocols are used in place of hardware. In some setups, these virtual networks may be integrated with and communicate with actual hardware-in-the-loop. Additionally, they may also be integrated with simulations, such as ns-3, that have abstracted hardware and software implementations. Virtual testbeds, or emulations, have been used to analyze various cyber attack and defense techniques, including scanning and detection. Savaglia and Wang have used emulations to test Nmap and Nessus scanning tools [24], and Wang and Yang use emulations to select preferred vulnerability scanning tools [28]. MIT Lincoln Lab’s LARIAT testbed has been used extensively to develop and evaluate IDS tools [14].

Cyber emulations provide a higher fidelity representation for study of cyber security issues in large part because they run real software, enabling them in some cases to be used in a predictive manner. The use of LARIAT to evaluate IDSs is one such example. However, the generalizability of emulations has its limitations. Emulation often requires significant setup and computing resources to conduct cyber experiments. Analyses that require large numbers of experiments (e.g., 1000s) to capture uncertainties can become time prohibitive because emulated systems generally operate in real-time.

## 1.2 Paper Organization

The remainder of this paper is organized as follows. Section 2 describes the specific attack scenario considered herein, and Section 3 describes the emulation testbed setup and the mathematical modeling approaches used to represent the scenario. Experimental results are discussed in Section 4, and Section 5 contains conclusions and discussion on next research steps.

## 2 SCENARIO DESCRIPTION

The attack scenario we consider is motivated by the 2015 and 2016 Ukraine power grid cyber attacks [1] and, in particular, by the use of special-purpose ICS malware in the second attack. We provide a brief description of the attacks for context.

In the first of these attacks, the operations of three power companies were disrupted. Attackers first gained access to corporate networks, and subsequently pivoted to the Supervisory Control and Data Acquisition (SCADA) networks. Once on the SCADA networks, attackers manually leveraged legitimate ICS network functionality to disconnect substations, rendering over 225,000 customers without power for several hours.

In the second attack, attackers used the CRASHOVERRIDE malware on the SCADA network to impact a single transmission level substation. The malware leverages industry standard control protocols to remotely control networked field devices connected to sensors and actuators on the physical process. Attackers can specify IP addresses for remote terminal units (RTUs), and the malware will open circuit breakers connected to those RTUs, de-energizing components [1].

In this paper, we consider a hypothetical case study motivated by the CRASHOVERRIDE attack. We focus on the latter stage of an ICS attack in which the attacker has established a foothold on the ICS network. The priority for the attacker at that point is to obtain knowledge of the network layout (discovery) so as to target vulnerable components. To protect the network, network

administrators, referred to herein as defenders, deploy an IDS that can detect indicators of the discovery process; detection can be followed by preventative actions that halt further discovery or any potential subsequent actions against the ICS network or physical process.

More concretely, we consider a hypothetical scenario in which an attacker has a remote presence on an ICS control center that controls and monitors several power grid substations. The control center consists of several x86 SCADA machines connected via a routed TCP/IP network to RTUs in various substations. The RTUs themselves are connected to grid components at the substations (e.g. buses, relays, breakers) that enable remote grid monitoring (e.g. voltages, angles, currents) and switching (e.g. disconnecting branches via breakers).

We assume the attacker has control over a SCADA machine (e.g., an engineering workstation) and is in possession of malware that can deploy a malicious payload to an RTU and, thereby, disconnect branches to disrupt grid operations. The attacker, however, does not have *a priori* knowledge of the location of the RTUs in IP space. We assume that the attacker uses active network scanning to discover that information.

From a defensive standpoint, we assume defenders installed a firewall that blocks communications to the RTUs from anomalous sources (e.g. the engineering workstation), but holes exist in the firewall due to, for example, a misconfiguration that renders some of the RTUs vulnerable to malicious communications. The defenders also deployed an IDS on the network that can detect spurious TCP/IP events indicative of active scanning.

For the given scenario description, we pose the following questions of interest:

- (1) what are the potential attack effects as measured in terms of knowledge discovery?
- (2) what is the effectiveness of the defenses for detecting such an attack?

We quantify knowledge discovery in terms of the number of vulnerable RTU IP addresses discovered over time. Vulnerable RTUs are of high value to the attacker, and finding them sooner provides higher utility to the attacker. We quantify defense effectiveness in terms of the probability that the attacker is detected over time. We recognize that other measures can be considered, but we have chosen these options because they represent both value and time, important factors for both attackers and defenders. We further note that intrusion detection of port scans is inherently stochastic, meaning the timing of vulnerability discovery can be stochastic, as will be discussed in Section 3. Hence, attack and defense effectiveness measures will need to include stochastic variability.

### 3 METHODOLOGY

To explore the questions of interest, we use the following approach:

- Perform scanning and detection experiments in a virtual test bed to collect attack and defense measures of effectiveness.
- Develop a mathematical model describing scanning and detection processes to predict attack and defense measures of effectiveness.
- Compare the experimental results with model estimates.

This section describes this process in greater detail.

### 3.1 Scanning and Detection Technologies

In this paper, we assume that the attacker uses Nmap for scanning and that the defender uses Snort for detection. Other options exist for scanning and detection, but we selected these options for this paper because of their widespread use.

**3.1.1 Scanning.** Nmap is a popular open source utility for discovering hosts on a network and the services those hosts are offering [13]. Nmap performs discovery in two stages, host discovery and port scanning. In the host discovery stage, Nmap determines if hosts are *up* or *down* using ICMP type 8 (echo request) expecting hosts to respond with ICMP type 0 (echo reply) (*PE*)<sup>1</sup>. Nmap categorizes hosts that respond as *up* and those that do not as *down*. After all hosts are categorized, Nmap switches to the port scanning phase. In this phase Nmap further categorizes specified ports (*p*) on *up* hosts as open (service actively accepting TCP connections), closed (port accessible but no service listening), or filtered (no response).

Nmap supports several port scanning methods, and for this paper, we focused on the popular half-open SYN scan [9] (*sS*). In this method Nmap sends a SYN packet to a host port to initiate a connection and then waits for a response. For an open port, hosts accept the connection and respond with SYN/ACK. Nmap then closes the connection by sending a RST. For a closed port, hosts respond with SYN/RST.

On a congested network, packets may get dropped, resulting in no response for Nmap. Alternatively, probing a filtered port can result in no response for Nmap. Nmap can retry probes that get no response, but once the retry limit (*max-retries*) has been exceeded, unresponsive ports are categorized as filtered.

Nmap includes configuration parameters designed to avoid detection by IDS. For example, by default, Nmap scans hosts in sequential order, but the user can specify random ordering (*randomize-hosts*) to make the scan less obvious to IDS. Other important IDS avoidance parameters control the rate at which the port scanning takes place. Nmap scans multiple hosts in parallel by dividing the hosts into groups (*min-hostgroup*, *max-hostgroup*). It sends the initial SYN packet to all hosts in the current group and waits for responses from these hosts. Nmap does not continue onto the next group until it categorizes all hosts in the current group. It will delay for a specified amount of time (*scan-delay*) before trying to rescan hosts within the current group. Nmap will also delay for the specified amount of time after completing the current host group and before moving onto the next host group. In general, decreasing *max-hostgroup* and increasing *scan-delay* will decrease detection by IDSs but increase the length of time required to scan all of the hosts. In the aforementioned attack scenario, we assume that the attacker specifies host group and delay parameters in hopes of evading detection.

**3.1.2 Detection.** Snort is a widely deployed, open source IDS [10]. Snort includes the *sfPortscan* module that is designed to detect half-open SYN scans. The module takes advantage of the fact that the scanner has no prior knowledge of the network so most probes will result in negative (closed or filtered) responses. The *sfPortscan* module keeps track of the number of negative responses that are observed within a specified time window. If the number of negative

<sup>1</sup>In this section, we denote Nmap option flags as italicized text in parentheses, e.g., the (*PE*) parameter enables echo requests.

**Table 1: Snort sfPortscan settings: (C) and (F) denote thresholds for responses from closed and filtered ports**

Setting	Window Size	Threshold
Low	60 s	5 (C), NA (F)
Medium	90 s	7 (C), 30 (F)
High	600 s	3 (C), 30 (F)

responses exceeds a specified threshold, Snort generates an alert containing the source and destination IP addresses of the probe that exceeded the threshold.

The sfPortscan module can run at low, medium, and high settings that specify different window sizes and thresholds (Table 1). Increasing the setting increases the probability of detecting the Nmap scan, and also increases the probability of false positives that could occur due to normal traffic.

### 3.2 Emulation Experiments

The emulation testbed used in this study serves as the experimental platform by which model predictions are validated. The testbed consists of a high performance compute node hosting virtual machines that are connected via a virtual network that mirrors the scenario environment of interest.

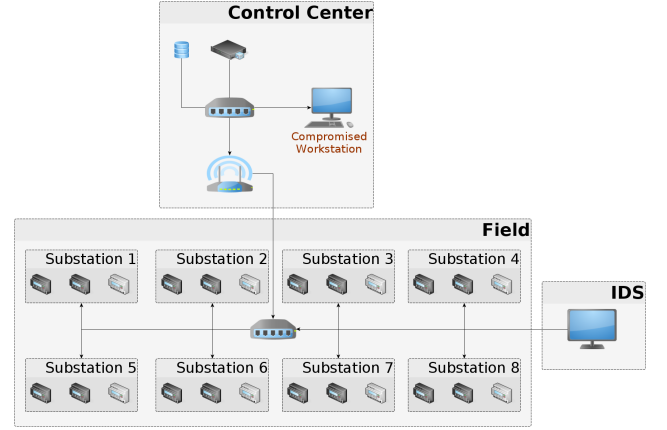
This section describes the emulation testbed. We delineate testbed components as belonging to either those that support the platform or those that are themselves the subjects of experimentation.

**3.2.1 Platform components.** The testbed’s primary platform software is the open source virtual machine management software *minimega* [12]. *minimega* is a virtualization platform that facilitates creation, deployment, and management of containers, virtual machines, and virtual networks on compute clusters. *minimega* uses QEMU [7] for hardware emulation and running unmodified operating systems, and Open vSwitch [22] for virtual switching.

We developed the python package *ScOrch* that directly integrates with *minimega* to enable automated experiment scenario orchestration and efficient instrumentation and data extraction. The package facilitates high-level specification of experiment scenarios in a modular fashion and provides an interface for implementation of scenario components, similar in concept to distributed experiment workflows (DEW) [20].

The hardware on which the emulation experiments run primarily consists of a single compute node with dual socket Intel ES-2683v4 2.10 GHz CPUs (32 total cores) and 512 GB DDR3 memory running a pared down version of Ubuntu 18.04 Linux. Although *minimega* supports deploying an emulation experiment across a network cluster of compute nodes with minimal additional configuration overhead, the experiments described herein were deployed on a single compute node.<sup>2</sup>

<sup>2</sup>Running on a single node reduces potential artifacts introduced from real networking supporting the emulation environments. We evaluated running multiple experiments in parallel on single compute nodes to decrease experiment times but have observed that parallelization introduces measurable artifacts that we do not yet understand well. We leave it as future work to explore the feasibility of parallelizing emulation experiments with minimal testbed artifacts.

**Figure 1: Notional ICS/SCADA topology used for the case-study. The components within substation boxes represent RTUs**

We note that these experiments do not include any background traffic, which in a real system might include additional SYN/RST messages. The number of spurious SYN/RST messages is likely low in ICS networks, so we believe this exclusion is reasonable for this experiment. Additionally, exclusion of background traffic has the effect of representing a best-case scenario for the attacker in that the scan will not be detected by Snort due to spurious SYN/RST messages that might occur from normal communications.

**3.2.2 Experiment components.** Each emulation experiment consists of a fixed (virtual) network of virtual machines deployed on a single compute node. The network topology is intended to mimic a notional ICS network managed by a single utility. The topology includes two switched subnetworks connected by a single router; one subnetwork represents the ICS control center network and the other represents the ICS field device network. The RTUs reside on the ICS field device network and serve to provide operations data from the substations to the control center. See Figure 1 for a schematic of the notional network layout.

The experiment includes four types of virtual machines: a scanning node, a detection node, a router, and RTUs. The scanning node represents the compromised engineering workstation that is controlled by the attacker. We implement the scanning node as a virtual machine, running a pared down version of Ubuntu 18.04 Linux, on the control center network; this machine runs Nmap 7.60 to scan the RTUs.

The detection node represents a computer attached to a SPAN (Switched Port ANalyzer) port on the ICS field device network switch. This machine runs the Snort IDS. In emulation, the SPAN port is implemented using Open vSwitch’s traffic mirroring capability [3]. The machine is implemented as a virtual machine, running a pared down version of Ubuntu 18.04 Linux, on the mirrored traffic network. This node runs Snort 2.9.13 during experiments. We perform packet captures on this node to calculate measures of effectiveness for the experiment.

The router represents a physical router connecting the two sub-networks, and it is implemented as a virtual machine running the

**Table 2: In-experiment components**

Component	Representation
ICS Network	virtualized network
Scanning Node	Ubuntu 18.04 VM with Nmap
Detection Node	Ubuntu 18.04 VM with Snort
Router	VyOS 3.13.11 VM
RTU	Ubuntu 18.04 VM with running service

VyOS 3.13.11 open source network operating system. RTUs in our emulation experiments represent physical RTU field devices on ICS networks. The relevant behavior of a real RTU with respect to scanning during the discovery stage of an attack is isolated to low-level TCP/IP network stack implementations. We postulate that this behavior is adequately captured by a general Linux kernel implementation; we motivate this claim by noting that some real RTUs run embedded Linux distributions (e.g. VersaTrak IPm RTU), and by pointing to the close agreement between the math model that abstracts away implementation details and the experimental results (see Section 4).

We implement RTUs as virtual machines running a pared down version of Ubuntu 18.04 Linux. For simplicity, we represent a running ICS protocol server on an RTU (e.g. Modbus or DNP3 server) by running a sshd server on the respective virtual machine. Because the response to a scanning probe is handled by the Linux kernel and not the application layer, the response behavior should be invariant to the actual service running. A subset of RTUs do not have the sshd service running; these represent "downed" or "backup" RTUs. An uninhibited scan to the sshd port of these RTUs gives a response of "closed" whereas an uninhibited scan to an RTU running the sshd service gives a response of "open."

Each RTU in the emulation experiment is running an iptables firewall. All RTUs have a stochastic drop rule that probabilistically drops incoming TCP packets. We use this approach to simulate packet loss on the network; the dropped packets manifest as timeouts from the perspective of Nmap port scanning<sup>3</sup>. Additionally, a subset of RTUs have a firewall rule blocking spurious traffic to the sshd service. A scan to the ssh port of these RTUs gives a response of "filtered."

Table 2 summarizes the various components that comprise a single emulation experiment.

**3.2.3 Experiment Workflow.** The experiment workflow consists of two phases: running the experiment/data collection and post-processing experiment data. The former is handled primarily by our custom-built scenario orchestration tool ScOrch.

A single experimental run consists of the following steps:

- (1) Set experiment parameters that establish topology, Nmap, and Snort configurations.
- (2) Create network topology representation.
- (3) Deploy virtual network on hardware according to the generated topology.

<sup>3</sup>Because the firewall rule works only on TCP packets, it will be ineffective during the initial host discovery stage of an Nmap scan; ICMP echo requests are used at that stage. Since we are assuming Snort is only configured for detecting port scanning behavior, we do not focus modeling efforts on initial host discovery using ICMP.

- (4) Begin SPAN interface packet capture on the detection node.
- (5) Run Snort IDS on the detection node.
- (6) Run Nmap scanner on the scanning node.
- (7) Once the scan is complete, extract Nmap log file.
- (8) Extract Snort log and alert files.
- (9) Extract packet capture file.

These steps are automated by the scenario orchestration tool and are defined in a scenario configuration file that contains parameters and behaviors for the given experimental scenario. Virtual machine behavior and data extraction is controlled through minimega's command and control interface. The interface includes a small binary running on each virtual machine; command upload and data ingestion and extraction occur through a serial port so as to be minimally invasive. Experiment parameters are described in detail in Section 4.1.

A given experimental run is executed by a single command-line call to the ScOrch command-line interface. Because of the various stochastic elements within the experiment, we run numerous trials of a single experimental configuration (typically 1000 trials) to facilitate statistical analysis. This process is executed by a script that calls ScOrch sequentially in a loop. Once all the runs are completed for a given experimental configuration, we post-process the data.

The first post-processing step is parsing the packet capture (PCAP) and alert files generated by Snort. For each experimental trial, we use *Scapy* [21] scripts to parse PCAPs to determine the time at which the RTUs responded to probes with either SYN/ACK or SYN/RST, indicating open and closed ports, respectively. For filtered ports we register the time as the time of last probe plus the packet response timeout ( $-max\text{-}rtt\text{-}timeout$ ). The Snort alert file is inspected to determine if and when Snort detected the scan. From these data, we create three time series for each experiment that specifies the cumulative number of open, closed, and filtered ports discovered before each one second time step. We can then average across all trials to determine the mean number of ports discovered over time. Similarly, we analyze the Snort alert files to estimate the probability that the scan is detected prior to each one-second time step.

### 3.3 Mathematical Model

We introduce a model of state transitions to represent discovery and detection by Nmap and Snort and the attack and defense effectiveness measures. The model relies on the following assumptions:

- $scan\text{-}delay > 0$ . This assumption reflects the use of Nmap parameters to avoid detection.
- $0 < max\text{-}rtt\text{-}timeout < scan\text{-}delay$ . This assumption is consistent with typical Nmap usage.
- The probability that a packet gets dropped and a probe times out for a closed or open RTU<sup>4</sup> is constant, and the probability is denoted  $p_{to}$ . These timeouts are assumed to be independent.

**3.3.1 Mathematical Specifications.** Consider the scenario described in Section 2 in which the network includes  $N$  RTUs. We use  $F$ ,  $O$ , and  $C$  to refer to the number of filtered, open, and closed RTUs,

<sup>4</sup>We use the terminology *open*, *closed*, and *filtered* RTUs to describe RTUs in which the scanned ports are open, closed, and filtered, respectively.

respectively ( $N = F + O + C$ ). (The attacker does not necessarily know the breakdown of RTUs.) Let  $h \in \mathbb{Z}^+$ ,  $d \in \mathbb{R}^+$ , and  $r \in \mathbb{Z}^+$  denote Nmap settings for host group size<sup>5</sup>, *scan-delay*, and *max-retries*, respectively. We consider discrete time steps where the  $k^{\text{th}}$  time step is  $t_k = k * d$ ,  $k = 0, \dots, T$  and  $T = \lceil \frac{N}{h} \rceil * (r + 2)$  is the maximum possible amount of time required to scan all the RTUs.

Let the notation  $S(k, \rho)$  denote the state

$$S(k, \rho) = \left( \begin{bmatrix} \mathcal{F}(k) \\ \phi(k) \\ \phi_i(k) \end{bmatrix}, \begin{bmatrix} O(k) \\ \omega(k) \\ \omega_i(k) \\ \omega_s(k) \end{bmatrix}, \begin{bmatrix} C(k) \\ \gamma(k) \\ \gamma_i(k) \\ \gamma_s(k) \end{bmatrix}, \rho \right) \quad (1)$$

where

- $\mathcal{F}(k)$ ,  $O(k)$ , and  $C(k)$  denote the number of filtered, open, and closed RTUs Nmap has not yet attempted to scan at least once by the  $k^{\text{th}}$  time step, respectively.
- $\gamma(k)$  denotes the number of closed RTUs in the current host group that Nmap has unsuccessfully attempted to scan by the  $k^{\text{th}}$  time step, i.e., they timed out  $\rho < r + 1$  times previously.
- $\gamma_i(k)$  denotes the number of closed RTUs that have been unsuccessfully scanned  $r + 1$  times by the  $k^{\text{th}}$  time step and have been designated as “inconclusive.” Nmap no longer attempts to scan these RTUs.
- $\gamma_s(k)$  denotes the number of closed RTUs that have been successfully scanned by the  $k^{\text{th}}$  time step.
- Definitions of filtered and open RTU variables (indicated with  $\phi$  and  $\omega$  and various subscripts) is analogous to definition of the closed RTU variables (indicated with  $\gamma$  and various subscripts).
- $\phi(k) + \omega(k) + \gamma(k) \leq h$ .
- $\rho = 0, \dots, r + 2$  denotes the number of times Nmap has attempted to scan the RTUs in the current host group. Nmap attempts to scan an RTU at most  $r + 1$  times. We use the notation  $\rho = r + 1$  to denote cases in which Nmap has either scanned all of the hosts in a group successfully (which may sometimes occur in less than  $r + 1$  attempts) or when at least one RTU in the current host group has been scanned  $r + 1$  times. We use the notation  $\rho = r + 2$  to denote that Nmap has paused for an additional delay after completing scans for the current host group.
- We constrain state  $S(k, \rho)$  such that all entries are integer-valued and  $F = \mathcal{F}(k) + \phi(k) + \phi_i(k)$ ,  $O = O(k) + \omega(k) + \omega_i(k) + \omega_s(k)$ ,  $C = C(k) + \gamma(k) + \gamma_i(k) + \gamma_s(k)$  for all  $k$ .
- If  $\rho \geq r + 1$ , then  $\phi + \omega + \gamma = 0$ . This constraint is included because  $\rho \geq r + 1$  indicates when Nmap has finished scanning all RTUs in the current host group.

The number of feasible states is finite, so we can enumerate them as  $S^j$ ,  $j = 1, \dots, M$ , where  $M$  denotes the total number of feasible states. We use superscript notation to denote the respective state value of  $S^j$ , i.e.,  $\rho^j$  is the value of  $\rho$  for state  $S^j$ . We define a “future” to be a sequence of feasible states realized at time steps  $k$ . We use the notation  $\{j_k\}_{k=0}^T$ ,  $j_k \in \{1, \dots, M\}$ , to denote a future in which the  $j_k^{\text{th}}$  state is realized at time step  $k$ . To calculate the probability that a future will be realized, we need to specify transition probabilities

<sup>5</sup>We use the term “host group size” to denote when the Nmap settings *min-hostgroup* = *max-hostgroup*.

$P(S^j \rightarrow S^{j_{k+1}})$ , i.e., the probability of transitioning from state  $S^j$  at the  $k^{\text{th}}$  time step to  $S^{j_{k+1}}$  at the  $k + 1^{\text{th}}$  time step. We define transition probabilities for the following cases.

**Case 1**  $\rho^j = r + 1$ : Nmap has attempted to scan some RTUs in the current host group  $r + 1$  times, so Nmap will wait for one time step (without scanning any RTUs) before continuing. Hence,

$$P(S^j \rightarrow S^{j_{k+1}}) = 1 \quad (2)$$

where  $\rho^{j_{k+1}} = r + 2$ ; otherwise,  $S^j$  and  $S^{j_{k+1}}$  are identical. No other state transitions are feasible.

**Case 2**  $0 < \rho^j < r$ ,  $\phi^j + \omega^j + \gamma^j \neq 0$ : Nmap has attempted to scan RTUs in the current host group but has not successfully scanned all of the RTUs and has not yet reached the maximum number of scan attempts  $r + 1$ . In this instance Nmap does not try to scan a new host group of RTUs; instead, Nmap tries again on the remaining  $\phi^j + \omega^j + \gamma^j$  RTUs. Multiple non-zero transition probabilities to different states may exist, and they are calculated as

$$P(S^j \rightarrow S^{j_{k+1}}) = p_{to}^{y_\omega + y_\gamma} (1 - p_{to})^{x_\omega + x_\gamma} \binom{\omega^j}{x_\omega} \binom{\gamma^j}{x_\gamma} \quad (3)$$

where  $S^{j_{k+1}}$  can be one of several states that is constrained to

- $\omega_s^{j_{k+1}} \geq \omega_s^j$ ,  $\gamma_s^{j_{k+1}} \geq \gamma_s^j$ ,  $\omega^{j_{k+1}} \leq \omega^j$ ,  $\gamma^{j_{k+1}} \leq \gamma^j$
- $\omega_s^{j_{k+1}} - \omega_s^j = \omega^j - \omega^{j_{k+1}}$  and  $\gamma_s^{j_{k+1}} - \gamma_s^j = \gamma^j - \gamma^{j_{k+1}}$
- 

$$\rho^{j_{k+1}} = \begin{cases} \rho^j + 1 & \text{if } \phi^{j_{k+1}} + \omega^{j_{k+1}} + \gamma^{j_{k+1}} \neq 0 \\ r + 1 & \text{if } \phi^{j_{k+1}} + \omega^{j_{k+1}} + \gamma^{j_{k+1}} = 0 \end{cases}$$

- All state entries not mentioned in the above constraints are identical between  $S^j$  and  $S^{j_{k+1}}$ .

We define the following notation for use in (3).

$$\begin{aligned} x_\omega &= \omega_s^{j_{k+1}} - \omega_s^j & y_\omega &= \omega^j - x_\omega \\ x_\gamma &= \gamma_s^{j_{k+1}} - \gamma_s^j & y_\gamma &= \gamma^j - x_\gamma \end{aligned}$$

$$\binom{A}{a} = \frac{A!}{a! (A - a)!}, \text{ for } A \geq a, A, a \geq 0$$

**Case 3**  $\rho^j = r$ ,  $\phi^j + \omega^j + \gamma^j \neq 0$ : This case is similar to Case 2, but Nmap will only attempt one more scan of the remaining  $\phi^j + \omega^j + \gamma^j$  RTUs in the current host group. Hence,  $\omega^{j_{k+1}} = \gamma^{j_{k+1}} = \phi^{j_{k+1}} = 0$ , and all of the RTUs in the current host group will be assigned to a “success” or “inconclusive” category.

Non-zero transition probabilities are calculated in the same manner as shown in (3) for Case 2. However, the constraints on  $S^{j_{k+1}}$  are slightly different and are as follows:

- $\omega^{j_{k+1}} = \gamma^{j_{k+1}} = \phi^{j_{k+1}} = 0$
- $\omega_s^{j_{k+1}} \geq \omega_s^j$ ,  $\gamma_s^{j_{k+1}} \geq \gamma_s^j$
- $\omega_i^{j_{k+1}} \geq \omega_i^j$ ,  $\gamma_i^{j_{k+1}} \geq \gamma_i^j$
- $\phi_i^{j_{k+1}} = \phi_i^j + \phi^j$
- $\omega_s^{j_{k+1}} - \omega_s^j + \omega_i^{j_{k+1}} - \omega_i^j = \omega^j$ ,  $\gamma_s^{j_{k+1}} - \gamma_s^j + \gamma_i^{j_{k+1}} - \gamma_i^j = \gamma^j$
- $\rho^{j_{k+1}} = r + 1$
- All state entries not mentioned in the above constraints are identical between  $S^j$  and  $S^{j_{k+1}}$ .

**Case 4**  $\rho^j = r + 2$ ,  $\phi^j + \omega^j + \gamma^j = 0$ : When  $\rho^j = r + 2$ , Nmap paused for the last time step. Over the next time step, Nmap must select new RTUs for the next time step and then attempt to scan

them. This is a multi-step process, and so calculation of transition probabilities for this case is also a multi-step process. Steps 1 and 2 correspond to selecting RTUs for the new host group, and Steps 3 - 5 correspond to determining if the scans were successful or not. Calculations for the steps are as follows.

**Step 1, determine intermediate transition states.** For state  $S^{jk}$ , we define an *intermediate transition state*  $\hat{S}^{jk}$  to be any state that satisfies the following constraints:

- $\hat{\phi}^{jk} + \hat{\omega}^{jk} + \hat{\gamma}^{jk} = m$  where  $m = \min\{h, \mathcal{F}^{jk} + \mathcal{O}^{jk} + \mathcal{C}^{jk}\}$
- $\hat{\phi}^{jk} \leq \mathcal{F}^{jk}$ ,  $\hat{\omega}^{jk} \leq \mathcal{O}^{jk}$ , and  $\hat{\gamma}^{jk} \leq \mathcal{C}^{jk}$
- $\hat{\mathcal{F}}^{jk} = \mathcal{F}^{jk} - \hat{\phi}^{jk}$ ,  $\hat{\mathcal{O}}^{jk} = \mathcal{O}^{jk} - \hat{\omega}^{jk}$ ,  $\hat{\mathcal{C}}^{jk} = \mathcal{C}^{jk} - \hat{\gamma}^{jk}$
- $\hat{\rho}^{jk} = 0$
- All state entries not mentioned in the above constraints are identical between  $S^{jk}$  and  $\hat{S}^{jk}$ .

**Step 2, calculate intermediate transition probabilities.** We use the notation  $P(S^{jk} \rightarrow \hat{S}^{jk})$  to denote the probability that the next host group will consist of  $\hat{\phi}^{jk}$ ,  $\hat{\omega}^{jk}$ , and  $\hat{\gamma}^{jk}$  filtered, open, and closed RTUs, respectively. We calculate the probability as

$$P(S^{jk} \rightarrow \hat{S}^{jk}) = \frac{\binom{\mathcal{F}^{jk}}{\hat{\phi}^{jk}} \binom{\mathcal{O}^{jk}}{\hat{\omega}^{jk}} \binom{\mathcal{C}^{jk}}{\hat{\gamma}^{jk}}}{\binom{\mathcal{F}^{jk} + \mathcal{O}^{jk} + \mathcal{C}^{jk}}{m}}$$

**Step 3, determine feasible transition states.** The next step is to determine for a given intermediate state  $\hat{S}^{jk}$ , what are the possible states  $\hat{S}^{j_{k+1}}$  that can result at the next time step and reflect whether the RTUs in the selected host group were successfully scanned or not. We specify that feasible states  $\hat{S}^{j_{k+1}}$  to have the following properties:

- $\hat{\omega}^{j_{k+1}} \geq \omega^{j_{k+1}}$ , and  $\hat{\gamma}^{j_{k+1}} \geq \gamma^{j_{k+1}}$
- $\omega_s^{j_{k+1}} = \hat{\omega}_s^{j_k} + \hat{\omega}^{j_k} - \omega^{j_{k+1}}$
- $\gamma_s^{j_{k+1}} = \hat{\gamma}_s^{j_k} + \hat{\gamma}^{j_k} - \gamma^{j_{k+1}}$
- 

$$\rho^{j_{k+1}} = \begin{cases} \rho^{j_k} + 1 & \text{if } \phi^{j_{k+1}} + \omega^{j_{k+1}} + \gamma^{j_{k+1}} \neq 0 \\ r + 1 & \text{if } \phi^{j_{k+1}} + \omega^{j_{k+1}} + \gamma^{j_{k+1}} = 0 \end{cases}$$

- All state entries not mentioned in the above constraints are identical between  $\hat{S}^{j_{k+1}}$  and  $\hat{S}^{j_k}$ .

**Step 4, calculate transition probability from intermediate state.** Calculation of  $P(\hat{S}^{j_k} \rightarrow \hat{S}^{j_{k+1}})$  and the constraints on  $\hat{S}^{j_{k+1}}$  are identical to the process in Case 2 if we replace  $S^{j_k}$  with  $\hat{S}^{j_k}$ .

**Step 5, calculate final transition probability.** We calculate

$$P(S^{j_k} \rightarrow S^{j_{k+1}}) = \sum_{\hat{S}^{j_k}} P(S^{j_k} \rightarrow \hat{S}^{j_k}) P(\hat{S}^{j_k} \rightarrow S^{j_{k+1}}).$$

**3.3.2 Algorithmic Implementation.** Given the specification of states and transition probabilities, we describe the numerical implementation for calculating futures and the probability of their occurrence. The result is the ability to calculate attack measures of effectiveness:  $\mathbb{E}(\Omega_k)$ ,  $\mathbb{E}(\Gamma_k)$ , and  $\mathbb{E}(\Upsilon_k)$ , the expected number of open and closed RTUs successfully scanned and the expected number of RTUs designated as inconclusive by the  $k^{\text{th}}$  time step. We can also calculate the defensive measure of effectiveness, the probability that the attacker is detected by the  $k^{\text{th}}$  time step.

Let  $f^t = \{j_k\}_{k=0}^t$ ,  $t \leq T \in \mathbb{Z}^+$ , denote a “partial” future where state  $S^{j_k}$  is realized at time step  $k$ . We calculate  $f^t$ ,  $\mathbb{E}(\Omega_k)$ ,  $\mathbb{E}(\Gamma_k)$ , and  $\mathbb{E}(\Upsilon_k)$ , with Algorithm 1.

---

**Algorithm 1:** Scanning Model Algorithm

---

**Result:** Calculate  $\mathbb{E}(\Omega_k)$ ,  $\mathbb{E}(\Gamma_k)$ , and  $\mathbb{E}(\Upsilon_k)$ , and set of possible  $f^T$  and  $P(f^T)$

initialization: set  $h, r, p_{t0}$  and initial state  $f^1 = S^{j_1}$  and probability  $P(f^1) = 1$ ;

**foreach**  $k = 0, \dots, T$  **do**

$\mathbb{E}(\Omega_k) \leftarrow 0$ ;

$\mathbb{E}(\Gamma_k) \leftarrow 0$ ;

$\mathbb{E}(\Upsilon_k) \leftarrow 0$ ;

**end**

**while**  $k \leq T$  **do**

**foreach**  $S^{j_k}$  where  $P(f^k = \{1, \dots, j_k\}) \neq 0$  **do**

        Find all  $S^{j_{k+1}}$  such that  $P(S^{j_k} \rightarrow S^{j_{k+1}}) \neq 0$ ;

$P(f^{k+1} = \{1, \dots, j_k, j_{k+1}\}) \leftarrow$   
          $P(\{1, \dots, j_k\}) P(S^{j_k} \rightarrow S^{j_{k+1}})$ ;

**end**

$k \leftarrow k + 1$

**end**

**foreach**  $f^T$ ,  $P(f^T) \neq 0$  **do**

**foreach**  $k = 0, \dots, T$  **do**

$\mathbb{E}(\Omega_k) \leftarrow \mathbb{E}(\Omega_k) + \omega_s^{j_k} P(f^T)$ ;

$\mathbb{E}(\Gamma_k) \leftarrow \mathbb{E}(\Gamma_k) + \gamma_s^{j_k} P(f^T)$ ;

$\mathbb{E}(\Upsilon_k) \leftarrow \mathbb{E}(\Upsilon_k) + (\phi_i^{j_k} + \gamma_i^{j_k} + \omega_i^{j_k}) P(f^T)$ ;

**end**

**end**

---

Let  $P_D \in [0, 1]^T$  be a vector in which the  $k^{\text{th}}$  entry  $P_D^k$  denotes the probability that the attacker is detected prior to the  $k^{\text{th}}$  time step if the attacker continues the scan until the scan is detected or completed. Let  $w$  be an integer such that  $w * d =$  the window size specified in Snort, and  $\theta$  is the detection threshold. The vector  $P_D$  is calculated with Algorithm 2. Algorithms 1 and 2 were implemented in MATLAB R2018a.

## 4 EXPERIMENTAL RESULTS

In this section, we present results from the emulation experiments and mathematical model. We describe input parameters and the compare experimental and modeling results.

### 4.1 Input Parameters and Output Variables

We consider three sets of static input parameters that are fixed across every experiment in this study (Table 3).

We consider two distinct attacker strategies to parameterize Nmap (Table 4): (1) a *slow, stealthy* strategy that prioritizes stealth over speed and (2) a *fast, loud* strategy that prioritizes speed over stealth. The specific parameter values for each scenario were chosen

**Algorithm 2:** Detection Model Algorithm

---

**Result:** Calculate  $P_D$ .

initialization: set  $w, \theta$ ; specify all futures  $f^T$  and  $P(f^T)$

from Alg. 1;

$B \leftarrow \lceil \frac{T}{w} \rceil$ ;

**foreach**  $k = 1, \dots, T$  **do**

$P_D^k \leftarrow 0$ ;

**end**

**foreach**  $f^T$  **do**

$b \leftarrow 1$ ;

$\sigma = 0$ ;

**while**  $b < B + 1$  **do**

$c_0 \leftarrow 0$ ;

$\tau \leftarrow \min\{w, T - b * w\}$ ;

**foreach**  $k = 1, \dots, \tau$  **do**

$c_k \leftarrow c_{k-1} + \gamma_s^{J_{\sigma+k}}$ ;

**end**

**if**  $c_\tau \geq \theta$  **then**

$\mu \leftarrow \text{argmin}(c_k \geq \theta)$ ;

**foreach**  $l \geq \sigma + \mu$  **do**

$P_D^l \leftarrow P_D^l + P(f^T)$ ;

**end**

$b \leftarrow B + 1$ ;

**else**

$b \leftarrow b + 1$ ;

$\sigma \leftarrow \sigma + w$ ;

**end**

**end**

**end**

---

**Table 3:** Static Input Parameters

	Parameter	Value
Network	Layout	1 router, 2 subnets
	Nodes	1 router, 1 scanning node 1 detection node, 24 RTUs: 4 open, 8 closed, 12 filtered
	Packet loss	10%
	IP Ranges	control center subnet: 10.0.2.0/24 field device subnet: 10.0.1.0/24
Nmap	Scan subnet	10.0.1.0/24
	Scan port	22
	Probe timeout	0.5 seconds
	Number of resends	1
Snort	Active module	sfportscan
	Sense type	port sweep
	Sense level	low (60 seconds, 5 events)

so as to give non-trivial detection results when the sensing level is set to *low* in Snort.

**Table 4:** Varied Experiment Parameters

Strategy	Parameters
Slow, Stealthy	$host\text{-}group = 4$ $scan\text{-}delay = 10$ s
Fast, Loud	$host\text{-}group = 6$ $scan\text{-}delay = 5$ s

We select the average number of open ports discovered over time by Nmap as the attack measure of effectiveness; we select the cumulative probability of detection over time as the defense measure of effectiveness.

## 4.2 Comparison

Figure 2 shows the mean number of open RTUs discovered by Nmap over time for the slow, stealthy strategy, as predicted by the math model and observed in the emulation experiments. The scans are completed within 180 seconds, and the two means agree closely at all time steps. At the end of the scan, an average of 3.96 discovered open ports are calculated by the math model, and the emulation experiments produce a mean estimate of 3.954 with 95% confidence interval [3.926, 3.982]. The mean from the math model falls within the 95% confidence interval at all time steps. Large increases in the means are observed approximately every 30 seconds; the increase occurs at this frequency because a new group of RTUs to scan is often selected after 30 seconds, i.e., after  $r + 2 = 3$  delays that correspond to 2 scan attempts plus 1 pause. Similar agreement between means and increases are observed for identification of closed and filtered RTUs but are not shown because of page length limitations.

Figure 3 shows the probability that the attacker is detected over time for the slow, stealthy strategy. At the time the scan is completed, the probability of detection is calculated with the math model to be 0.359, and the emulation experiments result in a probability of 0.379 with 95% confidence interval [0.349, 0.409]. The estimate from the math model falls within the 95% confidence interval from the emulation experiment at all time steps, but the estimates are not quite as close as the estimates for discovery of open RTUs. We believe that this larger difference is due to the precision of time steps in the math model and the less precise Nmap and Snort implementations in the emulation experiments. For example, delays between Nmap scans were often a slightly more or less than 10 seconds. In some experiments, we noted that Snort detected an attack before it was theoretically possible in the math model. Also, a detection would sometimes occur when the 5<sup>th</sup> event occurred at shortly after 60 seconds (e.g., at 60.1 seconds) in an emulation experiment. These seemingly small differences do not significantly affect the estimated Nmap scanning results; discovery of the open RTUs will happen slightly faster or slower. However, because the timing and order of events matters for detection, these differences in timing precision have a greater effect on the detection results. Nevertheless, the math model provides accurate estimates for detection and emulation experiments.

Figures 4 and 5 show the RTU discovery and probability of detection results over time for the fast, loud strategy. As with the first



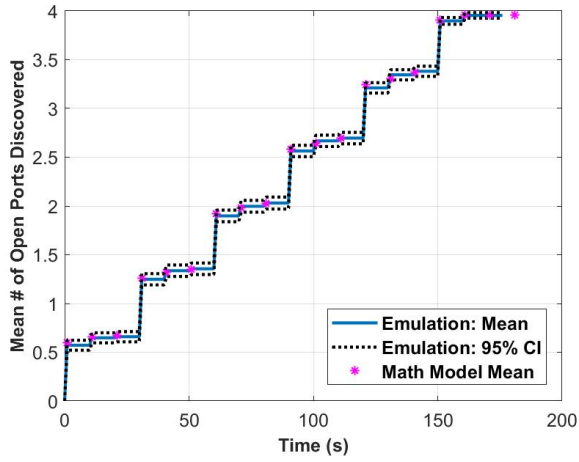


Figure 2: Discovery of Open RTUs: slow, stealthy strategy

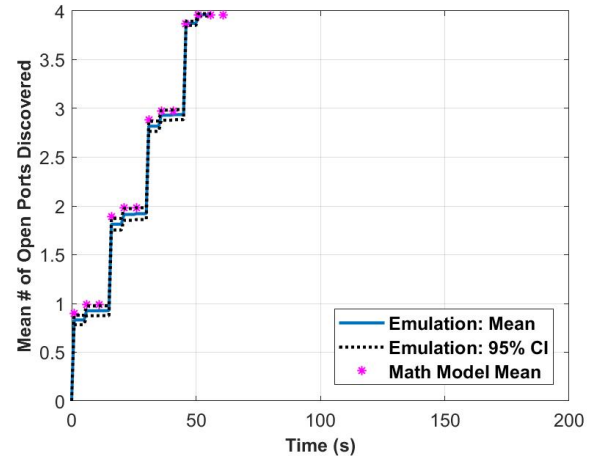


Figure 4: Discovery of Open RTUs: fast, loud strategy

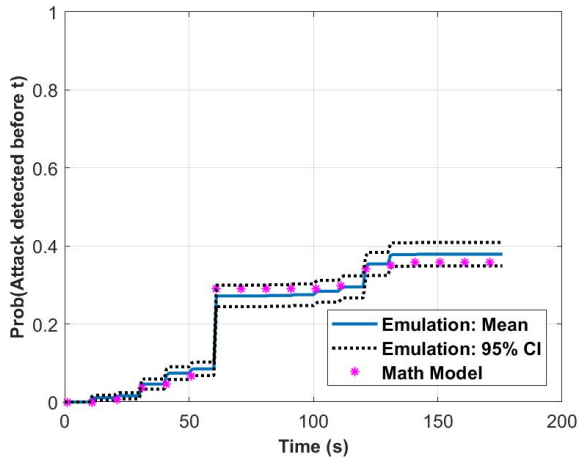


Figure 3: Detection of Attacker: slow, stealthy strategy

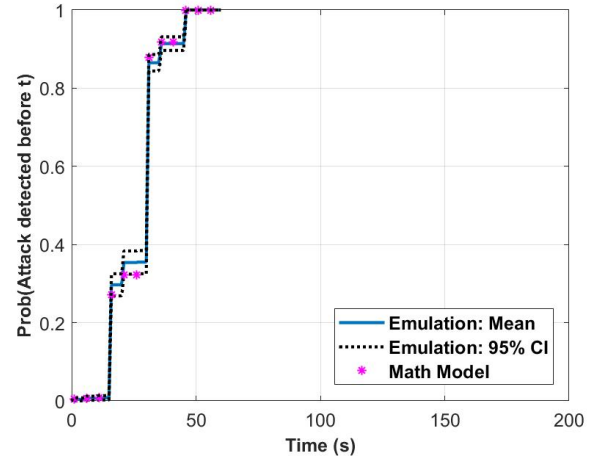


Figure 5: Detection of Attacker: fast, loud strategy

strategy, results from the math model and emulation experiments match well, and results from the math model fall within the 95% confidence intervals calculated from the emulation experiments.

For the fast, loud strategy, the scan is completed within 60 seconds. At the end of the scan, the mean number of open RTUs discovered is nearly identical to the quantity found with the first strategy (3.96 from math model and 3.960 with 95% confidence interval [3.948, 3.972]); however, the attacker gains this information much faster. The consequence of faster knowledge gain is a significant increase in detection probability; the attacker is almost guaranteed to be detected by the time the scan is complete. In fact, a detection happened in each of the 1000 emulation experiments.

These results alone are not sufficient to specify an *optimal* scanning strategy; rather, they start to show how the model can be used to describe the tradeoffs between different strategies. These tradeoffs, in combination with specification of attacker preferences,

could be used to rank strategies and, with some additional effort, possibly to identify optimal scanning strategies for the attacker.

We note that similar calculations and experiments were performed for a variety of host group sizes (1 to 12 RTUs) and delay parameter values (5 to 30 seconds). Results were as expected; that is, larger host group sizes and smaller delays increased detection probabilities and decreased times to complete scans and vice versa. Similar agreement was observed between results from the math model and emulation experiments. For the sake of brevity, those results are not shown herein.

## 5 CONCLUSIONS

Mathematical modeling has been used for decades to explore a variety of cyber security issues, including network scanning and detection. However, model validation has often been ignored or insufficiently addressed. In this paper we introduce a new mathematical model of scanning and detection activities that commonly

occurs during the discovery phase of a cyber attack. The model represents a scenario in which the attacker uses Nmap to scan for vulnerable RTUs in an electrical grid and the defender uses Snort to monitor network traffic to detect scanning attacks. The model describes scan/detect dynamics, and we validated the model through comparison with emulation experiments that were conducted using virtual testbeds. The two approaches were shown to provide statistically similar results.

Validation of the model is a significant result; however, the authors recognize the limited scope of this specific study. (The limited scope was intentionally selected to facilitate and enable validation.) Future efforts to extend this work could include:

- Representation of other tools: the current model represents Snort and Nmap, but other network scanning and detection tools could be considered, as well. Furthermore, other techniques (e.g., passive scanning) could be explored.
- Increased Scale: we conducted experiments with 10x RTUs and parallelized experiments to conduct them in a more time efficient manner. We are analyzing results to see how distributions are affected by the larger numbers of RTUs and if parallelization affects experimental results.
- Optimization: we did not discuss strategy optimization, but we are exploring mathematical tools for identifying optimal attacker (scanning) and defender (detection) strategies.
- Physical impacts: we did not discuss physical impacts on the power grid (e.g., loss of load) that the attacker could cause. Doing so would require the additional step of modeling the power grid and integrating that model with the testbed.
- Additional statistical measures: this paper contains comparisons of *mean* estimates from the math model and the emulation experiments, but other statistical measures are commonly considered for risk analyses. Measures such as *value at risk*, *worst case*, *distribution quantiles*, and others could be compared.

More generally, we provide a demonstration of how mathematical modeling and emulation can be used together to validate and develop models. This example focuses on scanning and detection, but a joint modeling-emulation approach could be used for other studies. The benefit would be development of computationally efficient, validated math models that produce high-confidence results.

## ACKNOWLEDGMENTS

The authors thank L. P. Swiler, G. Geraci, and E. Acquesta for their thoughtful insights on model development and uncertainty analysis.

## REFERENCES

- [1] 2017. *CRASHOVERRIDE Analysis of the Threat to Electric Grid Operations*. Technical Report version 2.20170613. Dragos Inc., Hanover, MD.
- [2] 2017. *TRISIS Malware Analysis of Safety System Targeted Malware*. Technical Report version 1.20171213. Dragos Inc., Hanover, MD.
- [3] 2019. Open vSwitch Basic Configuration. <http://docs.openvswitch.org/en/latest/faq/configuration/>
- [4] M. Ahmadi, M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen, and U. Topcu. 2018. The Partially Observable Games We Play for Cyber Deception. *CoRR* abs/1810.00092 (2018). [arXiv:1810.00092](http://arxiv.org/abs/1810.00092) <http://arxiv.org/abs/1810.00092>
- [5] T. Alpcan and T. Basar. 2003. A game theoretic approach to decision and analysis in network intrusion detection. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, Vol. 3. 2595–2600 Vol.3.
- [6] T. Alpcan and T. Basar. 2004. A game theoretic analysis of intrusion detection in access control systems. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, Vol. 2. 1568–1573 Vol.2.
- [7] F. Bellard. 2005. QEMU, a Fast and Portable Dynamic Translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC '05)*. USENIX Association, Berkeley, CA, USA, 41–41.
- [8] T. Benzel. 2011. The Science of Cyber Security Experimentation: The DETER Project. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*. ACM, New York, NY, USA, 137–148.
- [9] E. Bou-Harb, M. Debbabi, and C. Assi. 2014. Cyber Scanning: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* 16, 3 (Third 2014), 1496–1519.
- [10] Cisco. 2019. Snort intrusion detection and prevention system. <https://www.snort.org/>.
- [11] B. Ferguson, A. Tall, and D. Olsen. 2014. National Cyber Range Overview. In *2014 IEEE Military Communications Conference*. 123–128.
- [12] D. Fritz and J. Floren. 2013. minimega, Version 00.
- [13] G. Lyon. 2019. Nmap: the network mapper. <http://nmap.org/>.
- [14] J. W. Haines, L. M. Rossey, R. P. Lippmann, and R. K. Cunningham. 2001. Extending the DARPA off-line intrusion detection evaluations. In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, Vol. 1. 35–45.
- [15] K. Horák, Q. Zhu, and B. Bošanský. 2017. Manipulating Adversary's Belief: A Dynamic Game Approach to Deception by Design for Proactive Network Security. In *Decision and Game Theory for Security*, S. Rass, B. An, C. Kiekintveld, F. Fang, and S. Schauer (Eds.). Springer International Publishing, Cham, 273–294.
- [16] S. K. Katsikas, D. Gritzalis, and P. Spirakis. 1996. *Attack modeling in open network environments*. Springer US, Boston, MA, 268–277.
- [17] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* 2, 1 (2019).
- [18] S. M. Mc Carthy, A. Sinha, M. Tambe, and P. a Manadhata. 2016. Data Exfiltration Detection and Prevention: Virtually Distributed POMDPs for Practically Safer Networks. In *Decision and Game Theory for Security*, Q. Zhu, T. Alpcan, E. Panaousis, M. Tambe, and W. Casey (Eds.). Springer International Publishing, Cham, 39–61.
- [19] E. Miehling, M. Rasouli, and D. Teneketzis. 2018. A POMDP Approach to the Dynamic Defense of Large-Scale Cyber Networks. *IEEE Transactions on Information Forensics and Security* 13, 10 (Oct 2018), 2490–2505.
- [20] J. Mirkovic, G. Bartlett, and J. Blythe. 2018. DEW: Distributed Experiment Workflows. In *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*. USENIX Association, Baltimore, MD.
- [21] P. Biondi. 2019. Scapy. <https://scapy.net/>.
- [22] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. 2009. Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*.
- [23] K. Sallhammar, S. J. Knapkog, and B. E. Helvik. 2005. Using Stochastic Game Theory to Compute the Expected Behavior of Attackers. In *2005 Symposium on Applications and the Internet Workshops (SAINT 2005 Workshops)*. 102–105.
- [24] J. Savaglia and P. Wang. 2017. Cybersecurity Vulnerability Analysis via Virtualization. *Issues in Information Systems* 18, 4 (2017).
- [25] G. Serazzi and S. Zanero. 2004. *Attack modeling in open network environments*. Springer, Berlin, 26–50.
- [26] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas. 2018. *MITRE ATT&CK<sup>TM</sup>: Design and Philosophy*. Technical Report MP1 8 03 60. The MITRE Corporation, McLean, VA.
- [27] Y. Wang, J. Li, Kun Meng, Chuang Lin, and Xueqi Cheng. 2013. Modeling and security analysis of enterprise network using attack&Adefense stochastic game Petri nets. *Security and Communication Networks* 6, 1 (2013), 89–99. [arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.535](https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.535)
- [28] Y. Wang and J. Yang. 2017. Ethical Hacking and Network Defense: Choose Your Best Network Vulnerability Scanning Tool. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops*. 110–113.
- [29] S. Yu, G. Gu, A. Barnawi, S. Guo, and I. Stojmenovic. 2015. Malware Propagation in Large-Scale Networks. *IEEE Transactions on Knowledge and Data Engineering* 27, 1 (Jan 2015), 170–179.