

# **Continuous Security Metrics for Prevalent Network Threats: Introduction and First Four Metrics**

**R.P. Lippmann  
J.F. Riordan  
T.H. Yu  
K.K. Watson**

22 May 2012

---

**Lincoln Laboratory**  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
*LEXINGTON, MASSACHUSETTS*



---

Prepared for the Department of Defense under Air Force Contract FA8721-05-C-0002.

Approved for public release; distribution is unlimited.

This report is based on studies performed at Lincoln Laboratory, a federally funded research and development center operated by Massachusetts Institute of Technology. This work was sponsored by the Department of Defense under Air Force Contract FA8721-05-C-0002.

This report may be reproduced to satisfy needs of U.S. Government agencies.

The 66th Air Base Group Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

  
Gary Tutungian  
Administrative Contracting Officer  
Enterprise Acquisition Division

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission has been given to destroy this document when it is no longer needed.

Massachusetts Institute of Technology  
Lincoln Laboratory

Continuous Security Metrics for Prevalent Network Threats:  
Introduction and First Four Metrics

*R.P. Lippmann*  
*J.F. Riordan*  
*T.K. Yu*  
*Group 68*

*K.K. Watson*  
*NSA*

Project Report IA-3

22 May 2012

Approved for public release; distribution is unlimited.

Lexington

Massachusetts

**This page intentionally left blank.**

## EXECUTIVE SUMMARY

The goal of this work is to introduce meaningful security metrics that motivate effective improvements in network security. We present a methodology for directly deriving security metrics from realistic mathematical models of adversarial behaviors and systems and also a maturity model to guide the adoption and use of these metrics. Four security metrics are described that assess the risk from prevalent network threats. These can be computed automatically and continuously on a network to assess the effectiveness of controls. Each new metric directly assesses the effect of controls that mitigate vulnerabilities, continuously estimates the risk from one adversary, and provides direct insight into what changes must be made to improve security. Details of an explicit maturity model are provided for each metric that guide security practitioners through three stages where they (1) Develop foundational understanding, tools and procedures, (2) Make accurate and timely measurements that cover all relevant network components and specify security conditions to test, and (3) Perform continuous risk assessments and network improvements. Metrics are designed to address specific threats, maintain practicality and simplicity, and motivate risk reduction. These initial four metrics and additional ones we are developing should be added incrementally to a network to gradually improve overall security as scores drop to acceptable levels and the risks from associated cyber threats are mitigated.

**This page intentionally left blank.**

## ACKNOWLEDGMENT

We would like to thank George Moore from the Department of State for his consistently insightful and cogent remarks.

**This page intentionally left blank.**

## TABLE OF CONTENTS

	<b>Page</b>
Executive Summary	iii
Acknowledgment	v
List of Illustrations	xi
List of Tables	xiii
1. INTRODUCTION	1
2. LIMITATIONS OF PAST METRICS USED TO ASSESS RISK	3
3. RISK ASSESSMENT AS A SCIENTIFIC METHOD	9
4. GOALS	11
5. A METRIC MATURITY MODEL	13
6. DESIRED METRIC CHARACTERISTICS	15
7. OVERVIEW OF FIRST FOUR METRICS	17
8. ADVERSARIAL INTERACTION BETWEEN ATTACKERS AND DEFENDERS	19
9. NOTATIONAL CONVENTIONS AND INTRODUCTION	21

10.	MODELING DEFENDER AND ATTACKER SCANNING	25
10.1	Instantaneous Scanning	25
10.2	Periodic Scanning	25
10.3	Poisson Scanning	30
10.4	Pareto Sampling	31
10.5	Comparison of Four Types of Sampling	35
11.	USING PERIODIC SAMPLING ADVERSARY MODELS TO COMPUTE COMPROMISE PROBABILITIES	37
12.	COMBINING THE EFFECT OF MANY INDIVIDUAL INSECURE CONDITIONS	41
13.	CONSIDERING ERRORS CAUSED BY SAMPLING INSECURE CONDITION DURATIONS	45
14.	COMMON METRIC COMPONENTS AND OPTIONS	51
15.	LR-1 METRICS FOR ATTACKERS EXPLOITING UNAUTHORIZED DEVICES	55
15.1	LR-1 Checklist	57
15.2	LR-1 Capability Deficit Metric	59
15.3	LR-1 Operational Metric	60
16.	LR-2 METRICS FOR ATTACKERS EXPLOITING UNAUTHORIZED SOFTWARE	63
16.1	LR-2 Checklist	68
16.2	LR-2 Capability Deficit Metric	69
16.3	LR-2 Operational Metric	70
17.	LR-3 METRICS FOR KNOWN VULNERABILITIES	73
17.1	LR-3 Checklist	74

17.2	LR-3 Capability Deficit Metric	74
17.3	LR-3 Operational Metric	76
18.	LR-4 METRICS FOR MISCONFIGURATIONS	79
18.1	LR-4 Checklist	79
18.2	LR-4 Capability Deficit Metric	80
18.3	LR-4 Operational Metric	81
19.	LIMITATIONS	83
20.	DISCUSSION AND CONCLUSIONS	85
21.	BIBLIOGRAPHY	87

**This page intentionally left blank.**

## LIST OF ILLUSTRATIONS

Figure No.		Page
1	The number of serious vulnerabilities on a network is not an accurate risk metric.	5
2	The number of detected incidents in a network is not an accurate risk metric.	6
3	The total duration hosts are vulnerable or the total window of vulnerability is not an accurate risk metric.	7
4	Metrics support a continuous security improvement loop.	11
5	A network should be protected from at least three types of threats ranging from ankle biters to criminal malware to advanced persistent threats. Each threat will use the least sophisticated attack that can be successful, but each type has access to more sophisticated techniques.	12
6	Three stages of a metric maturity model.	13
7	Sequence of events when a defender detects and removes a security condition before it is discovered by an attacker.	19
8	Attackers and defenders scan for exploitable security conditions.	20
9	Graphical summary of notation used to describe sampling intervals and durations.	21
10	Periodic sampling used to detect insecure conditions.	26
11	Probability of detecting a security condition of duration $w$ by periodic scanning with a sampling interval of $\Delta$ .	28
12	Probability of missing a condition of duration $w$ by scanning with a sampling interval of $\delta$ .	29
13	Poisson sampling used to detect insecure conditions.	30
14	Probability of detecting a condition of duration $w$ by Poisson scanning with an average sampling interval of $\lambda$ .	31
15	Pareto sampling used to detect insecure conditions.	32
16	Probability of detecting a condition of duration $w$ by Pareto scanning with an average sampling interval of $m$ and $a = 1.5$ .	34
17	Probability of detecting a condition of duration $w$ by Pareto scanning with an average sampling interval of $m$ and $a = 1.5$ . The x-axis uses a log scale and ranges from 0.1 to 1,000.	34
18	Probability of detecting a condition of duration $w$ by four different types of sampling.	35

19	Probability of missing a condition of duration $w$ by four different types of sampling.	36
20	Event tree for a long-duration insecure condition.	37
21	Joint distribution of the time it takes an attacker and defender to detect a security condition.	38
22	Probability of compromise as a function of defender sampling time.	39
23	Estimating the duration of insecure conditions using periodic sampling.	45
24	Maximum expected error in estimating the probability of attacker detection.	48
25	LR-1 attack model.	56
26	Processing required to create LR-1 metrics.	57
27	Example inventory of authorized and unauthorized devices for LR-1.	58
28	The LR-2 attack model includes server-side attacks (left) and client-side attacks (right).	64
29	Processing required to create LR-2 metrics.	65
30	Device profiles and inventories of authorized and unauthorized software required by LR-2.	66
31	Package classification flowchart for LR-2.	67
32	Process to create and maintain lists of authorized and unauthorized software.	68
33	Combining vulnerability windows for unauthorized client and server software to assess risk.	71
34	Vulnerability detection process for LR-3.	76

## LIST OF TABLES

<b>Table No.</b>		<b>Page</b>
1	Characteristics of Four Lincoln Risk (LR) Metrics	17
2	Summary of the Most Important Notation	22
3	Failure Probability Combination Formulas	42
4	Two Options for Computing Capability Deficit Metrics. The Default is CD-A.	52
5	Three Options for Computing Operational Metrics. The Default is OM-A.	53
6	Characteristics of LR-1 Metrics	55
7	Characteristics of LR-2 Metrics	63
8	Characteristics of LR-3 Metrics	73
9	Characteristics of LR-4 Metrics	79

**This page intentionally left blank.**

## 1. INTRODUCTION

Government, commercial, and institutional computer networks are under constant cyber attack. Recently, high-profile successful attacks have been detected against the International Monetary Fund, Citibank, Lockheed Martin, Google, RSA Security, Sony, and Oak Ridge National Laboratory[13]. These and other attacks have heightened securing networks as a high priority for many organizations, including the U.S. Government. One of the most important strategies to protect networks is to know what types of attacks adversaries employ, to develop metrics to assess susceptibility to each attack type, and then to use metrics to guide addition of controls that are most effective in preventing attacks. Since we are engaged in an arms race with attackers, this strategy can only be effective if defender actions evolve more rapidly than those of the attacker. As such, metrics that accurately assess risk from current attacks must be rapidly developed and deployed. To be efficient and effective, metrics must (1) focus on the most common damaging attacks occurring today and anticipated in the near future, (2) be automated when possible and continuously evaluated, and (3) motivate and quantify security improvements. Driven by these concerns, this report presents a methodology that enables rapid metric development for new threats. This methodology consists of estimating risk by creating an accurate probabilistic model of threats and defenders informed by continuous real-time measurements of important network security conditions. It is illustrated by presenting new metrics for four of the most prevalent modern attack types. These metrics can be computed automatically and continuously, and they estimate the potential damage expressed as the expected number of hosts that can be directly compromised from each attack type.

All metrics assess risk for one attack type over a specific measurement interval. Risk represents the expected number of hosts or infrastructure devices directly compromised by an attacker that provide an initial foothold into a network. If asset values are available, risk can represent the total asset value of the compromised devices. The term “host” or “device” will refer to any network-connected device with an IP address including infrastructure devices such as routers, switches, and firewalls and also user devices such as desktops, laptops, netbooks, smart phones, PDA’s, music players and other devices. We will assume that attackers have the intent and capability to use specific attack types and will initially characterize attackers by the rate at which they scan network devices searching for exploitable security properties. We will use the term “compromise” to refer to a victim failing to perform its function including loss of confidentiality, integrity, or availability.

The remainder of this report reviews past related metric research, describes goals and a metric maturity model used to facilitate metric development and installation, presents desired metric characteristics, reviews common analyses and terminology used across all metrics, analyzes our adversary models, reviews and provides details of four metrics, reviews limitations, and ends with a discussion.

**This page intentionally left blank.**

## 2. LIMITATIONS OF PAST METRICS USED TO ASSESS RISK

Risk analysis is a methodology that requires the following three main steps (e.g., [1]):

1. Define and characterize the threat by specifying capabilities, goals, and possible outcomes resulting from the threat attempting to reach different goals.
2. Characterize the system being protected including vulnerabilities and defenses that are relevant to the specific threat.
3. Analyze the risk using statistical approaches to determine probabilities and the damage for different outcomes and the effect of relevant defenses.

In this section we review past approaches that have been developed to create risk metrics for large enterprise networks and also individual software applications. We find that some metrics that claim to measure risk omit one or more of these steps such as defining the threat. Others are labor intensive or require human judgment and thus can not be automated. Because our goal is to develop systems that provide real-time continuous risk assessment, we end this review by focusing on more recent data-driven approaches that motivate some of our work.

One of the most common past approaches to assess risk is to employ a group of security experts, often called a “red team,” who demonstrate that they can compromise the confidentiality, integrity, or availability of a network or application (e.g., [20]). Although the work factor required by a red team measured in person days or cost may be a useful metric, red teams are used infrequently because they are expensive, not repeatable, and have difficult-to-calibrate skill levels. They also may not comprehensively explore all important threat types.

A second approach is to subjectively measure risk by using subject matter experts who separately assess the potential impact and ease of exploitation for different threats. The overall risk is found by either multiplying the impact times the ease-of-exploit (often incorrectly called likelihood) or using a “risk matrix” that provides the risk for all combinations of impact and ease-of-exploit. This is the approach recommended by NIST [44], by the early security consulting firm @stake [17], and by many modern risk assessment approaches [15, 17]. The Mission Oriented Risk and Design Analysis (MORDA) methodology is another important example of a subjective approach that relies on subject matter experts to suggest attacks and defenses for a system being designed [4]. It combines adversary preference scores for various attacks, scores rating the effectiveness of a system with various countermeasures, and overall system cost together to assess the risk of alternative system designs that employ different countermeasures.

Two more recent subjective approaches have been created to assess the risk to software applications from a library of well-defined threats [11, 14]. The threat modeling approach of [11] begins by creating an annotated dataflow diagram that shows external entities, processes, and dataflow interactions. Each dataflow interaction is then compared to a library of 35 threats to determine if any threats apply to that interaction. The threat library includes entries such as “Access using default credentials,” “Network sniffing,” and “SQL Injection.” After a user fills in answers to 10

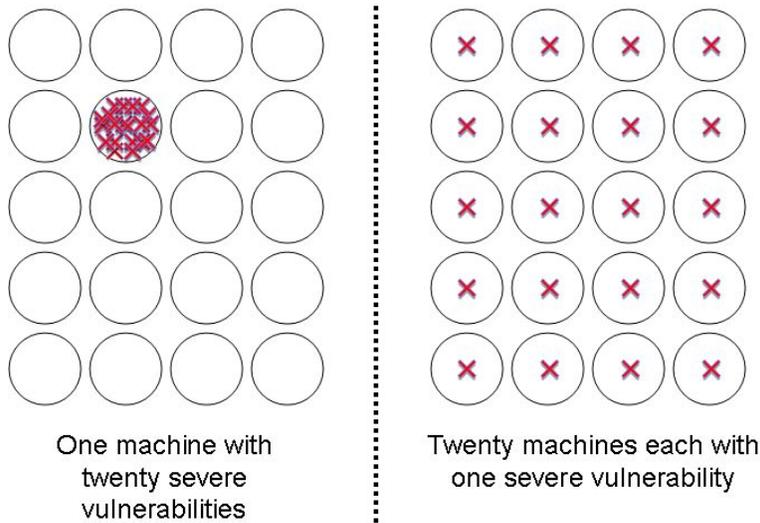
questions concerning the impact and ease of creating and executing an exploit corresponding to a threat, a risk score is created using a modified version of the Common Vulnerability Scoring System [23]. Risk scores are created for all relevant threats and interactions and sorted to address the most serious threats first.

All the above subjective approaches share common weaknesses. First, they are often highly subjective, labor intensive, and yield results that may vary dramatically across different subject matter experts. Results are often not comprehensive or explanatory and these approaches can not be automated and used for continuous security monitoring.

More recently, the Security Content Automation Program (SCAP) [34] was developed by the National Institute of Standards and Technology (NIST) and others to support data-driven risk assessment. It supports the National Vulnerability Database (NVD) [28] that provides a repository for known vulnerabilities and software that contains these vulnerabilities. The NVD is often used to associate vulnerabilities with software. SCAP includes the Common Vulnerability Scoring System (CVSS) [23]. This provides a score for each new software vulnerability discovered that prioritizes the importance of this vulnerability. It is created based on a few questions answered by a subject matter expert. Although the base CVSS score is not necessarily a risk, it is computed by adding a score that represents the ease of exploiting a vulnerability to another score that represents the impact of exploiting the vulnerability. CVSS scores and the common dictionary provided by the Common Vulnerabilities and Exposures (CVE) list [22] have been used extensively in data-driven risk assessment to both enumerate vulnerabilities and assess their importance.

SCAP components have enabled many modern data-driven risk assessment approaches for enterprise networks. The overwhelming majority of these risk metrics are simply counts or percentages that measure compliance to a policy or aspects of a network related to security. Representative examples of recently recommended metrics of this type are the number of open ports reachable from the Internet [3] and the number of missing patches, firewall rule changes, and vulnerabilities over a given interval [17]. Examples of counts provided by commercial tools include the number of high-severity vulnerabilities found by network vulnerability scanners (e.g., [40]) and the numbers or percentages of hosts that are not patched to a desired level (e.g., [12]). A recent comprehensive list of metrics for enterprise networks provided by the Center for Internet Security [5] is also dominated by percentages based on counts. For example, three key metrics in the proposed scorecard [5] are the percentage of systems with no known severe vulnerabilities, the percentage of systems that are compliant with the current patch policy, and the percentage of systems that are compliant with the current configuration policy. Other metrics also assess the percentage of systems that are covered or included in these metrics.

A more comprehensive example of how SCAP enables continuous security monitoring in real time is the U.S. Department of State's iPost system [15]. Metrics are gathered using commercial tools and include items such as the number of vulnerabilities on a host, the number of missing software patches per host, the number of account password ages above a threshold, the number of hosts not reporting to management tools, and the number of hosts not included in vulnerability scans. These per-host measures are combined across sites and across the entire enterprise to compute



*Figure 1. The number of serious vulnerabilities on a network is not an accurate risk metric.*

site and enterprise security summaries including a single score and a grade from “A” to “F.” Metrics are designed to motivate security improvements by computing scores consistently and fairly and by making scores visible to all. As this system was deployed and used, it was found that fairness was important for acceptance. Fairness included not penalizing a system administrator for failing to install a software upgrade that has not yet been approved or that is not approved because it disables critical functionality. It also includes not assuming a security condition fails a test because a test is not successfully completed, but waiting until a test has completed successfully and only then using the PASS or FAIL result of the test.

Risk is not measured by either the iPost metrics or any of the other metrics described above that claim to assess risk using counts or percentages. Although these counts and percentages measure network properties that are related to risk, they do not assess risk. As noted above, determining risk requires three major steps not provided by count and percentage metrics. In particular, counts and percentages do not define the threat, specify the probability of different outcomes, or indicate how probabilities change when defenses change. Without an explicit threat model that provides an answer to the question “Secure against what?,” counts and percentages are difficult to interpret. For example, it is impossible to know whether specific threats are prevented, to determine expected number of hosts that may be compromised for a specific threat type, or to determine whether different defensive strategies such as patching known vulnerabilities more rapidly might reduce risk significantly.

One of the difficulties in interpreting counts as a risk metric is illustrated in Figure 1. Consider a situation where the metric used is the total number of software packages on a network with severe vulnerabilities as indicated by a CVSS score of 10. As shown in Figure 1, this could represent a network where one host contains all the vulnerabilities as shown on the left or where the vulnerabilities are spread across all hosts as shown on the right. If the threat model of concern

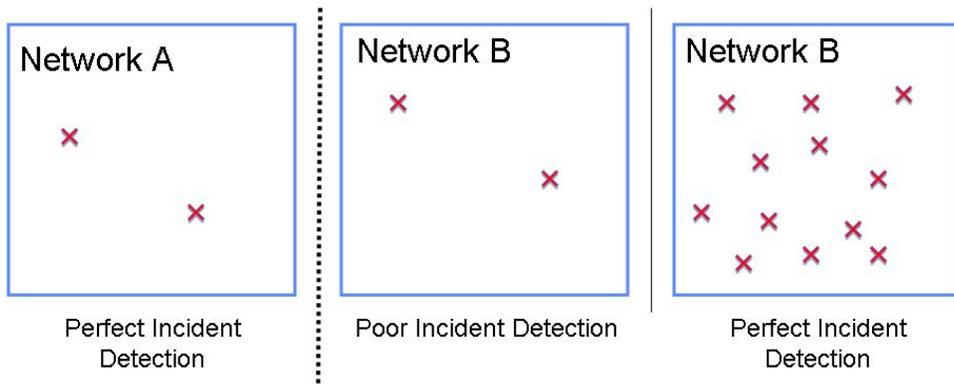
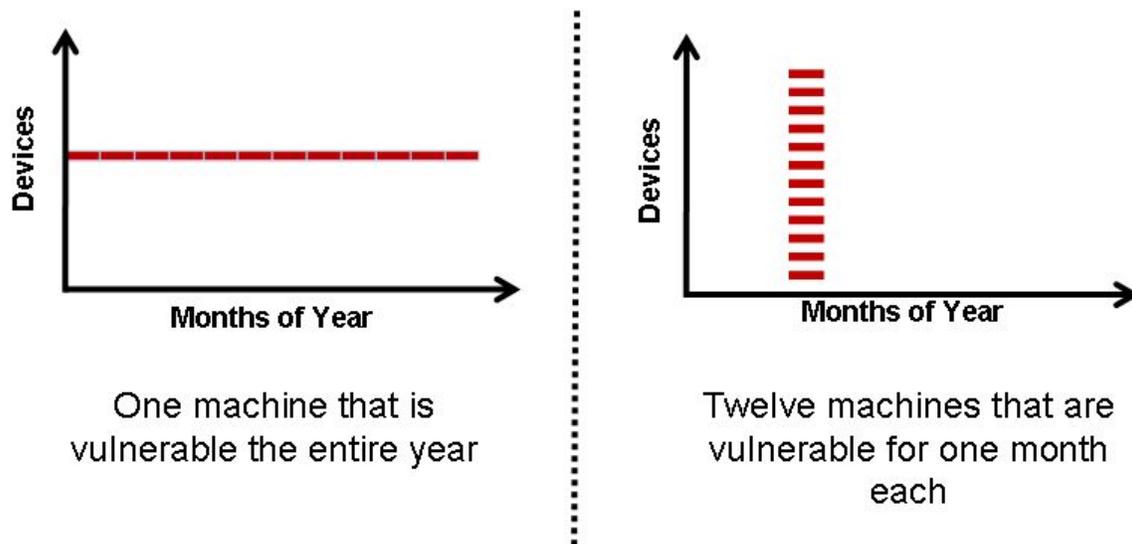


Figure 2. The number of detected incidents in a network is not an accurate risk metric.

was a rapidly scanning worm on the inside of this network attacking all severe vulnerabilities, and these vulnerabilities were all on web or database servers, risk could vary dramatically for these two situations. The expected number of hosts compromised by a rapidly scanning worm with access to the network could be one for the left network and 20 for the right network, even though they have the same number of total vulnerabilities. A second major problem with this simple count is that the importance or asset value of hosts in a network could differ substantially. For example, one host could be an administrative host that would enable compromise of all other hosts in the network, another might be a database server containing highly confidential material, and others might be simple user workstations with web servers left on by default. As noted above, accurate risk assessment must include the damage caused by the threat and this includes the host-specific asset values obtained by the threat. Instead of the expected number of hosts compromised, a more accurate risk metric would be the expected asset value compromised. This can be used whenever asset values are available.

Figure 2 illustrates another problem with using counts as metric. In this figure, we assume there is a mechanism to detect the number of incidents or hosts that have been compromised by an attacker in a network. Incidents could be detected by anti-virus tools, host-based firewalls, or known signatures in network traffic detected by an intrusion detection system when malware signals back to command and control hosts. Here we assume the number of incidents is the metric used to assess risk. The far left panel in Figure 2 represents a network (network A) where all incidents are detected and there are two incidents. The middle panel represents a second network (network B) with poor incident detection where only two incidents are detected. Without further information, it would appear that the risk of compromise is the same in networks A and B. The truth for network B is shown in the far right panel where it can be seen that with perfect incident detection, 12 incidents are detected. The risk as measured by the number of hosts compromised is thus six times as great in network B as in A, but using incident counts incorrectly indicates that the risk is identical. Another problem with using incident counts is that the result is not diagnostic because the number of attempted but failed attacks is usually unknown. One network could experience few successful attacks because there are few attempted attacks but poor defenses while another



One machine that is vulnerable the entire year

Twelve machines that are vulnerable for one month each

Figure 3. The total duration hosts are vulnerable or the total window of vulnerability is not an accurate risk metric.

network may experience few successful attacks because there are many more attempted attacks but strong defenses. Overall, the number of incidents is difficult to interpret as risk because it is usually impossible to normalize for both threat intensity and incident detection rate.

Figure 3 illustrates a final problem with using simple counts as a risk metric. Here we are interested in windows of vulnerability or the time intervals when hosts have vulnerabilities and can be successfully compromised by the appropriate threat. We assume the metric used is the total of all windows of vulnerability across all hosts measured in vulnerability months. Windows of vulnerability are accumulated across all hosts over a year to create this metric. The diagram on the left of Figure 3 shows a network where only one host is vulnerable, but it is vulnerable all year. The total window of vulnerability is 12 host months for this network. The diagram on the right of Figure 3 shows a network where all 12 hosts in the network are vulnerable but only for the same month. The total window of vulnerability for this network is also 12 host months. The risk from a specific threat for these two networks depends on the threat. For a threat that attempts to exploit these known vulnerabilities once a month all year long, the expected number of hosts compromised is 1 for the network on the left and 12 for the network on the right. For a threat that is present during only the last month of the year, the expected number of hosts compromised is one for the network on the left and zero for the network on the right. The actual risk depends on the temporal details of windows of vulnerabilities and the threat. This can not be determined by a simple overall sum of windows of vulnerabilities.

**This page intentionally left blank.**

### 3. RISK ASSESSMENT AS A SCIENTIFIC METHOD

Many of the above described approaches to risk assessment seem unscientific, poorly justified, and ad hoc. Our goal, however, is to develop a scientific method to risk assessment as discussed in a recent paper [43]. This paper presents some metrics that suggest how the security research field might become a science, but most of these metrics assess the computational complexity or utility of a specific attack or defense component but not the overall risk across network devices. A good discussion of requirements necessary for risk assessment to be considered a scientific method is provided in [1]. The following requirements for scientific risk assessment are adapted from these requirements:

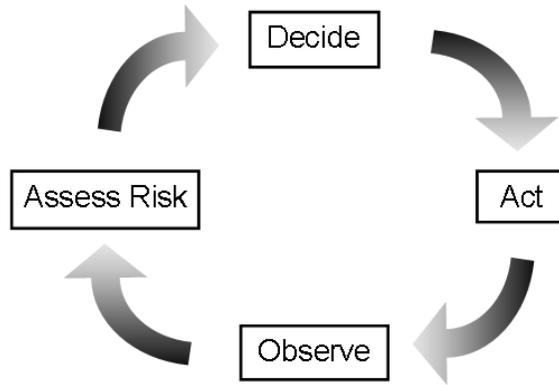
1. The mathematical approaches used to model threats, the overall system, and defense models are clearly stated and agree to the extent possible with known threat, system, and defense characteristics. Assumptions, simplifications, limitations, and constraints are clearly defined to make sure they can be reviewed for accuracy and the results can be replicated.
2. The analysis focuses on determining the risk or probability of different outcomes for different threats and with different defenses.
3. The analysis and results are *reliable*, meaning that a different group of researchers making the same assumptions would obtain similar results, and they are *valid*, meaning that they estimate the true underlying risk.

Our risk metric development discussed below is designed to satisfy these requirements. A key component of our scientific approach is threat modeling. Our initial approach to developing threat models is to take advantage of recent collaboration across security organizations that led to a document entitled “Twenty Critical Security Controls for Effective Cyber Defense: Consensus Audit Guidelines” [38]. This document identifies the twenty most important current cyber threats and also the critical security controls that protect against these threats. The four metrics we describe focus on four important foundational critical controls in [38] and the attacks these are designed to detect. We have refined and extended attack models and used them and the controls to create mathematical models of threats and defenses that are used to predict risk. Each new metric is based on a realistic and well-defined adversary model, directly measures the effect of controls that mitigate adversaries, continuously estimates risk, and provides direct insight into what network changes must be made to improve security. System, defense, and vulnerability descriptions required for metric development are also enabled by many SCAP components and the NVD National Vulnerability Database.

**This page intentionally left blank.**

## 4. GOALS

Our goal is to develop metrics that motivate the continuous security improvement loop shown in Figure 4. Important network characteristics such as active devices, operating system configurations, installed applications, and vulnerabilities are measured continuously at the bottom of this loop and are used to compute metrics at the left that estimate the risk for a specific attack type. Per-device metrics are used to determine the most efficient actions that reduce risk as shown on the top and actions taken such as removing unauthorized devices, removing unauthorized applications, and patching lead to more secure networks over time. This continuous process requires metrics that can be computed rapidly, continuously, and automatically using existing security tools.



*Figure 4. Metrics support a continuous security improvement loop.*

Each metric in this process assesses the risk from a specific threat and, if the feedback process is successful, the risk from that threat will be reduced. Figure 5 illustrates the three major types of threats that networks should be protected against. This notional chart indicates the range of sophistication of these threats. The simplest “ankle biter” threats are frequent and use attacks with little sophistication that are well known. It is usually easy to detect and protect a network from these attacks. Organized cyber criminals use more sophisticated malware with the goal of making money illicitly. This malware must constantly change to maintain success in compromising hosts and avoiding defensive measures. Compromised hosts may be used for many purposes including gathering personal information, sending spam, capturing online banking information, and launching distributed denial of service attacks. Criminal threats are carefully monitored by security companies that can provide tools for protection and detection. Nation states and other groups with sufficient funding can create the most sophisticated advanced persistent threats (e.g., [36]). It can be very difficult to detect and protect a network from these threats. As shown in Figure 5, each of the three threats has access to both unsophisticated attacks and ever more sophisticated threats. They will

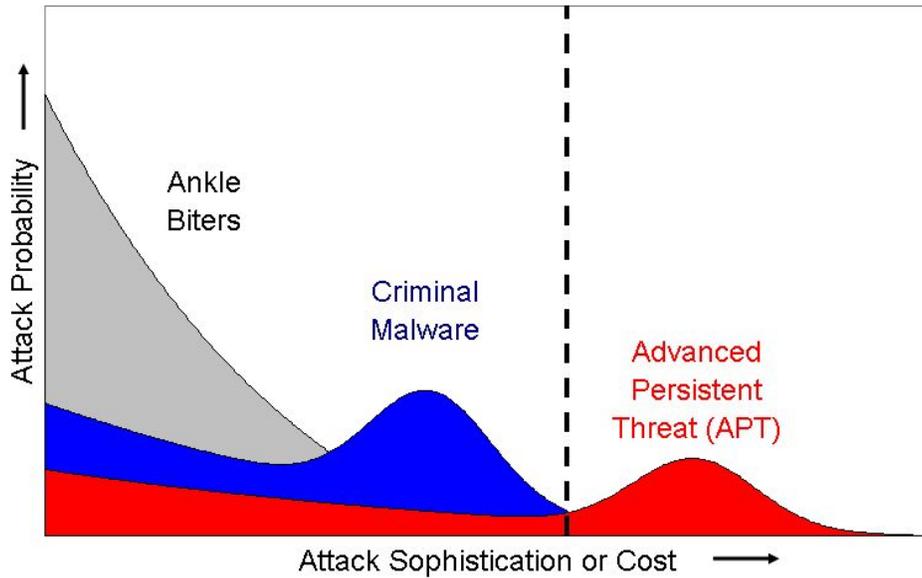


Figure 5. A network should be protected from at least three types of threats ranging from ankle biters to criminal malware to advanced persistent threats. Each threat will use the least sophisticated attack that can be successful, but each type has access to more sophisticated techniques.

use the simplest threat necessary to penetrate the defenses in a target network without exposing more sophisticated approaches to discovery and analysis that could lead to development of effective protection.

Our goal is to develop metrics for the most important known threats including attacks from ankle biters, organized crime, and some known advanced persistent threats. If these metrics are driven down to low enough values using defensive controls, a network will be protected from important known threats. This is indicated by the dashed line in Figure 5. Effective metrics can protect against the known threats to the left of this dotted line. They may not protect against unknown and extremely sophisticated advanced persistent threats to the right of this dotted line. They will, however, increase the difficulty of launching these advanced attacks, potentially make it easier to detect them, and force the adversary to use ever more sophisticated and costly attacks.

## 5. A METRIC MATURITY MODEL

The metric maturity model shown in Figure 6 is an essential foundation of our approach that supports gradual introduction and use of security metric components across an enterprise. The notional curve in this figure shows how security improves over time during three maturity stages.

The first checklist stage provides a foundation necessary to compute metrics. During this phase, system administrators develop an understanding of their system and the most important threats. They begin to implement processes, add tools to gather data, and develop controls. While this stage does not directly implement security measures, we posit that it does have a significant security impact. The slight improvement in security shown by the curve in the left of Figure 6 is presumed to be due to improvements in general network hygiene and the fact that improved understanding informs existing security processes. The large improvement towards the end of this phase might be caused by discovery of previously unknown network structure made possible by improved hygiene, new security tools, and by the repair of non-functional security tools, processes, and controls.

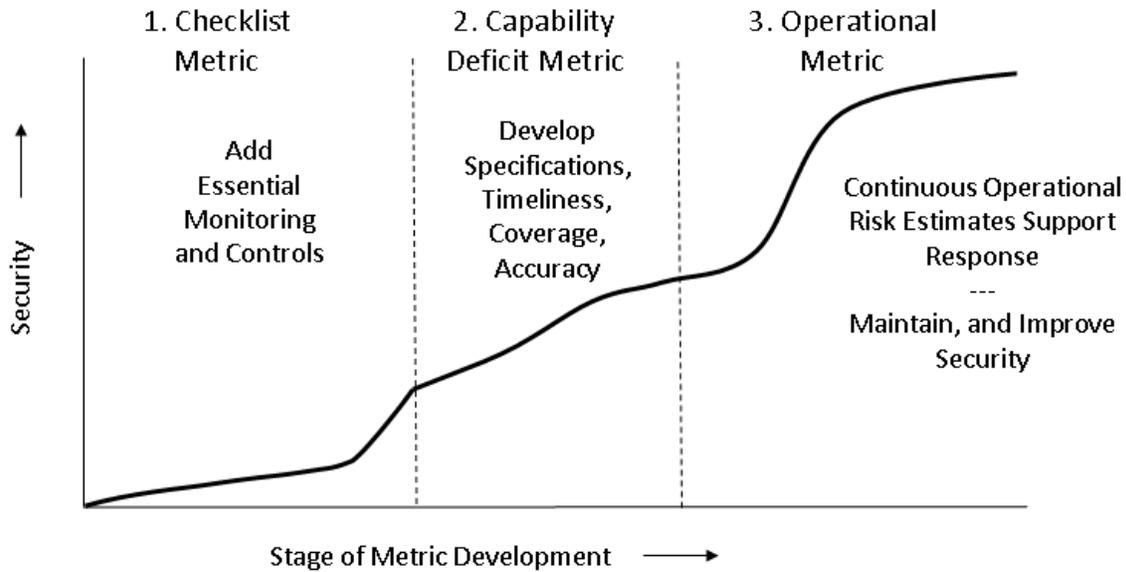


Figure 6. Three stages of a metric maturity model.

The second capability stage of the maturing process focuses on generating specifications of security properties and improving the timeliness, coverage, and accuracy of measurements used to compute risk in the third stage. Capability deficit metrics implemented in this stage measure the gap between the actual capability and a parametrized “gold standard” capability. Until values of these metrics are close enough to the “gold standard,” risk cannot be computed accurately. Security improves slowly during this stage due to further discovery and repair of security issues as coverage

improves across the entire network, specifications are developed, rapid sampling discovers previously missed short-duration security conditions, and security properties are measured accurately.

Operational metrics in the third maturity stage are the key component of the OODA-like [30] improvement loop described in Figure 4. They lead to continuous security improvements by assessing risk and indicating which devices, software packages, misconfigurations, or other network characteristics are responsible for the greatest increase in risk. In our current metrics, we assume that a threat is always present and attackers continuously scan for exploitable insecure conditions with a given sampling interval. Risk is thus the expected impact of the threat. This is normally the expected number of hosts compromised, or if asset values for hosts are available, it can be the total asset values of the compromised hosts. Assessing risk with operational metrics makes it possible to act on the metrics and modify a network to reduce risk. Operational metrics lead to a large improvement in security when followed by a response that modifies the network security properties in a way that reduces risk. They can only be determined after capability deficit metrics are low enough to guarantee that the operational metrics are computed accurately. This is why they follow capability deficit metrics in the metric maturity model.

After designing this security maturity model, we found that it is supported by, and in some ways parallel to, the the four step “Visible OPS” maturity process described in [18] that is designed to improve overall IT operations. The Visible OPS approach includes the following steps: 1) stabilize the patient and get plugged into production, 2) find business risks and fix fragile artifacts, 3) implement development and release controls, and 4) continual improvement. Networks where the Visible OPS process is followed will have good situation awareness, a good understanding of devices and software on networks, procedures in place for upgrades and change management, an understanding of asset values of devices and which devices and software packages are most critical, good visibility across the IT infrastructure, and a well-trained IT workforce. Although the Visible OPS process does not define continuous security metrics, it will be much easier to implement our metrics in a network where the Visible OPS process is followed.

## 6. DESIRED METRIC CHARACTERISTICS

We have already described key characteristics of a scientific approach to risk assessment. These include carefully defining assumptions and models, developing risk metrics that estimate the probability of various outcomes, and making sure metrics are accurate and the analysis is correct and consistent with the data and assumptions. In addition to these principles, we established the following three key additional guidelines for risk metric development:

1. Each metric must be simple to understand and practical to implement.
2. Each metric must accurately estimate the risk from one specific important threat.
3. Metrics must motivate actions to reduce the risk from threats.

Maintaining practicality and simplicity are common goals. Practicality includes using data that can be gathered with existing security tools and simplicity includes modeling mainly primary effects that can be predicted accurately with simple models. Accurately estimating the risk from one threat at a time allows us to focus on important threats and create additional detailed threat models and metrics as necessary without having to model all threats at once. Motivating defenders to make the right decisions to improve security, however, is more unusual guideline and is the focus of this section.

As noted in [37], motivational metrics must be objective, computed fairly, and be visible to all involved in the security process to allow comparisons over hosts, subnets, and enterprises and over time. As in [37] we have adopted the convention that high scores are bad and low scores (near zero) are good. If scores are normalized to range from 0 to 100 and high scores are good, it was found in the past [27] that system administrators stopped trying to improve scores once they reached 80 or 90 because these represent good test scores in an academic setting. Alternatively they continued to try to reduce scores to zero after reaching 10 or 20 if low scores are best.

One of the most difficult motivating metric characteristic to achieve is that incremental improvement in controls must lead to incremental improvements in the metric. If this doesn't occur, then there will be little positive incentive to add controls. When multiple controls contribute to overall security, their settings should be combined in a way that motivates addition of controls by giving credit to incremental improvements. This goal often needs to be traded off against the desire for accurate statistical defense models. An example of the application of this goal is providing credit for creating a specification that enables a security test when computing the capability deficit metric, even if the test is not yet implemented.

Another motivational feature is fairness. To be fair, a metric should not penalize a system administrator for a centrally controlled router with vulnerabilities if that system administrator has no control over that router. A metric should also not penalize a system administrator for a legacy system if that system is essential and it has been determined that the risk posed is acceptable. It should also be easy to determine major security flaws that create poor (high) metric scores. For

example, devices with the worst individual scores should be easy to identify and the reason for these high scores should be evident.

A final motivational feature is that the overall difficulty of obtaining a good low metric score value should increase over time. Initially it should be relatively easy to get a good low score, but this should become more difficult as capabilities, controls, and responses mature. This property is enabled by our metrics via parameters that can be adjusted over time to raise the standard of capability and operation required for good metric scores.

## 7. OVERVIEW OF FIRST FOUR METRICS

We will demonstrate how to develop risk metrics for any well-defined threat and system. As we develop new risk metrics, they will be labeled using the prefix “LR” followed by a dash and a number such as LR-1 for the first metric. The LR prefix stands for “Lincoln Risk” metric. Initially, we focus on the most important threats selected by the authors of the “20 Critical Controls” document [38]. This document describes 15 critical controls and associated attacks that are amenable to continuous monitoring and 5 that are not. We selected the threats from an initial 5 of these 15 (1, 2, 3, 4, and 10) to create foundational risk metrics to develop and denote our metrics as LR-1 through LR-4. Likewise, we denote the critical controls from [38] using the prefix CC followed by a dash and the number of the critical control.

TABLE 1

Characteristics of Four Lincoln Risk (LR) Metrics

LR	CC	Entity with security condition	Security condition exploited by attacker	Specification that enables security condition test	Data required by operational metric
1	1	Subnet	Unauthorized computer systems and laptops	Authorized devices on each subnet	List of first and last seen times of unauthorized devices
2	2	Device	Unauthorized software	Unauthorized and authorized software on each device	List of first and last seen times for each unauthorized software package on each device
3	10	Device	Known software vulnerabilities	List of known vulnerabilities on each device	List of first and last seen times and score for each vulnerability on each device
4	3,4	Devices	Misconfigurations	List of correct configurations on each device	List of first and last seen times and score for each misconfiguration on each device

The first two columns of Table 1 show the LR number followed by the corresponding number of the critical control (CC) in [38] whose threat is modeled by the LR metric. It can be seen that the

LR and CC numbers for the first two rows in this table correspond. For example, LR-1 models risk from the threat for CC-1 and LR-2 models risk from the threat for CC-2. This differs for the next two rows where LR-3 corresponds to CC-10 and LR-4 corresponds to CC-3 and CC-4. The fourth column of Table 1 indicates the insecure condition that is exploited by the threat being modeled. For example, LR-1 estimates the risk from an attacker who compromises unauthorized devices on a network such as test servers, personal laptops, or smart phones. It corresponds to the first critical control that is entitled “Inventory of Authorized and Unauthorized Devices.” Implementing the first two metrics (LR-1 and LR-2) provides general network situation awareness required for other metrics. LR-1 monitors unauthorized devices and its implementation provides a list of all devices on a network. LR-2 monitors all software packages and its specification provides a list of all software packages used in a network. The third and fourth metrics (LR-3 and LR-4) were selected because counts of vulnerabilities and misconfigurations are common, well supported by SCAP, and are often currently available at secure networks. These two metrics also illustrate how to combine multiple misconfigurations and vulnerabilities on one device and how to fuse data gathered from network scanners and host-based agents.

Table 1 describes further details of the first four metrics. All metrics are associated with exploitable insecure conditions that can occur at any time. If an attacker detects an insecure condition first after it occurs, a network device such as a host or laptop may be compromised. If a defender detects an insecure condition first, and removes it fast enough, the device is protected. The fourth column in Table 1 lists the exploitable security condition corresponding to each metric and the third column indicates the network entity that contains this condition. For LR-1, attackers look for unauthorized devices on each subnet. For LR-2, attackers look for unauthorized software on all devices. For LR-3, attackers look for known vulnerabilities on all software packages. Finally, for LR-4, attackers look for software misconfigurations. The fifth column in this table lists the specifications that are required to determine if an exploitable security condition is present. A major component of the second maturity stage of metric development is creating these specifications. As can be seen, sometimes they include lists of allowable devices, authorized and unauthorized software, known software vulnerabilities, and configuration checks. The final column in Table 1 describes the data required to compute the expected number of hosts compromised for the attack model associated with each metric. This is always a list containing each device with an exploitable insecure condition, the window of vulnerability (start and end times) of the condition, and some measure of the severity of the exploitable condition.

## 8. ADVERSARIAL INTERACTION BETWEEN ATTACKERS AND DEFENDERS

As noted above, all four metrics are associated with exploitable insecure conditions that can occur at any time. If a defender detects an insecure condition first, and eliminates it fast enough, the device is protected. If an attacker detects it first, a network device may be compromised. Successful defense is illustrated in Figure 7. The horizontal axis in this figure represent time starting when the insecure condition begins. For LR-1, this corresponds to an unauthorized device being placed on a network. The solid bar labeled “Window of Opportunity for Attack” marks the time interval that the insecure condition could be exploited by an adversary to compromise a host. Note that we assume the security condition is exploitable even while being processed by the defender. In this example, the defender detects the security condition and responds fast enough to remove it before it is detected by the adversary. If the attacker had detected the insecure condition before it was removed, it would have enabled an attack.

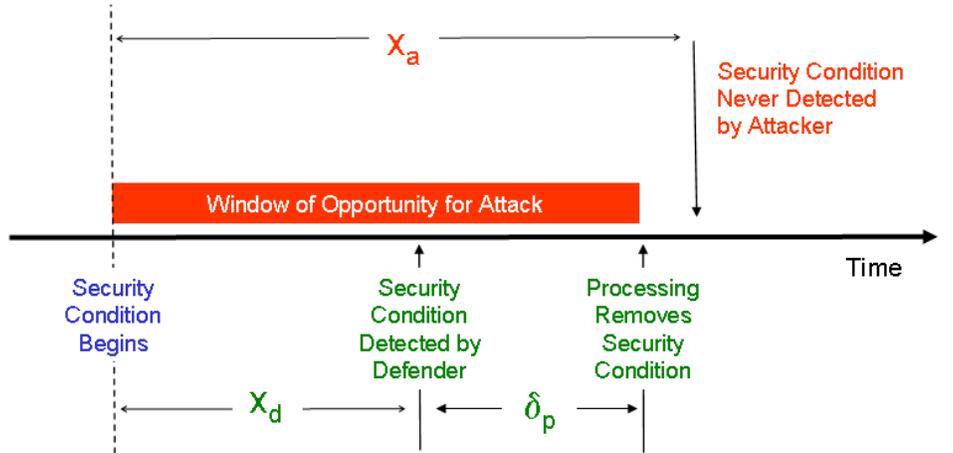


Figure 7. Sequence of events when a defender detects and removes a security condition before it is discovered by an attacker.

Figure 8 illustrates the interaction between defenders searching for and eliminating vulnerable conditions and attackers also searching for vulnerable conditions and using them to exploit devices. The vertical axis in this figure represents an enumeration of all possible insecure conditions in a network that could be exploited by one attack type. For LR-1, this would be the available IP address space. The horizontal axis represents time with vertical lines at weekly intervals. Each thick horizontal gray bar represents an insecure condition that begins and lasts for a specific duration. For LR-1, each bar represents an unauthorized device with an IP address that was put on the network. We normally assume defenders and attackers do not know when security conditions will begin and how long they will last.

The steeply slanted lines in Figure 8 represent sequential continuous periodic defender scans over all security conditions that are searching for insecure conditions. When a defender finds an insecure condition, it is processed. Processing can eliminate the insecure condition by adding controls or removing the entity with the vulnerability from the network or it can determine that the test used to detect the insecure condition was incorrect and the insecurity does not exist. This processing is indicated by making the horizontal bars darker. For LR-1, this corresponds to removing an unauthorized device from the network, adding patches and other security controls required to authorize the device, or analyzing the device and determining that it should be authorized. This defender processing occurs rapidly in Figure 8 as shown by terminating security conditions soon after they are detected by the defender.

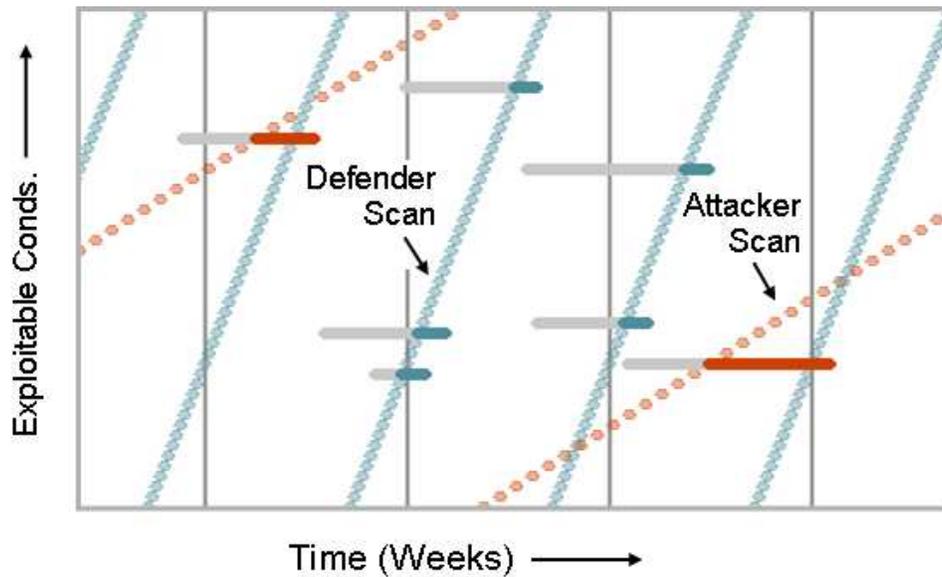


Figure 8. Attackers and defenders scan for exploitable security conditions.

The less steep horizontal dotted lines in Figure 8 represent sequential continuous attacker scans looking to find and exploit insecure conditions. Two insecure conditions are detected by the attacker and exploited in this figure as shown by the bars that become dark after attacker detection on the upper left and bottom right. For simplicity, we assume that a device is compromised as soon as an exploitable security condition is detected by an attacker. As noted above, we also assume that a device can be compromised as long as it is on the network and still assessed to be insecure, even if it has been detected by a defender who is processing and analyzing the device.

## 9. NOTATIONAL CONVENTIONS AND INTRODUCTION

In this section we introduce some notational conventions and describe important notation that is used in the introduction and the metric descriptions. Additional notation is introduced in the metric descriptions. The most important temporal notation that describes durations and sampling intervals is summarized in Figure 9 and described below. Additional notation used when computing metrics is also described below and both temporal and metric notation is summarized in Table 2.

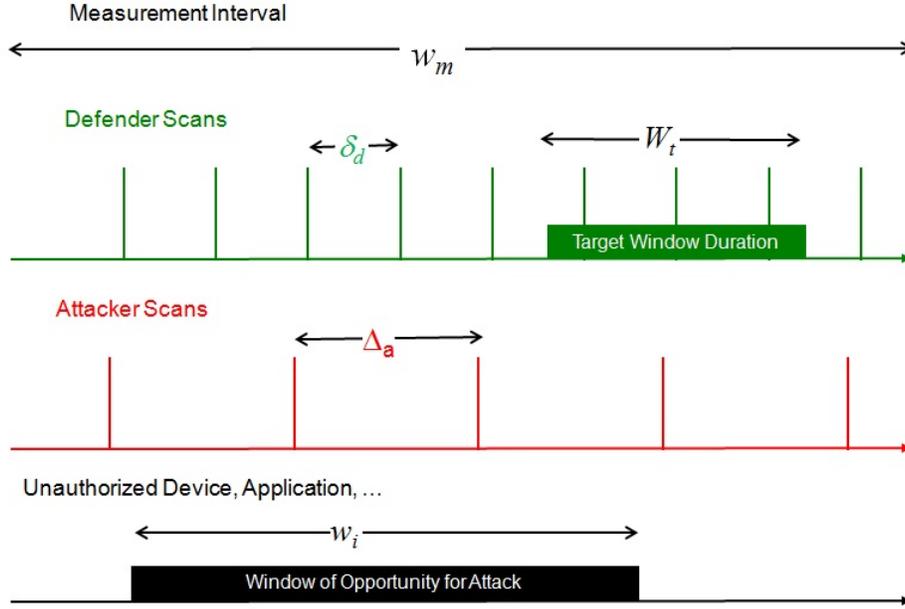


Figure 9. Graphical summary of notation used to describe sampling intervals and durations.

As noted above, we have adopted the convention that high scores represent poor security and low scores (near zero) represent good security. Capability deficit metrics are designed to range from 0 to 1. Low values near 0 are good and high values near 1 are bad. These scores can be normalized to range from 0 to 100 by multiplying by 100 for presentation to security analysts and administrators. Operational metrics normally represent risk or the expected number of devices compromised. If asset values are available for devices, they can represent the total assets compromised. Operational metrics can also be normalized to range from 0 to 100, when desired by taking into account the total number of devices or assets and other network characteristics.

Component functions used to compute operational and capability deficit metrics are also designed to follow the convention that high values represent poor security. For example, we derive formulas that compute the probability of an attacker detecting an insecure condition  $P_{\text{detect}}$  and the probability that a defender misses an insecure condition  $P_{\text{miss}}$ . High values for both of these

**TABLE 2**

**Summary of the Most Important Notation**

Term	Meaning
$W_m$	Measurement window or time span over which metrics are computed.
$\delta$ or $\delta_d$	Interval between defender scans that test for a specific security condition.
$\Delta$ or $\Delta_a$	Interval between attacker scans that test for a specific security condition.
$w_i$	Window of opportunity or duration that insecure condition $i$ is present.
$W_t$	Target window or minimum duration of an insecure condition that a defender should always detect.
$P_{\text{detect}}$	Probability that an attacker detects an insecure condition.
$P_{\text{miss}}$	Probability that a defender misses an insecure condition.
$CDM_i$	Capability Deficit Metric for Lincoln Risk Metric number $i$ .
$SD$	Specification Deficit component of a Capability Deficit Metric.
$TD$	Timeliness Deficit component of a Capability Deficit Metric.
$CD$	Coverage Deficit component of a Capability Deficit Metric.
$AD$	Overall Accuracy Deficit used to compute a Capability Deficit Metric.
$OM_i$	Operational Metric or expected devices compromised for Lincoln Risk metric number $i$ .
$P_{\text{comp}}$	Probability a device is compromised given that an insecure condition is detected on it by an attacker and the attacker launches an exploit.
$S$	Terms that weight importance begin with $S$ .

functions represent poor security and low values represent good security. Capability Deficit Metrics are indicated by  $CDM_i$ , where  $i$  is the LR number from Table 1. Capability deficit metrics are computed using measures of the Specification Deficit  $SD$ , the Timeliness Deficit  $TD$ , the Coverage Deficit  $CD$ , and the overall Accuracy Deficit  $AD$ . Operational metrics are indicated by  $OM_i$ , where  $i$  is the LR number from Table 1. In addition, terms that begin with  $S$  are used to weight importance.

We use capital letters and symbols such as  $W$  and  $\Delta$  to represent quantities that are, at least initially, defined and are free parameters. We use lower case letters and symbols such as  $w$  and  $\delta$  to represent quantities that are measured and known.

As shown in Figure 9, the symbol delta ( $\delta$ ,  $\Delta$ ) is used to indicate the interval between sampling times for periodic sampling or the average sampling interval for non-periodic sampling. The symbol  $\delta$  or  $\delta_d$  is used for defender sampling and  $\Delta$  or  $\Delta_a$  is used for attacker sampling. The letter  $w$  is used to represent opportunity windows or durations of different types of insecure conditions. For example,  $w_i$  is the window of opportunity for an attacker or the duration that insecure condition  $i$  is

present. In addition,  $W_t$  is the target duration or the minimum duration of an insecure condition that a defender should always detect and  $W_m$  is the measurement interval over which metrics are computed.

**This page intentionally left blank.**

## 10. MODELING DEFENDER AND ATTACKER SCANNING

Modeling the adversarial interaction between defenders and attackers requires a mathematical model of how attackers and defenders scan for insecure conditions. The initial insecure conditions scanned for are shown in Table 1 and include unauthorized devices, unauthorized software, software misconfigurations, and software vulnerabilities. The following sections describe how to model four approaches to scanning for insecure conditions. These are 1) instantaneous scanning where an insecure condition is discovered immediately, 2) periodic scanning where every security condition is examined periodically with a given sampling interval, 3) Poisson sampling where the sampling interval has a Poisson distribution with a given mean value, and 4) Pareto sampling where the sampling interval has a long-tailed Pareto distribution with a given mean. These span a wide range of sampling strategies that might be used by attackers or defenders. In each section we describe what type of adversary or defender this type of sampling might represent. We also derive two equations. One computes the probability that an attacker detects an insecure condition with a duration or window of vulnerability  $w$  when using an average sampling interval of  $\Delta$ . The other computes the probability that a defender misses an insecure condition of duration  $w$  when using an average sampling interval of  $\delta$ . These formulas and this analysis forms a foundation that is used to develop both capability deficit and operational metrics. Following the analysis of each of the four sampling regimes, a summary contrasts and compares them.

### 10.1 INSTANTANEOUS SCANNING

Instantaneous scanning occurs when a security condition is detected immediately as soon as it occurs. For example, for LR-1, a defender could detect an unauthorized device as soon as it is attached to a network if switches implement Network Access Control (NAC) approaches (e.g., [41]) that allow only specific devices to be attached to a network. New unauthorized software could also be detected immediately for LR-2 by a defender if software installation and maintenance was centralized and users were not allowed to install software. On the attacker side, new devices could be detected immediately on a local area network for LR-1 if malware installed on a local host monitors ARP or DHCP traffic to detect a recently-connected device being assigned an IP address. This type of detection isn't actually scanning, but is event-driven. With instantaneous sampling, we will normally assume that the probability of an attacker detecting an insecure condition is 1 and the probability of a defender missing a security condition is 0 for all security condition durations.

### 10.2 PERIODIC SCANNING

Periodic scanning is illustrated in Figure 10. The horizontal axis in this figure represents time, and there is a vertical line or impulse whenever a specific security condition is sampled. As can be seen, sampling occurs periodically every 10 weeks. Periodic sampling occurs when attackers or defenders sample each potentially insecure condition periodically with a fixed sampling interval and an initial sampling time that can be considered uniformly distributed over the measurement

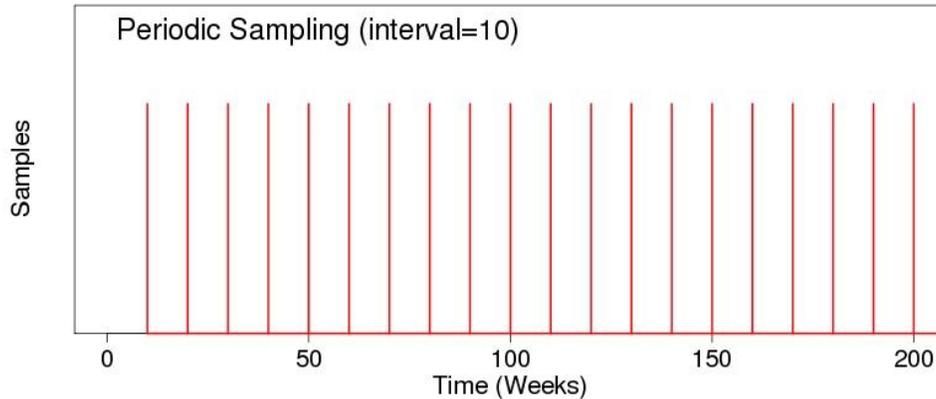


Figure 10. Periodic sampling used to detect insecure conditions.

interval compared to the onset or offset of insecure conditions. Figure 8 illustrates attacker and defender interactions where both use periodic scanning. For periodic sampling, the attacker sampling interval is initially unknown, so we assume a given sampling interval  $\Delta_a$  that can be used to vary the difficulty of obtaining a good low operational metric score. A small sampling interval represents a fast aggressive attacker and a large sampling interval represents a slower more stealthy attacker. The defender scan interval is usually obtained from cooperative defenders and is known. It is denoted by  $\delta_d$  and both attacker and defender scan intervals are measured in seconds. Although Figure 8 illustrates sequential scanning, we will assume devices or other security conditions are not scanned in any particular sequential order. The only key assumption is that each device is examined every  $\Delta_a$  seconds by the attacker and every  $\delta_d$  seconds by the defender.

The simple model of periodic scanning can closely model many types of defender and attacker activity observed on the Internet. Periodic scanning is a good model for many defender scans that are run daily or weekly to detect insecure conditions. Periodic scanning is also a good model for many worms that find new victims by randomly or sequentially scanning IP addresses. These have been widely analyzed and modeled (e.g., [8, 42]) and are still common (e.g., [9, 21, 33]). Worms are typically characterized by their scan rate specified as IP addresses scanned per second. This rate often varies as a function of the time and of the geographic location or IP address range. Variation is caused by many factors including the number of compromised computers contributing to a scan, a preference for local or remote scanning, a preference or aversion for certain languages or countries, and the time of day as it affects the number of computers that are on at any time. Periodic scanning where the scan interval varies with time and with the IP address range of the scanned network is generally a good model for worm-like scanning. It can model slow and stealthy scans with long scan intervals and rapid scans with short intervals. Periodic scanning is also a good model for scans performed by inside attacks that are launched from a computer inside a network boundary. Such scans often explore the whole IP address space of a local network, and they may be rapid or slow and stealthy to avoid discovery.

Even though attackers do not launch client-side attacks directly against target computers, client-side attacks can also be modeled as an attacker performing periodic scanning of victim hosts. In a client-side attack, an attacker either inserts malware on legitimate web servers or directs a user using email, social media, or other approaches to browse to a malicious web site with embedded malware. Alternatively, an attacker can trick a user into downloading malicious content that will run an exploit when the application required to open the content runs. In either case, when a user clicks on the infected web page or opens the infected file, malicious content is executed that exploits vulnerabilities in the browser or in helper applications used to display images, play movies, edit documents, or perform other functions. In the common case of malware on a web site, the attacker sampling interval is the average interval between visits of malicious web sites for a user. Compromising hosts using malware on web sites can be extremely effective because there are so many helper applications and so many vulnerabilities in these applications and browsers. As an example of the extent of this problem, the Google Safe Browsing Initiative detected more than 1.6 billion unique web pages with malicious content from Dec 2006 to April 2011 [35]. A recent study of a residential network with 20,000 hosts also found that after two weeks roughly 20% of these hosts had connected to one or more web sites found to contain malware by the Google Safe Browsing Initiative [21] and roughly 3% of these hosts became infected with malware. Although browsing patterns for individual users are complex (e.g., [19]), periodic scanning where the scan interval varies with time and IP address is a simple approach to model client-side attack attempts on hosts. Periodic sampling is also a reasonable model of spear-phishing attacks where a user is sent a carefully crafted email designed to entice them to open an attachment or visit a malicious web site. Here the attacker sample interval is the interval between receiving spear-phishing emails.

Actual attacker sampling intervals are difficult to determine because most attackers attempt to be stealthy and avoid detection. Our strategy is to initially set the attacker sampling interval  $\Delta_a$  to be large (e.g., slow sampling) to make it easier to obtain good low metric scores for motivational purposes. As time progresses and the network becomes more secure, the attack sampling interval should be lowered to values that are near actual values. Although actual values will never be known exactly because all attacks will never be detected, they can be estimated. For client-side attacks they can be estimated by examining the rate at which hosts visit black-listed web sites or download known malicious content. For server-side attacks, they can be estimated by examining traffic directed at monitored servers. Malicious intent might be determined from the source IP address or content. Rates also might be estimated from data collected by client honey nets (e.g., [35]) or by from analysis of recent worms (e.g., [33, 42]).

Periodic scanning with a time varying sample interval can also be used to model defender scans. Here, the actual sampling interval is known exactly. If defenders detect security conditions instantaneously, this can be modeled as periodic sampling with a sampling interval of zero. If there is a delay between the beginning of the insecure condition and detection, this can also be modeled as periodic sampling, but with a sampling interval equal to the delay. If scans occur weekly or monthly, as sometimes occurs for software vulnerability scans, the sampling interval is set to the time between examining each device which would be a week or month for this example. In addition to the sampling interval, defenders are characterized by the time it takes to process and eliminate

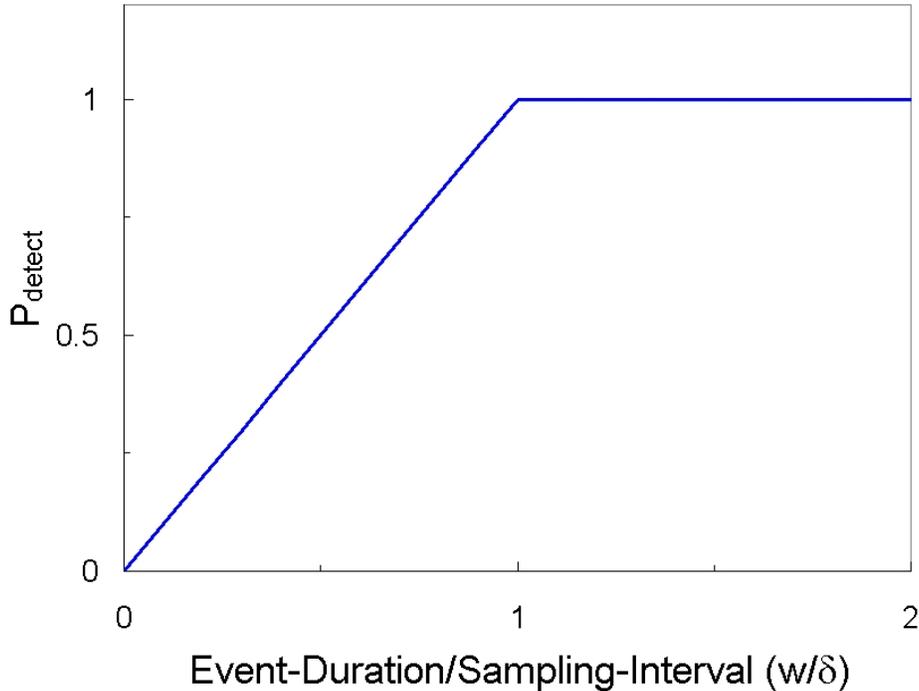


Figure 11. Probability of detecting a security condition of duration  $w$  by periodic scanning with a sampling interval of  $\Delta$ .

a security condition from a network. Since devices can be compromised during processing, small processing times are desired.

Two important attacker and defender characteristics required to estimate risk and capability are the probability that an attacker detects an insecure condition for operational metrics and the probability that a defender misses a security condition for capability deficit metrics. If we specify the duration of a security condition by  $w$  and assume periodic sampling with a sampling interval of  $\delta$  for a specific security condition, then the probability that the condition is detected is given by  $P_{\text{detect}} = \min(1, w/\delta)$ . This can be seen by considering an insecure condition with a duration that is much smaller than the sampling interval. As noted above, we assume that the starting time of an insecure condition is random and uniformly distributed relative to sampling times. Consider the illustration in Figure 10 and the regions in this figure where an insecure condition could start and be detected. When  $w$  is small, these regions will be of length  $w$  and occur immediately before each sampling time. The probability of detection will thus be the region length divided by the sampling interval. As  $w$  increases, the fraction of the space between samples filled by the region where the insecure condition is detection will increase until  $w$  reaches  $\delta$ . When this occurs, the insecure condition is always detected because the condition is always sampled at least once. The detection function is shown in Figure 11. It is drawn as a function of  $w/\delta$  because  $w$  normally varies across different insecure conditions and normalizing by  $\delta$  makes it possible to use this one curve for

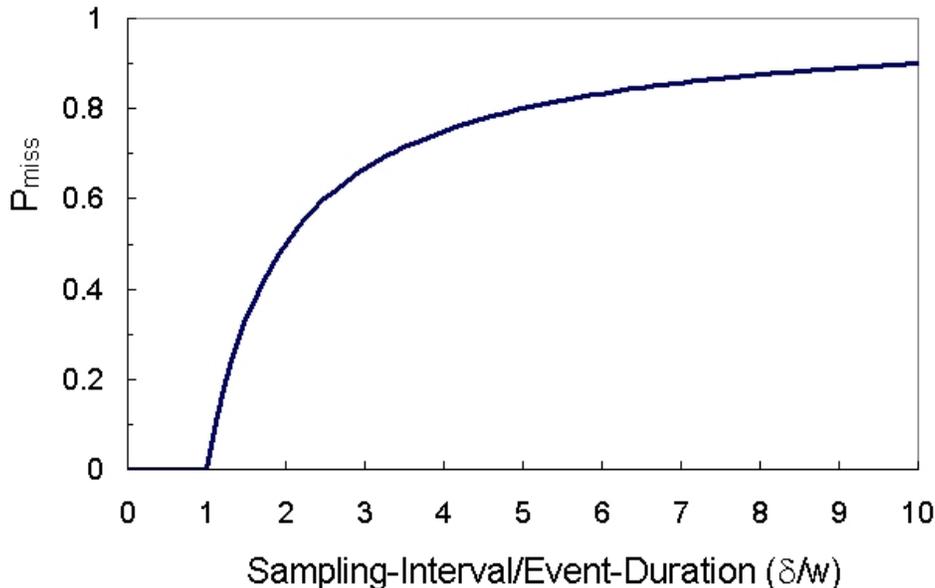


Figure 12. Probability of missing a condition of duration  $w$  by scanning with a sampling interval of  $\delta$ .

all predefined attacker sampling intervals. As described, the curve increases from zero for very short duration conditions to 1.0 when the duration of the condition is equal to the sampling interval and then stays at 1.0 as the condition duration increases. To be more specific, the probability that an attacker detects a condition  $i$ , present for duration  $w_i$  is

$$P_{\text{detect}}(i) = \min(1, w_i/\Delta_a). \quad (1)$$

To characterize defenders using periodic scanning, we first specify the shortest duration insecure condition that the defender must detect accurately and denote this minimum target duration as  $W_t$ . This normally should be shorter than the attacker sampling interval and than most normally occurring insecure conditions. We need to compute the probability that a defender using a scan interval  $\delta$  misses the shortest duration security condition of interest. The probability that a defender misses a security condition with a target duration  $w_i$  when using a sampling interval of  $\delta_d$  is 1 minus the detection formula in Eq. 1 but with defender notation, or

$$P_{\text{miss}} = 1 - \min(1, W_t/\delta_d) = \max(0, 1 - W_t/\delta_d). \quad (2)$$

This function is plotted in Figure 12. It is drawn as a function of  $\delta/w$  because because the defender varies the sampling interval  $\delta$  to obtain good detection while the target duration  $w$  is fixed. Normalizing by  $w$  makes it possible to use this one curve for all predefined target durations. This function is zero when the defender sampling interval is less than the target duration because the defender is then guaranteed to detect the condition. It then gradually rises to 1 as the sampling

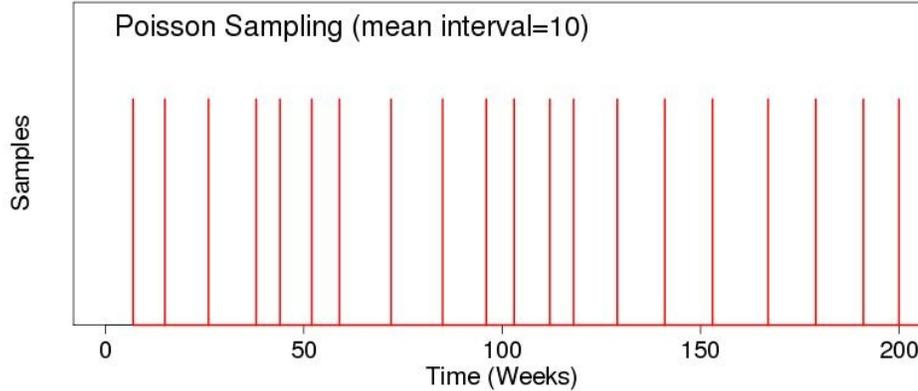


Figure 13. Poisson sampling used to detect insecure conditions.

interval increases. The increase is not linear because the horizontal x axis in Figure 12 is  $\delta/w$  but the miss probability is a function of the inverse of this ratio.

### 10.3 POISSON SCANNING

A Poisson process (e.g., [7]) has often been used to model users arriving to perform tasks of different durations. In our use of a Poisson process, users would be attackers or defenders arriving to detect insecure conditions, or users browsing web sites and being compromised by client-side attacks. This model assumes security conditions are sampled continuously and the intervals between samples are independent of one another. Specifically, the interval between samples  $\delta$  is assumed to have an exponential distribution with scale parameter  $\lambda$ ,  $p(\delta) = \frac{1}{\lambda} \exp^{-\frac{\delta}{\lambda}}$ . This model has historically been applied to telephone call arrivals at a switchboard. A Poisson process also accurately models some aspects of Internet traffic. For example, user-initiated remote-login and file-transfer sessions are well modeled as Poisson processes with hourly rates that vary with time [32]. Start times of user-initiated “navigation bursts” or tightly-spaced sequences of web pages downloaded by the same web client also are also well modeled using a homogeneous Poisson process where the time interval between bursts is caused by users reading web content before exploring other web pages [31].

Poisson sampling is illustrated in Figure 13. Once again, the horizontal axis represents time, and there is a vertical line or impulse whenever a security condition is sampled. As can be seen, there is more variability in the interarrival times than with periodic sampling. In this example, the scale parameter  $\lambda$ , which is the mean inter arrival time, is 10 weeks. Although the average interarrival time is 10, intervals vary according to the exponential distribution above. Variability is not too extreme and, for an exponential distribution with scale parameter  $\lambda$ , the standard deviation of intervals is  $\sqrt{\lambda}$  and 95% of intervals will be between zero and roughly  $3\lambda$ .

In a Poisson process, the probability of missing an insecure condition of length  $w$  is the probability that no sample occurs in interval  $w$ . Because intervals are independent and exponential,

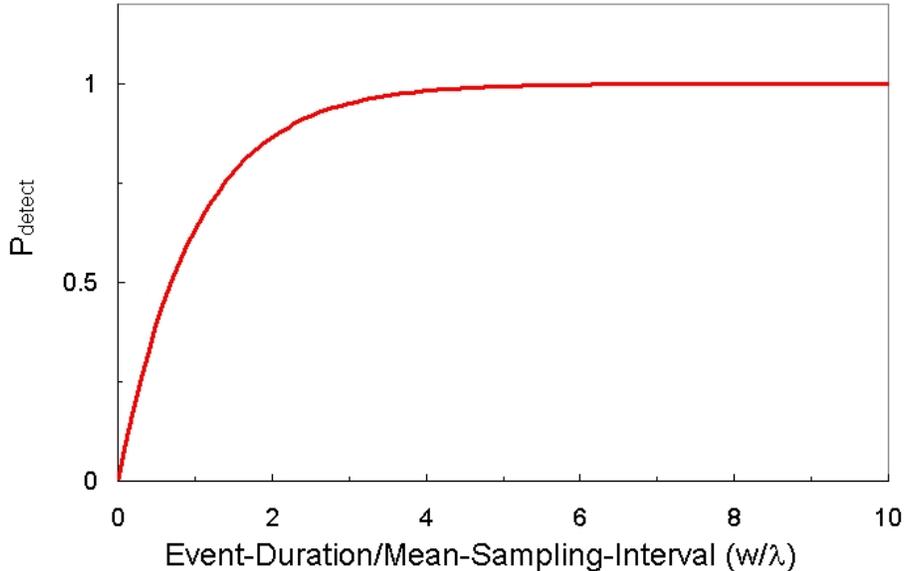


Figure 14. Probability of detecting a condition of duration  $w$  by Poisson scanning with an average sampling interval of  $\lambda$ .

this can be expressed as

$$P_{\text{miss}} = \exp^{-w/\lambda}. \quad (3)$$

The probability of detection is one minus this or

$$P_{\text{detect}} = 1 - \exp^{-w/\lambda}. \quad (4)$$

This detection curve is plotted in Figure 14. As in Figure 11, this curve increases from zero when the insecure condition duration is small relative to the average sampling interval to 1 when the insecure condition duration is large relative to the average sampling interval. Instead of rising linearly to 1 when the insecure condition is equal in duration to the sampling interval, the rise is gradual and exponential with Poisson sampling. The Poisson detection plot is thus a smoothed version of the detection plot obtained with periodic sampling. Accurate detection is only achieved when the insecure condition duration is 3 to 5 times the average sampling interval instead of equal to the sampling interval as with periodic sampling.

#### 10.4 PARETO SAMPLING

Measurements of web traffic have demonstrated that interarrival times for web browsing from individual browser clients exhibit self-similar bursty behavior that is not well modeled by a periodic or Poisson process (e.g., [2, 32]). Users browsing web sites and being exposed to client-side attacks may thus be more accurately modeled using a heavy-tailed distribution. As in [2], we model the inter

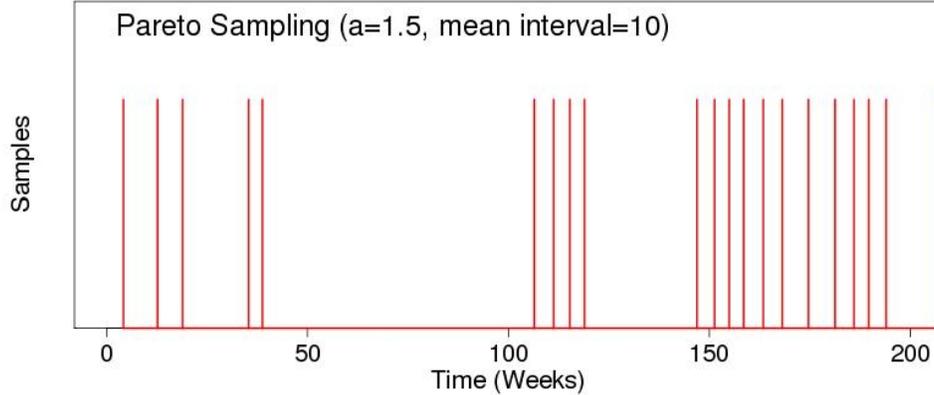


Figure 15. Pareto sampling used to detect insecure conditions.

arrival time between a single client visiting a web page, downloading all the content on that page, and then visiting again as heavy-tailed Pareto distribution. This distribution has two parameters. The minimum interarrival time is denoted as  $x_m$  and the Pareto index is denoted as  $a$ . The distribution of interarrival times  $\delta$  is non-zero only for  $\delta \geq x_m$  and is given by  $p(\delta) = a \frac{x_m^a}{\delta^{a+1}}$ . As the interarrival time increases, this function falls off more slowly than a Gaussian or Poisson distribution. The mean of this distribution,  $m$ , is only defined if  $a > 1$  and is specified by  $m = \frac{ax_m}{a-1}$ .

Pareto sampling is illustrated in Figure 15. As can be seen, there is much more variability in interarrival times than with Poisson or periodic sampling. In this example, the average interarrival time is 10 weeks,  $a$  is 1.5 as in [2], and the minimum duration is roughly 3.33 weeks. These interarrival times exhibit a bursty behavior where there are short bursts containing samples with relatively smaller interarrival times followed by much longer gaps. This type of bursty behavior can be observed if similar sample time plots are created over longer time scales and is often called self-similar behavior. The result is that the probability of long interarrival times falls off slowly only as an inverse power of the interarrival time. In a long measurement interval, there almost always will be a few longer duration interarrival times.

We developed a closed-form solution to the probability that Pareto sampling detects a insecure condition present for duration  $w$ . This combines an equation when security condition durations are less than the minimum interarrival time with another equation when security conditions are greater than this time. It also assumes that  $a > 1$ , so the mean of the Pareto distribution is defined. First consider  $N$  sample times as in Figure 15 over a finite, but long measurement interval. When  $w$  is less than  $x_m$ , start times for a security condition that will be detected will extend from a duration  $w$  before each sample time to the sample time. Because the minimum interval between two samples is  $x_m$ , each security condition with duration less than  $x_m$  will only be detected by one sample, and the same analysis used with periodic sampling can be applied. The total average duration for  $N$  samples is  $(N - 1) \cdot m$ , where  $m$  is the mean interval. The total duration of time where a short security condition can start and be detected is  $(N - 1) \cdot w$ . The probability of detecting a short security condition that begins uniformly randomly over the measurement interval

is thus  $P_{\text{detect}} = \frac{(N-1) \cdot w}{(N-1) \cdot m} = \frac{w}{m}$ . This is the same linear equation used to calculate the probability of detection for periodic sampling except the fixed interarrival interval for periodic sampling is replaced by the mean interarrival time and the equation is valid only when  $w \leq x_m$ .

When the security condition duration is larger than  $x_m$ , it can interact with more than one sample. Instead of computing the probability of detection, we can compute the probability that the large interval is missed by Pareto sampling. To be missed, the interarrival time between two samples must be greater than  $w$ . Again, considering  $N$  samples, we first determine the expected number of samples with interarrival times greater than  $w$  denoted as  $N_l$ . This will be the probability that the interarrival time  $\delta$  is greater than  $w$  times  $N - 1$ . For the Pareto distribution, this is  $N_l = (N - 1) \cdot P(\delta > w) = (N - 1) \cdot \left(\frac{x_m}{w}\right)^a$ . The total duration where a security condition of length  $w$  can start and not be detected will be  $N_l$  times the average duration of intervals of length greater than  $w$  minus  $w$  because a security condition will be detected in the region of length  $w$  immediately before each sample. The average length of security conditions of length greater than  $w$  is  $m_l = \frac{aw}{a-1}$  due to the self-similarity property of the Pareto distribution. The total average duration over the measurement interval where a security condition of length  $w$  will not be detected is thus  $N_l \cdot (m_l - w) = (N - 1) \cdot \left(\frac{x_m}{w}\right)^a \cdot \left(\frac{aw}{a-1} - w\right) = (N - 1) \cdot \left(\frac{x_m}{w}\right)^a \frac{w}{a-1}$ . The total expected duration of the measurement interval is  $N - 1$  times the average interarrival time or  $(N-1) \cdot m = (N-1) \cdot \frac{ax_m}{(a-1)}$ . The probability of missing a long security condition that begins uniformly randomly over the measurement interval is the ratio of these durations or  $P_{\text{miss}} = \frac{(N-1) \cdot \left(\frac{x_m}{w}\right)^a \frac{w}{a-1}}{(N-1) \cdot \frac{ax_m}{(a-1)}} = \frac{1}{a} \cdot \left(\frac{x_m}{w}\right)^{a-1}$ . The probability of detecting a long duration security condition is 1 minus this or  $P_{\text{detect}} = 1 - \frac{1}{a} \cdot \left(\frac{x_m}{w}\right)^{a-1}$ . Combining these two detection probabilities for small and large security condition durations results in the following formula

$$P_{\text{detect}} = \begin{cases} \frac{w}{m} & \text{for } w < x_m \\ 1 - \frac{1}{a} \cdot \left(\frac{x_m}{w}\right)^{a-1} & \text{for } w \geq x_m \end{cases} . \quad (5)$$

The probability of missing a security condition is 1 minus this equation.

Figure 16 shows this detection probability as a function of  $w/m$  when  $a = 1.5$  using a linear x-axis scale as in Figure 11. As can be seen, the probability of detection increases much more slowly than with periodic or Poisson sampling. Even when the duration of the security condition is 10 times the average sample interval, the probability of detection is still slightly below 0.9. This is caused by the many long-duration intervals generated by Pareto sampling.

A plot that shows the detection probability for Pareto sampling over a wider range of  $w/m$  is shown in Figure 17. In this figure, the x-axis uses a log scale and  $w/m$  ranges from 0.1 to 1,000. Here it can be seen that when security condition duration is 1,000 times the average sampling duration, the probability of detection is .988. The probability of detection is above 0.9 when the security condition duration is 15 times the average sampling interval. Pareto sampling is clearly more problematic for an attacker or defender. For the same effort, measured by the number of times tests are performed to detect an insecure condition, attackers and defenders are more likely to miss short duration insecure conditions.

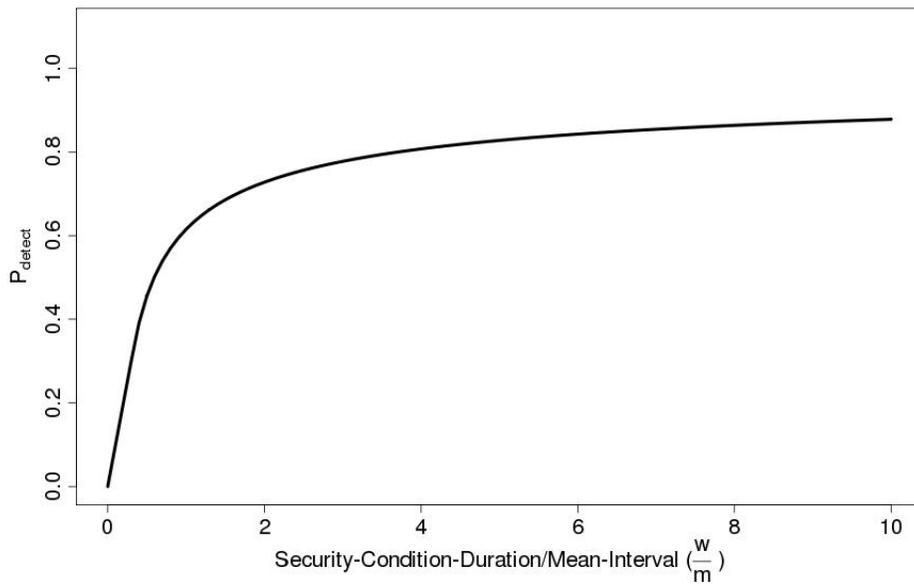


Figure 16. Probability of detecting a condition of duration  $w$  by Pareto scanning with an average sampling interval of  $m$  and  $a = 1.5$ .

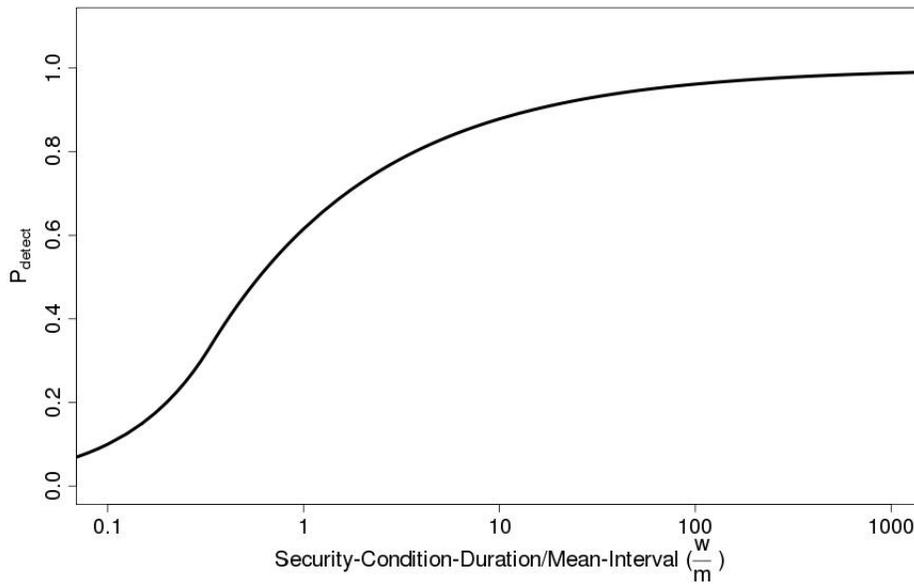


Figure 17. Probability of detecting a condition of duration  $w$  by Pareto scanning with an average sampling interval of  $m$  and  $a = 1.5$ . The  $x$ -axis uses a log scale and ranges from 0.1 to 1,000.

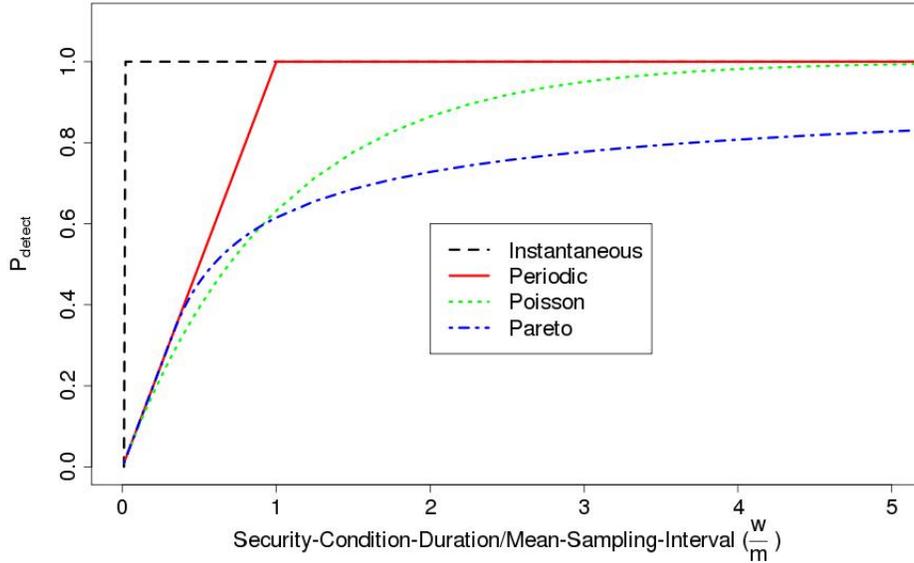


Figure 18. Probability of detecting a condition of duration  $w$  by four different types of sampling.

## 10.5 COMPARISON OF FOUR TYPES OF SAMPLING

Figure 18 shows the probability of detecting an insecure condition of duration  $w$  using the four types of sampling described above. Instantaneous sampling always has a probability of detection of 1. The detection probabilities for other types of sampling rise from zero when the insecure condition is much shorter than the sampling interval to equal or approach 1 when the insecure condition is much longer than the average sampling interval.

Figure 19 shows the probability of missing an insecure condition of duration  $w$  using the four types of sampling described above. Instantaneous sampling always has a miss probability of 0. The mean sampling interval for periodic sampling only needs to be below the duration of the security condition for a miss probability of zero. For Poisson and Pareto interarrival times, the mean sampling interval has to be much less than the insecure condition duration for low miss probabilities.

In the remainder of this report we will make the simplifying assumption that attackers and defenders scan periodically. Instantaneous scans will be modeled by periodic scans with a sampling interval equal to zero or the detection delay. This is a conservative, worse-case, assumption for attacker scans because the probability of detection curve in Figure 18 for periodic scanning is always greater than the probability of detection for Poisson and Pareto scanning. It is a reasonable assumption for defender scans because most defender scans are performed periodically. In situations where scanning differs from periodicity, the equations in this section for the other types of scanning should replace the simpler detection and miss probability equations derived for periodic scanning.

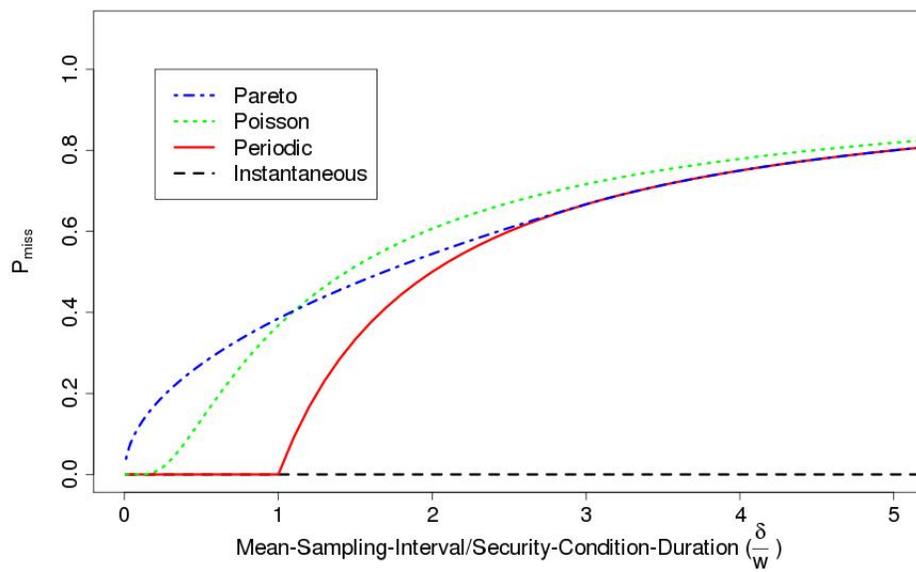
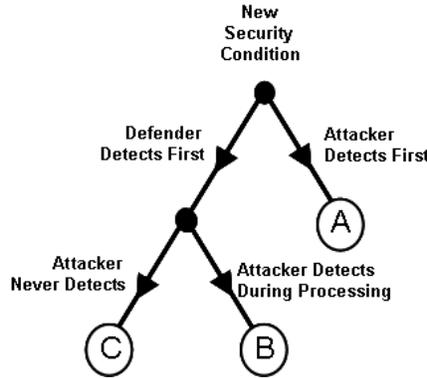


Figure 19. Probability of missing a condition of duration  $w$  by four different types of sampling.

## 11. USING PERIODIC SAMPLING ADVERSARY MODELS TO COMPUTE COMPROMISE PROBABILITIES

To perform a theoretical analysis that provides some insight into the adversarial process, we can assume that insecure conditions are present for a known duration and use adversarial modeling to predict the probability of compromise for a device with an insecure condition as defender processing and sampling intervals vary and as attacker sampling intervals vary. In practice, the duration of insecure conditions needs to be measured and the number and duration of insecure conditions can vary substantially across devices. We will perform the analysis for a single device with one security condition and assume the device is compromised if it is discovered by the adversary before it is discovered by the defender or while it is being processed by the defender. To simplify the computation, we will assume that the defender sampling interval is smaller than the attacker sampling interval ( $\delta_d < \Delta_a$ ), that the security condition duration is greater than the attacker sampling interval ( $w_i > \Delta_a$ ), and that all durations and times are known exactly.



*Figure 20. Event tree for a long-duration insecure condition.*

Under these assumptions, there are three events that can occur as shown in the event tree of Figure 20. Starting at the top of the tree when a new security condition occurs, there are two choices. If the attacker detects the security condition before the defender, this leads to state A on the right. If the defender detects the security condition first, then the defender processes and then eliminates the security condition as shown on the left. If the attacker still does not detect the security condition during processing, this leads to state C and if the attacker detects the security condition during processing, this leads to state B. A device with a security condition is compromised in states A and B and not in state C. Figure 7 shows the sequence of events and notation for state C when the device is not compromised. The first event that occurs along the time line is that the security condition begins. After a time interval of  $x_d$ , the security condition is detected by the defender and processing begins. Processing is complete after another time interval of  $\delta_p$ , and

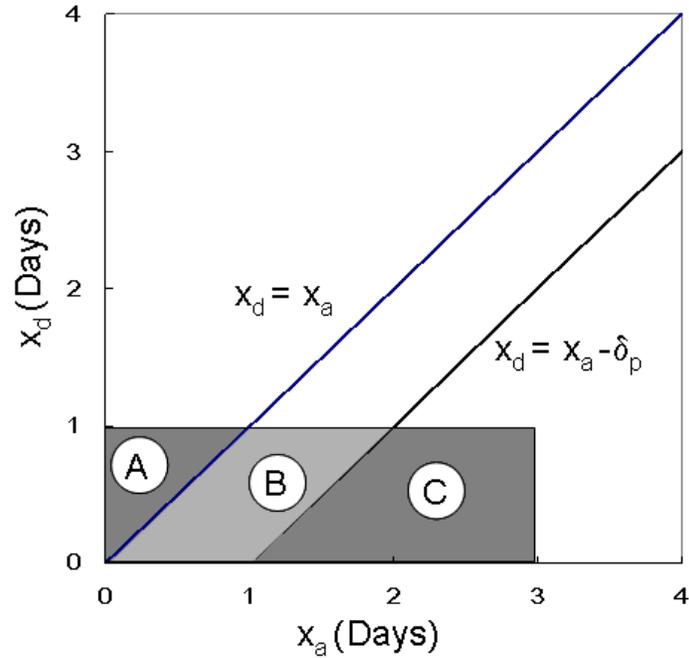


Figure 21. Joint distribution of the time it takes an attacker and defender to detect a security condition.

then the security condition no longer exists. The final event is the attacker who samples the device looking for the security condition after a time interval of  $x_a$  after the security condition begins.

Figure 21 shows the joint distribution of  $x_a$  and  $x_d$ , which is a rectangular region with a constant height where  $x_a$  extends from 0 to  $\Delta_a$  and  $x_d$  extends from 0 to  $\delta_d$ . In this figure, the horizontal axis represents the time from when the security condition begins to the first attacker sampling time and the vertical axis represents the time to the first defender sampling time. The rectangle is wider than it is high because the attacker sampling interval is longer than the defender sampling interval. The distribution shown assumes that the attacker samples each device every 3 days and the defender samples each device every day. The rightmost dark diagonal line indicates the boundary between successful and failed attacks. In regions “A” and “B,” to the left, the attacker detects the insecure condition before the defender detects and processes it. In region “C,” the attacker fails to detect the insecure condition. The leftmost diagonal line indicates the boundary when the defender processing time is zero. If the attacker sampling time is much greater than the defender sampling time and the processing time is small, region “C” will be a large fraction of the rectangle and the probability that the attacker fails to detect the insecure condition will be high.

It is possible to integrate the area to the left of the rightmost slanted line in Figure 21 to create a mathematical expression for the probability of compromise as a function of the processing time, defender sampling interval, and attacker sampling interval. If the attacker sampling interval is greater than the processing time, and both values are constant, then the probability of compromise

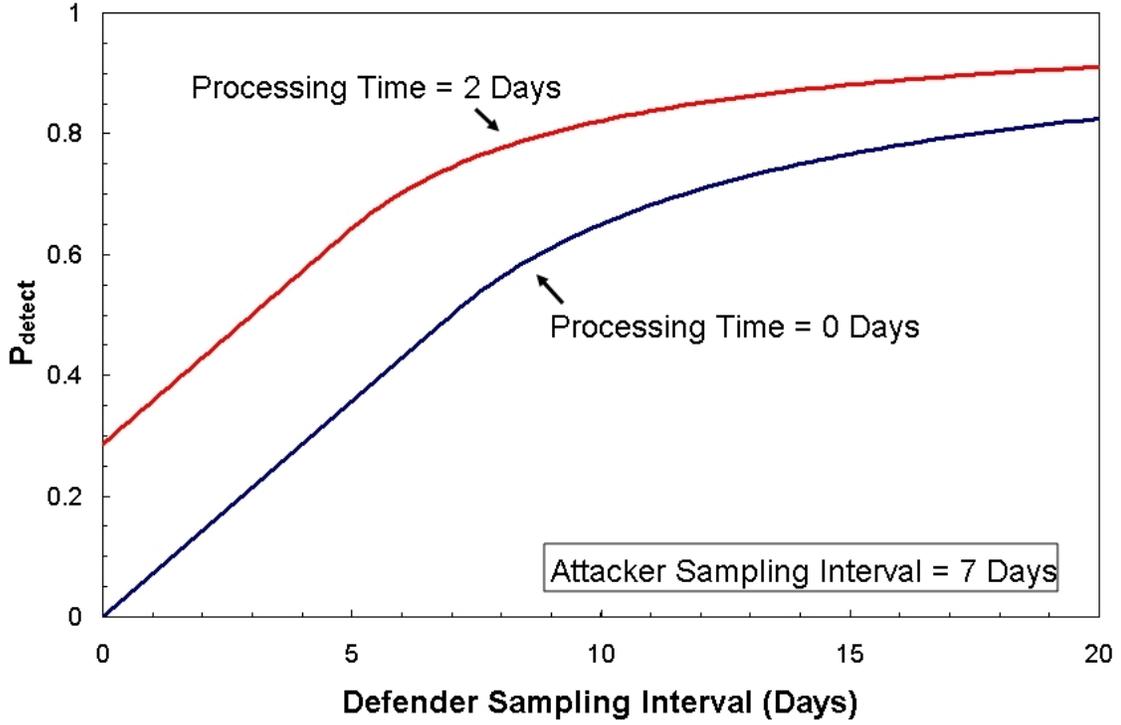


Figure 22. Probability of compromise as a function of defender sampling time.

can be expressed as a function of the defender sampling interval as

$$P_{\text{compromise}}(\delta_d) = \frac{1}{\Delta_a \delta_d} \left\{ \delta_p \delta_d + \frac{\min(\delta_d, \Delta_a - \delta_p)^2}{2} + \max\{0, \delta_d - (\Delta_a - \delta_p)\}(\Delta_a - \delta_p) \right\}.$$

A plot of this function is shown in Figure 22 when the attacker sampling interval is 7 days and the processing time is 0 days (lower curve) or 2 days (upper curve). It can be seen that the minimum probability of compromise occurs when both the processing time and defender sampling times are minimum. Increasing the processing time increases the probability of compromise as does increasing the defender sampling interval.

When the processing time approaches zero, the probability of compromise is determined primarily by the attacker sampling interval. It is half the ratio between the defender and attacker sample times or  $P_{\text{compromise}} = 0.5\delta_d/\Delta_a$ . This holds only if the attacker sampling interval is greater than the defender sampling interval. If the ratio of the attacker sampling interval to the defender sampling interval is 10 to 1, then the probability of compromise is .05. This increases to .25, when the ratio of attacker to defender sampling intervals drops to 2 to 1.

When the defender sampling interval approaches zero, the probability of compromise is determined primarily by the processing time. As long as the processing time is less than say 90% of

the attacker sampling interval, the probability of compromise is roughly the processing time divided by the attacker sampling interval or  $P_{\text{compromise}} = \delta_p / \Delta_a$ . The probability of compromise thus increases linearly as the processing time increases.

These limiting cases and the overall result demonstrate that both the attacker sampling interval and the processing time must be small relative to the attacker sampling interval to prevent attackers from exploiting insecure conditions.

## 12. COMBINING THE EFFECT OF MANY INDIVIDUAL INSECURE CONDITIONS

When developing security metrics, it often happens that a device, software package, or other entity has many insecure conditions where the probability of successful compromise after discovery using each condition alone is known and less than 1.0. For example, a device may have many software packages, each with one or more vulnerabilities, as in LR-3, or the operating system on a device may have many misconfigurations as in LR-4. In this situation, we would like to compute the overall probability of successful device compromise.

There are many attack models that we could use to compute the probability of successful compromise. The model that we use in this report is an attacker who tries to exploit all vulnerabilities one at a time and stops only after success. This is simple and well defined, which is why we use it. When an attacker tries to exploit all vulnerabilities, the probability of a device not being compromised is the probability that exploits for all vulnerabilities fail. One minus this will then be the probability of compromise. We denote the probability of a successful compromise using vulnerability  $i$  as  $P_{\text{success}}(i)$ . Assuming that this probability is independent for each vulnerability, then the probability of all vulnerabilities failing is the product of all  $(1 - P_{\text{success}}(i))$  terms. The probability of successfully compromising a device is one minus this product or

$$P_{\text{device\_compromise}} = 1 - \prod_{i \in \text{vulns}} (1 - P_{\text{success}}(i)). \quad (6)$$

We refer to this common computation as a complementary product and use this formula to model the effect of multiple vulnerabilities and misconfigurations.

The complementary product has more common-sense properties than other simple approaches to combining compromise probabilities as shown in Table 3. In this table, the probability of a successful compromise using only the single vulnerability  $i$  is represented by  $P_i$ . The first column of this table shows alternative formulas that can be used to combine compromise probabilities. The second column describes the attacker model that motivates each formula. The remaining columns indicate whether the formulas have four common-sense properties. Each row of this table corresponds to a different approach to combining compromise probabilities. The first row uses the complementary product described above and represents an attacker who tries all exploits. The second row uses a simple average and represents an attacker who randomly tries one exploit. The third row uses a weighted average and represents an attacker who selects a single exploit, but the selection probability is weighted by the probability of compromise for each vulnerability. The final row uses a  $\text{max}()$  function and represents an attacker who selects the single vulnerability with the highest probability of compromise. In the middle boxes of this table,  $\checkmark$  indicates a property holds and  $\times$  indicates it does not. The bracketed examples in the tables represent compromise probabilities for vulnerabilities. As can be see, the complementary product  $1 - \prod (1 - P_i)$  that we use is the only combining formula that has all common sense properties listed in this table. All other methods fail one or more of the common-sense tests.

**TABLE 3**

**Failure Probability Combination Formulas**

Combination formula for overall probability of failure	Derived from an attack model where the attacker tries	Adding a vulnerability does not improve device security	Using a single vulnerability results in the compromise probability for that vulnerability $\{P\} \rightarrow P$	Adding a vulnerability worsens device security unless the compromise probability is already 1.0	Worsening a vulnerability does not improve device security
$1 - \prod(1 - P_i)$	✓ all exploits	✓	✓	✓	✓
$\sum P_i/N$	✓ single exploit at random	× [1.0] vs. [1.0, 0.1]	✓	× [0.5] vs. [0.5, 0.5]	✓
$(\sum P_i^2) / \sum P_i$	✓ single exploit at random weighted by success	× [1.0] vs. [1.0, 0.1]	✓	× [0.5] vs. [0.5, 0.5]	× [1.0, 0.1] vs. [1.0, 0.2]
$\max(P_i)$	✓ single best exploit	✓	✓	× [0.5] vs. [0.5, 0.5]	✓

Using a complementary product assumes that the attacker has exploits for all vulnerabilities and isn't concerned about exposing potentially expensive-to-develop exploits that a defender may discover and then render ineffective by creating effective defenses. It also assumes the attacker isn't concerned about being detected but is willing to try many new and old exploits that may be detected by recently updated intrusion detection or prevention systems. Other more complex attacker models that we plan to explore in the future include those that are more concerned about cost of exploit development, probability of success, and detection. Some examples include 1) an attacker concerned about detection and the cost of exploit development who tries to exploit only the single vulnerability with the highest probability of success or the first few vulnerabilities with the highest probabilities of success, 2) an attacker concerned about detection who randomly tries to exploit one or a small number of vulnerabilities, 3) an attacker with a limited budget to develop exploits who uses only publicly available exploits available from organizations such as Metasploit [25], and 4) an attacker with a larger budget who illegally purchases vulnerabilities as is done ethically by the Tipping Point Zero Day Initiative TippingPoint [45].

**This page intentionally left blank.**

### 13. CONSIDERING ERRORS CAUSED BY SAMPLING INSECURE CONDITION DURATIONS

In the above analyses, we assume that the durations of insecure conditions are known exactly. In practice, these durations are determined by sampling performed by the defender. This sampling complicates the analysis and can introduce errors in metric computations. This section analyzes the errors caused by sampling. First, we demonstrate how to estimate the duration of an insecure condition by periodic defender sampling in a way that eliminates statistical bias. Second, we analyze the maximum error that occurs when estimating the probability that an attacker detects an insecure condition when attackers and defenders use periodic sampling. We show that estimation error is acceptable and less than 10% when the defender sampling interval is less than the minimum duration of insecure conditions of interest.

First, we focus on estimating the duration of insecure conditions by periodic defender sampling. A key component of our analysis of periodic sampling is Equation 1 which shows that the probability that an attacker detects an insecure condition present for duration  $w$  using periodic sampling with a sampling interval  $\Delta$  is

$$P_{\text{detect}} = \min(1, w/\Delta). \quad (7)$$

In practice, the insecure condition duration  $w$  is unknown and the attacker sampling interval is assumed to be known. When a defender estimates the insecure condition duration using instantaneous sampling, then this duration is known exactly and the formula can be computed accurately. When the defender estimates the duration using periodic sampling, errors are introduced because the estimated insecure condition duration  $\hat{w}$  is different than the true condition duration.

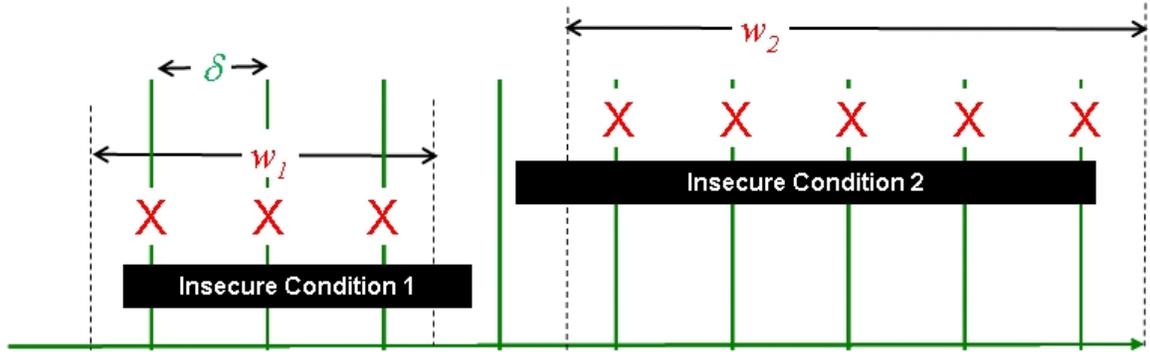


Figure 23. Estimating the duration of insecure conditions using periodic sampling.

Figure 23 illustrates the errors in measuring the start and end times of security conditions that can be caused by sampling. The horizontal axis in this figure represents time and the vertical lines represent times when the defender, using periodic sampling with a sampling interval of  $\delta$ , tests

to find the insecure condition. Two different insecure conditions are shown by the black rectangular regions. The X's show where sampling detected the insecure conditions. The condition on the left labeled "Insecure Condition 1" is sampled three times and the condition on the right labeled "Insecure Condition 2" is sampled five times. As can be seen, the actual begin and end times of these security conditions are not detected accurately by sampling times. In "Insecure Condition 1," the first sample is relatively close to the beginning of the insecure condition but the last sample is relatively farther away from the end of the insecure condition. For "Security Condition 2," the measurement error between the nearest sample time within the insecure condition and actual start and end times is greater for the start than for the end of the insecure condition. This measurement error ranges from 0 to  $\delta$  and has an average of  $0.5 \cdot \delta$  assuming the start and end times of insecure conditions are uniformly distributed over a measurement interval. The first sample time when a condition is detected is on the average  $.5 \cdot \delta$  after the true insecure condition start time and the last sample time when a condition is detected is on the average  $.5 \cdot \delta$  before the true end time. An unbiased estimate of the duration of an insecure condition is thus the number of defender samples that detect the condition times the sample interval or

$$\hat{w} = NS_{\text{detect}} \cdot \delta, \quad (8)$$

where  $NS_{\text{detect}}$  is the number of defender samples that detected the insecure condition. This estimate is shown by the intervals labeled  $w_1$  and  $w_2$  in Figure 23. Although the start and end times of the estimated regions are different from the true start and end times of the insecure conditions, the duration estimates that results are statistically unbiased.

In some situations, defenders sample to detect the beginning of the insecure condition but know exactly when the insecure condition ends because they process the insecure condition and know exactly when it is eliminated. In this case, the insecure condition duration should be estimated using the first sample time adjusted to eliminate bias and the actual end time as

$$\hat{w} = (t_{\text{true\_end}} - [t_{\text{first\_seen}} - .5 \cdot \delta]), \quad (9)$$

where  $t_{\text{true\_end}}$  is the known end time of the insecure condition and  $t_{\text{first\_seen}}$  is the time of the first sample to detect the insecure condition.

Now we focus on determining the error in estimating the probability that an attacker detects an insecure condition. The above formulas compensate for the average sampling bias in measuring the duration of insecure conditions. Even with zero bias, however, an expected error can be introduced when using the duration estimates in Eq. 8 to estimate the probability of detection of an insecure condition by an attacker because the duration estimates are processed nonlinearly as shown in Figure 11. It is important to determine when this error is so large that risk computed using the attacker probability of detection is no longer meaningful. Such an error analysis will enable us to specify how small the defender sampling interval  $\delta$  needs to be relative to the security condition duration  $w$  to enable accurate risk estimation.

Following from above, the estimated probability of an attacker detecting and compromising a device is

$$\hat{P}_{\text{detect}} = \min \left( 1, \frac{\hat{w}}{\Delta} \right), \quad (10)$$

where  $\hat{w}$  is the estimated insecure condition determined by periodic defender sampling and calculated using Eq. 8, and  $\Delta$  is the attacker sampling interval. If we assume insecure conditions occur with start and end times that have a uniform distribution relative the measurement interval, then the ratio of  $\hat{w}$  and  $\delta$  determines the number of times the defender will sample the condition. The defender will either sample the insecure condition  $\lceil \frac{w}{\delta} \rceil$  times or  $\lceil \frac{w}{\delta} \rceil - 1$  times, where this ‘‘ceiling’’ function  $\lceil x \rceil$  represents the smallest integer greater than or equal to  $x$ . The probabilities of sampling the insecure condition these number of times are listed in the following table.

Probability	# Times Defender Samples Insecure Condition
$\frac{w}{\delta} - \lceil \frac{w}{\delta} \rceil + 1$	$\lceil \frac{w}{\delta} \rceil$
$\lceil \frac{w}{\delta} \rceil - \frac{w}{\delta}$	$\lceil \frac{w}{\delta} \rceil - 1$

These probabilities and Eq. 8 lead to the expected estimated duration equations in second column of the table below. These expected durations and Eq. 10 lead to the probability of attacker detection equations in the third column of the table below.

Probability	Expected Duration	Probability of Attacker Detection
$\frac{w}{\delta} - \lceil \frac{w}{\delta} \rceil + 1$	$\lceil \frac{w}{\delta} \rceil \cdot \delta$	$\min \left( 1, \lceil \frac{w}{\delta} \rceil \cdot \frac{\delta}{\Delta} \right)$
$\lceil \frac{w}{\delta} \rceil - \frac{w}{\delta}$	$(\lceil \frac{w}{\delta} \rceil - 1) \cdot \delta$	$\min \left( 1, (\lceil \frac{w}{\delta} \rceil - 1) \cdot \frac{\delta}{\Delta} \right)$

The expected value of the probability of attacker detection estimate can be computed from the equations in the above table as

$$E \left[ \hat{P}_{\text{detect}} \right] = \left( \frac{w}{\delta} - \lceil \frac{w}{\delta} \rceil + 1 \right) \min \left( 1, \lceil \frac{w}{\delta} \rceil \frac{\delta}{\Delta} \right) + \left( \lceil \frac{w}{\delta} \rceil - \frac{w}{\delta} \right) \min \left( 1, (\lceil \frac{w}{\delta} \rceil - 1) \frac{\delta}{\Delta} \right).$$

The error in this estimate is the difference between this expected value and the true probability of detection or

$$Error = E \left[ \hat{P}_{\text{detect}} \right] - \min \left( 1, \frac{w}{\Delta} \right).$$

This error equation is a complex function that depends on the ratio of the insecure condition duration and the defender sampling interval, and the ratio between the defender and attacker sampling intervals. By selecting the ratio between the defender and attacker sampling interval that

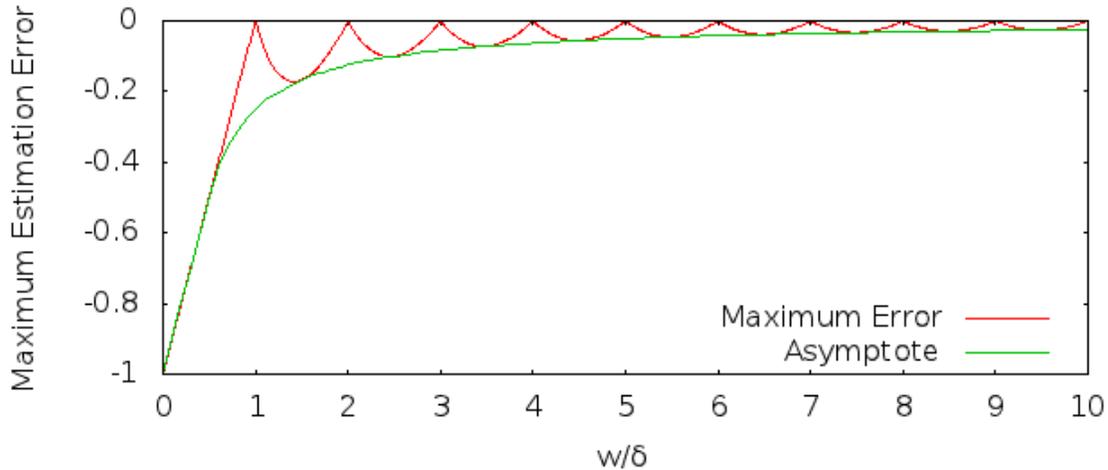


Figure 24. Maximum expected error in estimating the probability of attacker detection.

maximizes this error for each value of the ratio between the insecure condition duration and the defender sampling interval, we created the plot of this maximum error shown in Figure 24.

This graph indicates that risk estimation is unreliable when the defender sampling interval is greater than the duration of the insecure conditions. This is shown by the high error values near -1 in the region to the left of the value where  $\frac{w}{\delta} = 1$  on the x axis. For example, if we test yearly for a condition that lasts one week, then  $\frac{w}{\delta} = \frac{7}{365} \approx .02$  which is much less than 1. In this example, the probability of our detecting the insecure condition is only .02. If we believe that the attacker is able to exploit this vulnerability once every two weeks, then the actual probability of compromise is 0.5 whereas our observation based estimate will be only .02 resulting in an error of -0.48.

This analysis demonstrates that the error in estimation attacker probability of detection is acceptable when the defender sampling interval is less than the minimum duration insecure condition of interest. It shows that the maximum error approaches zero towards the right of the graph when the defender sampling interval becomes small relative to the insecure device duration and the insecure condition is sampled many times. For the same example of a security condition that lasts 1 week, if we test every day, our maximum risk estimate error will be only a few percent.

In addition, the maximum error, when the defender sampling interval is less than the minimum duration insecure condition of interest, is roughly 20% and this occurs only when the probability of attacker detection is near 1. This results in only a small underestimate of risk in the worst case when the attacker sampling interval is equal to the duration of the insecure condition.

This analysis assumes the start and end times of a security condition are estimated using periodic sampling. If a defender detects the start time, begins processing the security condition, and knows the end time exactly, the maximum error reduced by a factor of 2. The maximum error

rate when the defender sampling interval is shorter than the minimum duration insecure condition of interest is thus less than 10% which is acceptable for most risk assessments.

**This page intentionally left blank.**

## 14. COMMON METRIC COMPONENTS AND OPTIONS

Every new metric requires three components corresponding to the three stages of the metric maturity model described in Section 4. In the first maturity stage, checklist metrics are developed. These specify the data, processes and capabilities required to assess risk from the associated attack type. In the second maturity stage, Capability Deficit (CD) metrics are developed. These measure whether specifications of security properties are available and whether the timeliness, coverage, and accuracy of measurements are sufficient to compute accurate risk estimates in the third maturity stage. Until CD metrics are low enough, risk can not be estimated accurately in the third maturity stage.

The overall value of a CD metric is a combination of its specification deficit ( $SD$ ) and accuracy deficit ( $AD$ ). The specification deficit indicates whether all specifications required to compute risk have been created. Examples are lists of authorized devices and authorized software as shown in the fourth column of Table 1. The accuracy deficit is computed from two components. First, the timeliness deficit ( $TD$ ) measures both coverage or whether tests are performed over all relevant entities in the network and whether security conditions are scanned frequently enough to reliably detect the shortest duration security condition of interest. Second, the total testing error ( $TTE$ ) measures the fraction of errors or misclassifications for the security tool that performs the test assuming the test is timely and specifications are accurate. The total testing error allows us to take into account the underlying accuracy of the approach used to perform security tests. It is an advanced feature that can be set to 0 if it can not be accurately estimated. For example, for CC-1, if unauthorized devices are detected using a MAC address as a unique identifier, misclassifications can occur for multiple virtual machines that use the same auto-generated MAC address. If the total testing error has been measured experimentally on a testbed with known characteristics, we use the measured value. If it is composed of multiple measurable components, we combine the error of each component using a complementary product. Without any information on the misclassification rate of an approach, for simplicity, we assume zero total testing error.

We provide two options and two formula to compute the CD metric. A motivational option (CD-A) was developed to motivate system administrators to reduce the CD metric and a probabilistically rigorous option (CD-B) was developed to be more accurate under common probabilistic assumptions. In option CD-A, credit is provided for creating specifications even when no tests are performed. In option CD-B no credit is provided until the overall accuracy is reduced. We will describe both options here, but present only option CD-A when describing individual metrics.

In general, for a particular entity  $i$  (subnet, device,...), the CD metric is defined as a function of the specification deficit and the accuracy deficit. When using the motivational option CD-A we use the formula

$$\text{Option CD-A (Motivational): } CDM(i) = \frac{SD(i) + 3 \cdot AD(i)}{4}. \quad (11)$$

TABLE 4

Two Options for Computing Capability Deficit Metrics. The Default is CD-A.

Options	Formula
CD-A (Motivational)	$\frac{SD(i)+3 \cdot AD(i)}{4}$
CD-B (Rigorous)	$1 - (1 - SD(i)) \cdot (1 - AD(i))$

This formula is intended to motivate improvement by providing some credit for creating a specification before sampling has begun. A system administrator can reduce the CD metric from 1.0 to .75 simply by creating specifications for all security conditions. The CD metric then can only be reduced to zero if the AD is reduced to zero. The decision to weight the SD by 1 and the AD by 3 is reasonable because CD scores for option CD-A are similar to those for the more rigorous option CD-B when they are low, but a system administrator sees some change after the first step of creating specifications.

For the more rigorously accurate option CD-B we use a complementary product to combine  $SD$  and  $AD$  which assumes that these terms represent the probability of a condition being specified and accurately detected and that these are independent. This leads to the following formula

$$\text{Option CD-B (Rigorous): } CDM(i) = 1 - (1 - SD(i)) \cdot (1 - AD(i)). \quad (12)$$

The two options for computing CD metrics are summarized in Table 4.

The global CD metric is computed as a weighted average over entities such as subnets and devices as listed in column 2 of Table 1. To compute the global CD metric, we first assign  $S_{\text{entity}}(i)$  as the importance of given entity  $i$  and  $T_{\text{Entities}}$  as the total importance across entities

$$T_{\text{Entities}} = \sum_{i \in \text{Entities}} S_{\text{entity}}(i).$$

The global CD metric is then the importance-weighted average of the per-entity Capability Deficit Metric values

$$CDM = \frac{1}{T_{\text{Entities}}} \sum_{i \in \text{Entities}} S_{\text{entity}}(i) \cdot CDM_{\text{entity}}(i). \quad (13)$$

If all entities are weighted equally, this reduces to

$$CDM = \frac{1}{N} \sum_{i \in \text{Entities}} CDM_{\text{entity}}(i), \quad (14)$$

where  $N$  is the number of entities. As noted above, CD metrics vary from 0, the best possible score, to 1, the worst.

TABLE 5

Three Options for Computing Operational Metrics. The Default is OM-A.

Option	Equation
OM-A: Basic	$\sum_{i \in \text{devices}} P_{\text{detect}}(i),$
OM-B: Asset Values	$\frac{1}{T_{AV}} \sum_{i \in \text{devices}} AV(i) \cdot P_{\text{detect}}(i),$
OM-C: Exploit Success Probability	$\sum_{i \in \text{devices}} P_{\text{ExploitSuccess}}(i) \cdot P_{\text{detect}}(i)$

Operational metrics measure the risk posed by the critical control’s attack type *based on* insecure condition durations measured by the defender. Until capability is reasonably mature and the CD metric is small, the operational metric has no meaning. There are again a few options when computing the operational metric. The simplest computation (OM-A) occurs when we assume that all devices have equal asset values and any exploit launched against an insecure condition detected by an attacker is always successful. The operational metric is then give by

$$\text{Option OM-A (Basic): } OM = \sum_{i \in \text{devices}} P_{\text{detect}}(i), \quad (15)$$

where the sum is over all devices with insecure conditions that can be detected by the attacker and  $P_{\text{detect}}$  is the probability of an attacker detecting the insecure condition(s) on the device. (The term  $P_{\text{detect}}(i)$  has a more complex meaning when there are multiple vulnerabilities on a device and when some of these can not be exploited with probability 1). This equation represents the expected number of devices that are compromised by an attacker. For simplicity, this is the equation that we will use when presenting equations for individual metrics.

If devices have different asset values we represent the asset values for device  $i$  as  $AV(i)$  and modify this equation to create option OM-B where

$$\text{Option OM-B (Asset Values): } OM = \frac{1}{T_{AV}} \sum_{i \in \text{devices}} AV(i) \cdot P_{\text{detect}}(i), \quad (16)$$

where the normalizing constant  $T_{AV}$  is given by

$$T_{AV} = \sum_{i \in \text{devices}} AV(i).$$

Finally, if we assume that the probability that an exploit launched by an attacker against a device with a detected insecure condition is not always successful at compromising the device,

we can specify a probability of compromise or exploit success for device  $i$  as  $P_{ExploitSuccess}(i)$  and extend Eq. 15 to produce a third option OM-C where,

$$\text{Option OM-C (Exploit Success Probability): } OM = \sum_{i \in \text{devices}} P_{ExploitSuccess}(i) \cdot P_{detect}(i). \quad (17)$$

This equation computes the probability that a device is successfully compromised as the probability of detecting the insecure condition(s) on the device times the probability that an exploit launched against the device is successful. The term  $P_{ExploitSuccess}(i)$  can depend on the details of the insecure condition(s) on the device, on the difficulty of exploiting these conditions, and on the attackers skill and competence. Options OM-B and OM-C can also be combined. The three operational metric options are summarized in Table 5.

## 15. LR-1 METRICS FOR ATTACKERS EXPLOITING UNAUTHORIZED DEVICES

Table 6 summarizes the characteristics of LR-1 as extracted from Table 1. Capability deficit metrics are computed for each subnet and attackers look for and exploit unauthorized devices such as servers, laptops, smart phones, and personal laptops. A defender must create specifications consisting of an inventory of authorized devices for each subnet and must process and remove unauthorized devices when they are discovered. Finally, the data required to compute risk using the operational metric is a list of the first and last seen times of unauthorized devices for each subnet.

**TABLE 6**

**Characteristics of LR-1 Metrics**

<b>Entity with security condition</b>	<b>Security condition exploited by attacker</b>	<b>Specification that enables security condition test</b>	<b>Data required by operational metric</b>
Subnet	Unauthorized computer systems and laptops	Authorized devices on each subnet	List of first and last seen times of unauthorized devices

The attack model for LR-1 is the threat that Critical Control 1 in [38] is designed to mitigate. It is an attacker who scans a network either internally or externally looking for unauthorized software and is assumed to opportunistically compromise any unauthorized device discovered. This threat is illustrated in Figure 25 where it can be seen that unauthorized devices include recently installed test servers, unconfigured switches, laptops that have been off the network for a long time and plugged in without being updated and patched, personal laptops that may be insecure, virtual machines (VMs), and personal devices such as smart phones or music players plugged into desktop computers. A common situation noted in [38] is for an unconfigured and insecure server to be attached to a network in the afternoon and left on the network overnight before being configured and patched in the morning. Such servers are often compromised overnight by attackers scanning networks looking for insecure servers. Note that that LR-1 does not consider devices that are deliberately installed for malicious purposes but rather devices that are benignly placed and then maliciously exploited.

All components of the process shown in Figure 26 need to be in place to compute LR-1 metrics. This process begins with creating an inventory of authorized devices shown on the upper left for each subnet. All subnets in the network are continuously scanned by the network defender for new

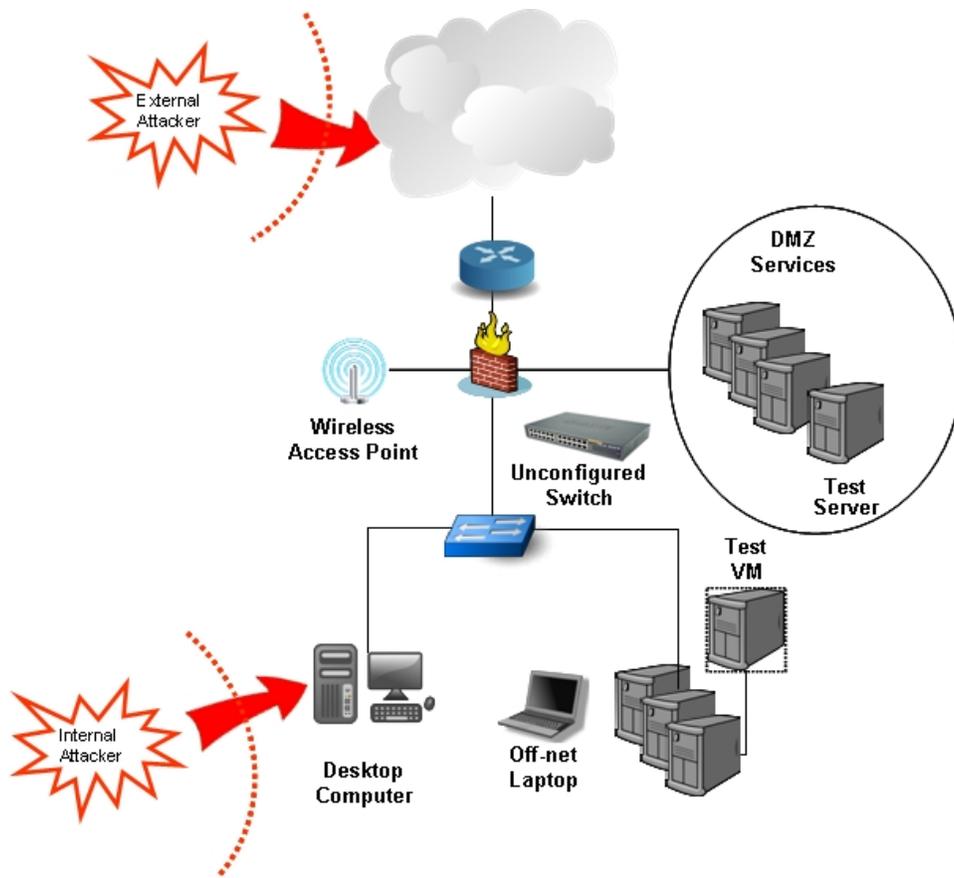


Figure 25. LR-1 attack model.

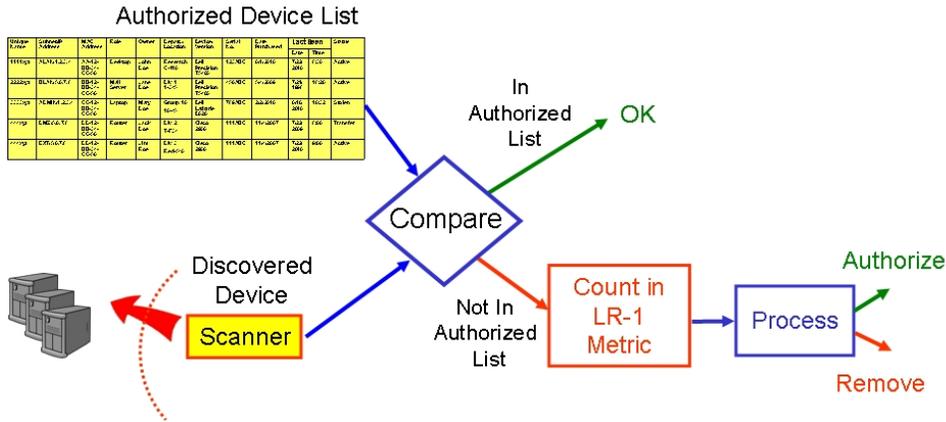


Figure 26. Processing required to create LR-1 metrics.

devices as shown on the bottom left. New devices that are discovered are compared to the list of authorized devices. Discovered devices that occur in the authorized inventory are allowed. They do not require any processing and do not affect metrics. Discovered unauthorized devices not in the authorized inventory are counted in the operational metric and passed to a remediation process that either authorizes the device or removes it. They are considered vulnerable to attack until removed from the network or authorized. The processing required for LR-1 maintains an up-to-date device inventory across all subnets that provides situation awareness of all authorized devices. This list of authorized devices is used by other metrics.

The foundation for LR-1 metrics is an inventory of authorized and unauthorized devices. Figure 27 shows an example of such an inventory. Key aspects of this inventory are that it includes a unique identifier for each device (combination of name, IP address, MAC address and subnet), its owner and role, its location, the date and time last seen by a scan, and status. In addition, there should be a scan history indicating when it was seen in the past and other information such as the type of device (e.g., is it a virtual machine, workstation, ...). For unauthorized devices the inventory should indicate when the device was first seen, when processing began, and the time when processing was complete and the device was either removed from the network or authorized. If the exact end of processing time is not available, the inventory should include the last scan time when the unauthorized device was detected. These times are required to compute the LR-1 operational metric. Detailed instructions on creating an inventory that could support LR-1 are available in [29].

### 15.1 LR-1 CHECKLIST

Assessment of the LR-1 capability deficit and operational metrics requires the following checklist items:

- **Entities:** All subnets within the enterprise must be identified and enumerated.

Unique Name	Subnet/IP Address	MAC Address	Role	Owner	Depart./ Location	Device/ Version	Date Purchased	Last Seen		Status
								Date	Time	
1111xyz	ALAN/1.2.3.4	AA-12-BB-34-CC-56	Desktop	John Doe	Research C-110	Dell Precision T5400	6/1/2010	7/22/2010	8:30	Authorized
2222xyz	BLAN/5.6.7.8	BB-12-BB-34-CC-56	Mail Server	Jane Doe	Div 1 1-345	Dell Precision T5400	5/4/2009	7/21/1998	18:20	Authorized
3333xyz	ADMIN/1.2.3.4	CC-12-BB-34-CC-56	Laptop	Mary Doe	Group 10 10-45	Dell Latitude D620	2/2/2010	6/10/2010	16:32	Stolen
444xyz	DMZ/5.6.7.8	DD-12-BB-34-CC-56	Router	Jack Doe	Div 2 T-F34	Cisco 2900	11/4/2007	7/22/2009	8:00	Quarantined
444xyz	EXT/5.6.7.8	DD-12-BB-34-CC-56	Router	Jim Roe	Div 3 Red-546	Cisco 2900	11/4/2007	7/22/2010	9:00	Authorized

Figure 27. Example inventory of authorized and unauthorized devices for LR-1.

- **Specification:** An inventory of all authorized devices permitted on each subnet must be created and maintained.
- **Monitoring:** Each subnet must be continuously scanned to find recently installed devices, to compare them to the inventory of authorized devices, and thus to detect and keep an inventory of unauthorized devices. Our analyses support many types of scanning including periodic scanning as well as instantaneous sampling as described in Section 10.
- **Resolution process:** A process must be in place to remove, isolate, or authorize any unauthorized devices detected and to maintain the the first detection times and end of the processing times for all authorized devices in the inventory of unauthorized devices.
- **Maintenance process:** A process must be in place to add new authorized devices to the inventory and to de-authorize devices as they are removed from the inventory.

In these checklist metric components we assume that the same technique (e.g., scanning or automatic detection of new MAC addresses) is used to detect all unauthorized devices on each subnet. Different subnets may be scanned at different rates or use different device discovery procedures. To define a subnet, we first define a reachability group as a collection of IP addresses that are treated identically by rules in filtering devices. For example, hosts in a DMZ and normal user desktops may be in different reachability groups. We then define subnets as collections of one or more reachability groups that use the same mechanism to discover unauthorized hosts and verify authorized hosts. We will perform a separate mathematical analysis to compute the probability of not detecting an unauthorized device present for a given length of time in each subnet. We will also determine coverage based on the number of subnets where unauthorized devices are detected.

## 15.2 LR-1 CAPABILITY DEFICIT METRIC

Specification requirements for LR-1 as shown in Table 1 are that a per-subnet inventory of authorized devices is required. The **specification deficit** is thus

$$SD_{\text{subnet}}(i) = 1 - I_{\text{inventory\_specified}}(i),$$

where  $I_{\text{inventory\_specified}}(i)$  is an indicator function that is 1 if a device inventory is available for subnet  $i$  and 0 otherwise.

The **timeliness deficit** for LR-1 is derived from both the sampling rate and coverage information. It is the probability that the defender misses an unauthorized device present for a minimum target duration window  $W_t$  given a defender sampling interval  $\delta_d(i)$  in subnet  $i$  for all subnets where there is sampling and they are thus covered. Using Equation 2, this is given by

$$TD_{\text{subnet}}(i) = P_{\text{miss}}(i) = \max\left(0, 1 - I_{\text{covered}}(i) \cdot \frac{W_t}{\delta_d(i)}\right).$$

In this equation, the term  $I_{\text{covered}}(i)$  is another indicator function that is 1 if subnet  $i$  is being scanned to discover devices and 0 otherwise. To simplify notation, for other metrics, we will express the timeliness deficit as

$$TD_{\text{subnet}}(i) = P_{\text{miss}}(i) = \max\left(0, 1 - \frac{W_t}{\delta_d(i)}\right), \quad (18)$$

where, by convention, a subnet that is not covered and not sampled will be assigned an infinitely long sampling interval ( $\delta_d = \infty$ ) with the result that  $\frac{W_t}{\delta_d} = 0$  and  $P_{\text{miss}}(i) = 1$  as when  $I_{\text{covered}}(i)$  was used and set to 0. In this way the timeliness deficit assesses both timeliness and coverage.

The **total testing error** for LR-1 is an advanced feature that can be set to 0 as an assumption of accurate testing if no additional information on the accuracy of tests used to determine the presence of an unauthorized device is available. It can also be set based on 1) Actual tests performed by attaching unauthorized and authorized devices, 2) An analysis of testing assumptions and how testing systems work, or 3) A subjective comparison between different testing methods.

The **accuracy deficit** is defined as the complementary product of the timeliness deficit and the total testing error:

$$AD_{\text{subnet}}(i) = 1 - (1 - TD_{\text{subnet}}(i)) \cdot (1 - TTE_{\text{subnet}}(i)).$$

The total capability deficit metric for each subnet can be computed using the default motivational option from Eq. 11 or the rigorous option from Eq. 12. Using the default motivational option results in

$$CDM_{\text{subnet}}(i) = \frac{SD_{\text{subnet}}(i) + 3 \cdot AD_{\text{subnet}}(i)}{4}.$$

To compute the overall capability deficit network we can weight the importance of each subnet as shown in Eq. 13. If no other information is available on the importance of different subnets, we recommend the following weighting function:

$$S_{\text{subnet}}(i) = \frac{(\# \text{ of assigned devices on } i) + 40}{40}.$$

This equation does not simply count the number of devices on a subnet, but is designed to prevent small subnets such as DMZs that contain a few servers from being assigned negligible importance. The constant value of 40 in the equation indicates that the value of a subnet without any devices is equivalent to that of 40 devices or roughly 20% of the value of a fully populated class C network. This value can be changed, if necessary. More complex subnet weightings are also possible given knowledge of the asset value of devices and the importance of each subnet in an enterprise. Given these or other weightings, the overall capability deficit metric is computed as shown in Eq. 13 as

$$CDM_1 = \frac{1}{T_{\text{subnets}}} \sum_{i \in \text{subnets}} S_{\text{subnet}}(i) \cdot CDM_{\text{subnet}}(i), \quad (19)$$

where

$$T_{\text{subnets}} = \sum_{i \in \text{subnets}} S_{\text{subnet}}(i).$$

This total capability deficit metric determines whether the observe step of the OODA-like loop shown in Figure 4 is accurate enough to support accurate risk estimation and security improvements. This metric needs to be low enough (e.g., below 0.2) before operational metrics can be computed accurately. The target device duration ( $W_t$  in Eq. 18) can be decreased over time for motivational purposes to make the capability deficit easy to reduce to low values initially but then to lower the duration of the minimum duration insecure condition that must always be detected over time to improve security. This decrease needs to be performed consistently across all networks that use this metric to allow fair comparisons. The target duration used also needs to be noted explicitly whenever the metric is presented.

### 15.3 LR-1 OPERATIONAL METRIC

Each identified unauthorized device  $i$  has a probability of being compromised based upon the time window it is present and available for compromise  $w_i$  and the attacker scanning rate  $\Delta_a$ . The operational metric is the expected number of compromised devices as described in Eq. 15:

$$OM_1(\Delta_a) = \sum_{i \in \substack{\text{unauthorized} \\ \text{devices}}} P_{\text{detect}}(i), \quad (20)$$

where the probability of the attacker detecting and compromising device  $i$  is given as

$$P_{\text{detect}}(i) = \min\left(1, \frac{w_i}{\Delta_a}\right) \quad (21)$$

as in Equation 1. This metric is affected by both the number of unauthorized devices being placed on the network and the efficiency of their removal via the process in Figure 26.

To motivate improvement in detecting and processing unauthorized devices, the attacker sampling interval  $\Delta_a$  in Eq. 21 can be decreased over time to make it easier to initially achieve a lower score, but then to make the network more secure over time as part of the capability/maturity approach. This decrease again needs to be performed consistently across all networks that use this metric to allow fair comparisons. The attacker sampling interval used also needs to be noted explicitly whenever the metric is presented.

**This page intentionally left blank.**

## 16. LR-2 METRICS FOR ATTACKERS EXPLOITING UNAUTHORIZED SOFTWARE

Table 7 summarizes the characteristics of LR-2 as extracted from Table 1. Capability deficit metrics are computed for all authorized devices identified by LR-1. Attackers look for and exploit unauthorized software found on devices including desktop workstations, laptops, smart phones, and tablets.

TABLE 7

Characteristics of LR-2 Metrics

Entity with security condition	Security condition exploited by attacker	Specification that enables security condition test	Data required by operational metric
Device	Unauthorized software	Unauthorized and authorized software on each device	List of first and last seen times for each unauthorized software package on each device

A defender must create specifications consisting of lists of authorized and unauthorized software and process software packages using these lists when they are discovered. Lists of both authorized and unauthorized software are used because new software packages, not on the authorized list, are often discovered. If these discovered software packages are on the unauthorized list, they can be immediately classified as unauthorized instead of being put on a list of unclassified software for further analysis. The data required to compute risk using the operational metric is a list of the first and last seen times of unauthorized software packages on each device.

The attack model for LR-2 is the threat that Critical Control 2 in [38] is designed to mitigate. It expands the attacker model of LR-1 by including both server-side and client-side attacks. Server-side attacks shown on the left of Figure 28 consist of external or internal attackers who scan a network looking for unauthorized server software running on ports of a computer that can be reached from the attacker's location. Server software could include web-servers, database servers, email servers, chat servers, and other type of server software. These attacks are similar to those in LR-1 except the attacker is looking for unauthorized servers instead of unauthorized devices. We assume that unauthorized server software is insecure and will be compromised if discovered.

Client-side attacks shown in the right of Figure 28 have a few more stages and variations. As described above in Section 10.2, in a client-side attack, an attacker either inserts malware on legitimate web servers or directs a user using email, social media, or other approaches to browse to

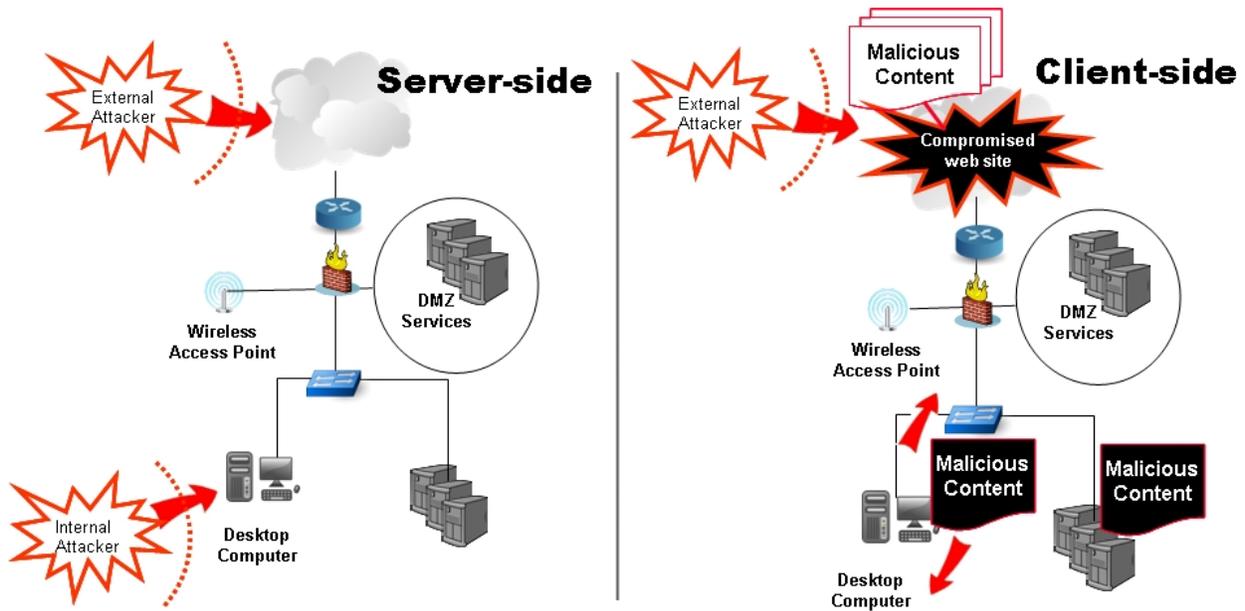


Figure 28. The LR-2 attack model includes server-side attacks (left) and client-side attacks (right).

a malicious web site that contains embedded malware. Alternatively an attacker can coerce a user to download malicious content that will run an exploit when the application required to open the content runs. In either case, when a user clicks on the infected web page or opens the infected file, malicious content is executed that exploits vulnerabilities in the client browser or helper applications used to display images, play movies, edit documents, or perform other functions. This approach can be extremely effective because there are so many helper applications and so many vulnerabilities in these applications and browsers. As with server-side attacks, we will assume that unauthorized client software will be exploited and compromised if present on a device. As noted, unauthorized client software may include web browsers, components of web browsers, document readers, movie and image display software, and other client software. Metric computations used for LR-2 are more complex than in LR-1 because these two attack types have different attacker sampling rates. Note that LR-2 does not consider software that is deliberately installed for malicious purposes but rather software that is benignly placed and then maliciously exploited.

All components of the process shown in Figure 29 need to be in place to compute LR-2 metrics. This process begins on the upper left with creating lists of authorized software, unauthorized software, and exceptions. Software on all authorized devices is then continuously scanned and reported by software agents running on devices as shown on the bottom left. These agents periodically create a list of all software packages on devices. Observed software packages are compared to the lists of authorized and unauthorized software. Authorized software is allowed. Unauthorized software is counted in the operational metric and passed to a remediation process that either authorizes the software or removes it. A device is considered vulnerable to attack until its unauthorized software is removed or authorized. If software is not on the authorized or unauthorized list, it is logged and

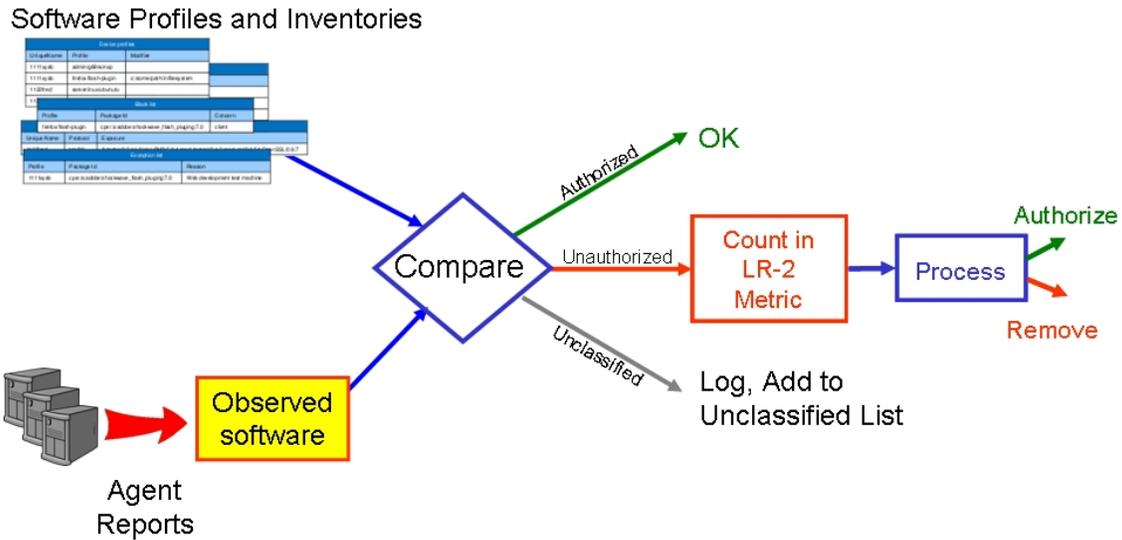


Figure 29. Processing required to create LR-2 metrics.

added to the unclassified software list for further analysis. This list needs to be analyzed frequently to keep it small. LR-2 processing maintains an up-to-date software inventory across all subnets that provides situation awareness of all software packages. This software list is used by other metrics.

LR-2 requires a collection of software profiles for each device that is used to build lists of authorized and unauthorized software. It is assumed that each device has a role or collection of roles that it serves within an organization. Performing these roles requires the software package or packages associated with each role. These software packages should thus be authorized for a device. For example, the upper table in Figure 30 lists the profiles or roles for two different devices. The device named “111xyab” serves as a Windows XP administrator and needs a Flash Player plug-in for the Firefox browser. The device named “1122fred” supports an Apache web server running under Gentoo Linux server software. The middle table in Figure 30 lists the software packages allowed for the Gentoo Linux server, for the Apache web server, and for the Firefox Flash plug-in. Software packages are named in this table using the Common Processing Enumeration (CPE) conventions [26, 46] but other naming standards could be used such as cryptographic hash values for program executables. The first four rows of content in the middle table of Figure 30 list some of the authorized software for profiles used by the device named “1122fred” and the last row lists some of the authorized software for the Firefox/flash-plug-in profile used by the device labeled “1111xyab.” The bottom table in Figure 30 indicates that Java plug-in software for Firefox browsers is unauthorized and that it enables a client-side attack.

Software packages that are not needed should be unauthorized because they can be used by attackers to compromise a device or because they are restricted for policy reasons. Policies could include such restrictions as not allowing peer-to-peer file sharing or real-time video streaming. In the remainder of this description we focus on software that is unauthorized because it allows attackers

Device profiles		
Unique Name	Profile	Modifier
1111xyab	admin/g68/winxp	
1111xyab	firefox/flash-plugin	c:/some/path/in/filesystem
1122fred	server/g68/gentoo	
1122fred	webserver/apache2	/var/www/path

Authorized list	
Profile	Package Id
server/g68/gentoo	cpe:/a:isc:bind:9.2.9
server/g68/gentoo	cpe:/a:gentoo:clibc:2.5:r3
server/g68/gentoo	cpe:/a:gentoo:security-agent:1.0.0
webserver/apache2	cpe:/a:apache:cxfs:2.2.11
firefox/flash-plugin	cpe:/a:adobe:shockwave_flash_pluging:7.0

Unauthorized list		
Profile	Package Id	Concern
Firefox-java-plugin	cpe:/a:sun:java_plug-in-for-mozilla-browsers:1.6	client

Figure 30. Device profiles and inventories of authorized and unauthorized software required by LR-2.

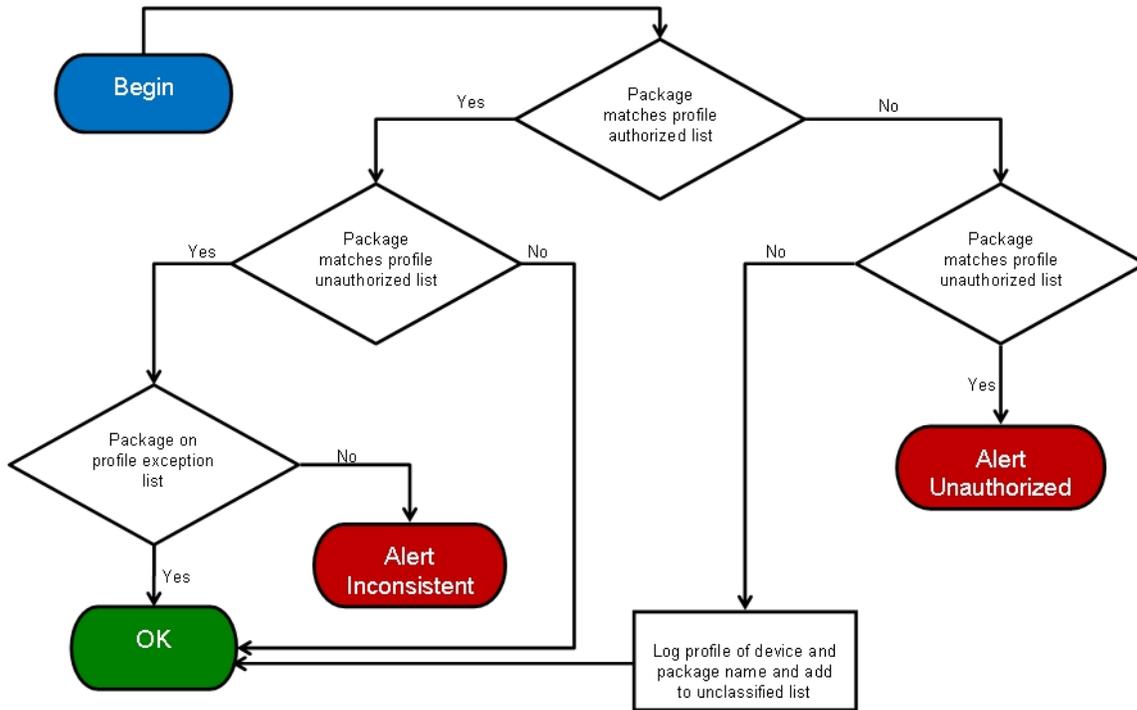


Figure 31. Package classification flowchart for LR-2.

to compromise a host. We assume that the inventory contains information for each unauthorized software package that indicates whether it facilitates a client-side attack, a server-side attack, or whether it is restricted due to policy guidelines.

The flowchart in Figure 31 provides details of how packages could be classified as authorized, unauthorized, or added to the unclassified list. It allows an exception list to handle packages that are allowed because they serve an essential function and their security risk is acceptable. Software packages on the exception list are assumed to also be on the authorized and unauthorized list. The flowchart shows when unauthorized software is detected (middle right), when authorized software is allowed (bottom left), and when software is added to the unclassified list (bottom right). The log of unclassified device profiles and package names shown on the bottom right is used to generate the list of unclassified packages.

Creating and maintaining the authorized and unauthorized software package lists and keeping the size of the unclassified list small can be supported by the process shown in Figure 32. The process begins by collecting reports from all device agents to obtain a comprehensive list of all software packages. Those packages that are vulnerable as noted in the National Vulnerability Database should be (explicitly) unauthorized. Other packages discovered by system observation need to be manually analyzed and compared to security and other policies to determine whether they belong

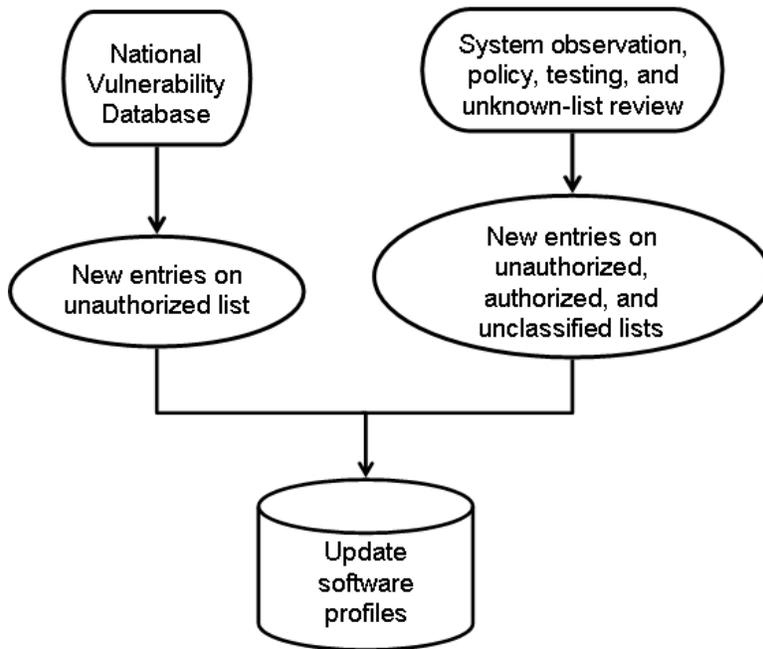


Figure 32. Process to create and maintain lists of authorized and unauthorized software.

on the authorized or unauthorized lists. The automated creation of the list of unclassified packages shown in Figure 29 feeds the manual process of creating the authorized list essentially by answering the question “does this package need to be installed on a device of a particular profile?”. The automatic creation of an inconsistent list feeds the manual process of creating an exception list by answering the questions such as “Why should this unauthorized package be permitted? Who is responsible? How long is it necessary?”

## 16.1 LR-2 CHECKLIST

Assessment of CC-2 capability deficit and operational metrics requires the following checklist items:

- **Entities:** LR-1 should be implemented to provide an enumeration of authorized devices and an enumeration of all software packages on each device should be created using local software agents and CPE nomenclature. If LR-1 is not fully implemented, LR-2 may be used for those devices that have been specified as authorized by LR-1.
- **Specification:** A software profile should be created for each authorized system that lists the software authorized and often unauthorized for that system. Each known device/software package needs to be designated as authorized, unauthorized, or unclassified. Additionally, the attack type enabled by the unauthorized software should be specified as client-side or

server-side. Different treatment of the two attack types is required because they normally have different attacker sampling intervals.

- **Monitoring:** Each device is monitored by a local agent to generate a list of installed software packages. This is compared to the list of authorized, unauthorized, or unclassified software packages.
- **Resolution process:** A process exists to resolve discrepancies between observed and authorized software packages and either remove or authorize each observed unauthorized package.
- **Maintenance process:** A process exists to classify software as authorized or unauthorized as it is discovered and a process exists to ensure that currently authorized software should remain authorized (e.g., by analyzing software packages using information in the National Vulnerability Database). A list of exceptions should also be maintained for software packages and devices.

## 16.2 LR-2 CAPABILITY DEFICIT METRIC

Specification requirements for LR-2 are that a profile of authorized and unauthorized software packages must exist for each device. The specification deficit is thus

$$SD_{device}(i) = 1 - I_{profiled}(i),$$

where  $I_{profiled}(i)$  is an indicator function that is 1 if software profiles exist for device  $i$  and 0 otherwise.

The **timeliness deficit** expresses the probability of missing a package present for minimum target duration window  $W_t$  given a scan rate of  $\delta_d(i)$  as

$$TD_{device}(i) = P_{miss}(i) = \max\left(0, 1 - \frac{W_t}{\delta_d(i)}\right). \quad (22)$$

The **total testing error** should ideally use a complementary product of two terms. The first term is the probability that when a software package is on the authorized or unauthorized list and it is classified by a software agent, the agent makes a mistake in the classification. We will assume this is 0. It can also be measured as discussed above. The second term is the probability that a software package on a device is unclassified  $P_{unclassified}(i)$  as measured by the fraction of unclassified software packages on a device. This term is motivated by considering the resultant accuracy of software package classification when all the unclassified software packages are unauthorized. This would incur an error similar to that caused when the agent accuracy for the first term is equal to  $P_{unclassified}(i)$ . We will thus assume that the total testing error is given by

$$TTE_{device}(i) = P_{unclassified}(i) = \text{Fraction of Unclassified Software on Device } i.$$

The **accuracy deficit** is defined as the complementary product of the timeliness deficit and the total testing error:

$$AD_{device}(i) = 1 - (1 - TD_{device}(i)) \cdot (1 - TTE_{device}(i)).$$

Computing the the capability deficit metric for each device using the motivational option from Eq. 11 yields

$$CDM_{device}(i) = \frac{SD_{device}(i) + 3 \cdot AD_{device}(i)}{4}.$$

For the purposes of computing the overall CDM, we can weight each device with a factor  $S_{device}(i)$  and compute the overall capability deficit metric using Eq. 13 as

$$CDM_2 = \frac{1}{T_{devices}} \sum_{i \in devices} S_{device}(i) \cdot CDM_{device}(i), \quad (23)$$

where

$$T_{devices} = \sum_{i \in devices} S_{device}(i).$$

As with LR-1, the overall capability deficit metric needs to be low before operational metrics can be computed accurately and the target device duration ( $W_t$  in Eq. 22) can be decreased over time to increase accuracy as overall system security increases.

### 16.3 LR-2 OPERATIONAL METRIC

The operational metric assesses risk as the probability that a device will be compromised due to unauthorized software either via client-side or server-side attack. Risk computation is identical to that used in LR-1 when the windows of vulnerability or intervals when unauthorized software packages are present are non-overlapping and disjoint over time. We use some simplifying assumptions to calculate risk when there are many unauthorized software packages on a device with overlapping windows of vulnerability. This situation is illustrated in Figure 33. The horizontal axis of this figure represents the total extent of the measurement interval where risk is computed. The vertical axis represents types of unauthorized software. The upper half of Figure 33 represents unauthorized server software that provides services by opening network ports on a device. The only unauthorized server software shown is one software package labeled “A” that is present for a short time interval. As shown on the uppermost part of Figure 33 this one package simply creates an identical window

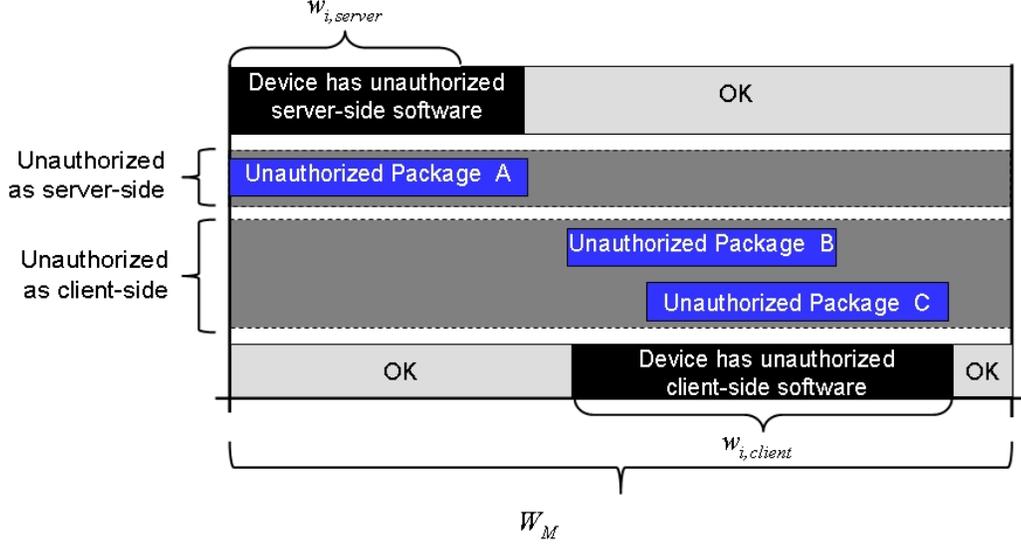


Figure 33. Combining vulnerability windows for unauthorized client and server software to assess risk.

of vulnerability that is used to compute risk from server-side attacks. The lower half of the figure represents unauthorized software that provides client-side services. Here, there are two pieces of unauthorized software labeled “B” and “C” that have overlapping windows of vulnerability. Instead of treating each overlapping window and software package separately, we combine them to form one total window of vulnerability. This creates the single window of vulnerability shown on the bottom of Figure 33 that is used to compute the risk from client-side attacks. We combine windows of vulnerability for software packages for simplicity, because modern attackers generally exploit many vulnerabilities, and because it is currently difficult to rate the exploitability of different types of unauthorized software. Our core assumption is that the probability of host compromise with any single unauthorized software package is 1. Providing multiple unauthorized software packages to an attacker thus does not increase the probability of compromise, but it always remains at 1.

To compute the operational metric, we introduce some new notation. The subscripts “client” and “server” indicate client-side and server-side attackers and the sample intervals for these two attack types are indicated by  $\Delta_{a,client}$  and  $\Delta_{a,server}$ . The probability of an attacker detecting windows where unauthorized software exists on client and server software are  $P_{detect,client}(i)$  and  $P_{detect,server}(i)$  and the windows where attackers can detect unauthorized software are  $w_{client}(i)$  and  $w_{server}(i)$  where overlapping windows are combined as shown in Figure 33. The resulting operational metric is the expected number of devices that will be compromised via server-side *or* client-side attacks due to the presence of unauthorized, hence vulnerable, software

$$OM_2(\Delta_{a,client}, \Delta_{a,server}) = \sum_{i \in \text{Profiled Devices}} P_{\text{detect}}(i), \quad (24)$$

where  $P_{\text{detect}}(i)$  is defined by the complementary product

$$P_{\text{detect}}(i) = 1 - (1 - P_{\text{detect},\text{client}}(i)) \cdot (1 - P_{\text{detect},\text{server}}(i)). \quad (25)$$

Using Equation 1,

$$P_{\text{detect},\text{client}}(i) = \min\left(1, \frac{w_{\text{client}}(i)}{\Delta_{\text{a},\text{client}}(i)}\right),$$

and

$$P_{\text{detect},\text{server}}(i) = \min\left(1, \frac{w_{\text{server}}(i)}{\Delta_{\text{a},\text{server}}(i)}\right).$$

As with LR-1, this metric is affected by both the number of unauthorized software packages and the efficiency of their removal via the process in Figure 29. To motivate improvement in detecting and processing unauthorized software, the attacker sampling intervals ( $\Delta_{\text{a},\text{client}}, \Delta_{\text{a},\text{server}}$ ) can also be decreased over time to make it easier to initially achieve a lower score, but then to make the network more secure over time as part of the capability/maturity approach. The client and server attacker sampling intervals used also need to be noted explicitly whenever the metric is presented.

## 17. LR-3 METRICS FOR KNOWN VULNERABILITIES

The third Lincoln Risk metric concerns known vulnerabilities. Table 8 summarizes the characteristics of LR-3 as extracted from Table 1. Capability deficit metrics are computed for an attacker who attempts to exploit known server-side or client-side vulnerabilities on each device in an enterprise. The adversary in this metric corresponds to the threat for Critical Control 10, “Continuous Vulnerability Assessment and Remediation” in [38].

**TABLE 8**

**Characteristics of LR-3 Metrics**

<b>Entity with security condition</b>	<b>Security condition exploited by attacker</b>	<b>Specification that enables security condition test</b>	<b>Data required by operational metric</b>
Device	Known software vulnerabilities	List of known vulnerabilities on each device	List of first and last seen times and score for each vulnerability on each device

We focus on two fundamental mechanisms to detect vulnerabilities. The first is maintaining an inventory of software installed on each device, as in LR-2, and using a vulnerability database to lookup vulnerabilities from software package names. The second is to actively scan each device either remotely or locally using a collection of vulnerability signatures.

From the standpoint of the metrics there are two important differences between these mechanisms. The first concerns the coverage of vulnerabilities. LR-2 provides a software inventory whose coverage tends to include all officially installed packages even if a specific vulnerability signature does not exist. By contrast, signature based scanning might detect illicitly installed packages for which signatures exist. The second difference is the speed with which updated vulnerability knowledge, via a database or signature set, leads to knowledge of vulnerability instance on the device base. In the case of the database, this knowledge is essentially instantaneous. In the case of the signature set, devices need to be rescanned. This said, direct scanning for vulnerabilities is often also necessary as some vulnerabilities are dependent upon run-time parameters and thus must be live tested as different instances of the same package have different vulnerability profiles. The capability deficit metric allows various forms of vulnerability detection and allows for fusion of multiple information channels.

## 17.1 LR-3 CHECKLIST

Assessment of LR-3 capability deficit and operational metrics requires the following checklist items:

- **Entities:** LR-1 and LR-2 must be implemented to provide an inventory of devices and software packages. The LR-2 database should be extended to specify which vulnerabilities should be tested for each authorized device of the LR-1 database and for which concern they are being tested (client-side versus server-side).
- **Monitoring:** A vulnerability scanning process must exist. Ideally, it should use both remote scanning and local agent-based data collection as well as the device software inventory of LR-2 to associate vulnerabilities to software packages using a vulnerability database.
- **Resolution process:** A vulnerability remediation process exists, such as patching or removing software.
- **Maintenance process:** A process exists to update the two data sources used for monitoring: the vulnerability signatures and the vulnerability database.

## 17.2 LR-3 CAPABILITY DEFICIT METRIC

The **specification deficit**

$$SD_{\text{device}}(i) = 1 - I_{\text{profiled}}(i)$$

of LR-3 requires that each authorized device has a specified list of vulnerabilities for which it should be scanned. These should be specified using vulnerability assignments from the Common Vulnerability Enumeration (CVE) dictionary [34].

For a given device with a specification, we estimate the probability that each vulnerability will be missed due to timeliness, coverage or test error. LR-3 allows for multiple detection modalities, initially focusing on remote scanning and inventory-based discover using a vulnerability database.

For each vulnerability test and detection modality, we compute the timeliness deficit and testing error for that combination. We define the indicator function  $I_{\text{detectable,modality}}(j)$  which is 1 if vulnerability  $j$  is detectable via modality **modality** and 0 otherwise. As a concrete example, a client-side vulnerability in a browser is not typically detectable via remote scanning while it is detectable via a local agent. As such, the indicator function would have value 0 for remote scanning and value 1 for a local agent.

The **timeliness deficit** for a test  $j$  on device  $i$  is then given by

$$TD_{\text{modality}}(i, j) = P_{\text{miss,modality}}(i, j) = \min_{\text{modality}} \left[ 0, 1 - I_{\text{detectable,modality}}(j) \cdot \frac{W_t}{\delta_d(i)} \right].$$

In this equation, the  $\min()$  function selects the modality with the lowest probability of miss. This could be remote scanning for some vulnerabilities and inventory-based analysis for others.

The **total testing error**, in addressing inaccuracy and incompleteness, includes a term that models the failure to update security signatures/vulnerability information in the scanner used to detect vulnerabilities. For simplicity, if we assume that the measured misclassification error is zero, then the total testing error can be modeled as the probability of missing an update for each detection modality  $X$  as as in Equation 2:

$$TTE_X = \max \left( 0, 1 - I_{\text{detectable},X}(j) \cdot \frac{W_{t,\text{update},X}}{\delta_{\text{update},X}(i)} \right).$$

In this equation,  $\delta_{\text{update},X}(i)$  is the actual time between updating signature files for the vulnerability scanner and modality  $x$  while  $W_{t,\text{update},X}$  is the actual update interval for the vulnerability scanner. This term penalizes signature update intervals that are longer than the actual vendor signature update intervals and makes sure signatures exist for all vulnerabilities. It is similar to the total testing error term for LR-2 that penalizes sites where there are too many unclassified software packages.

Finally, different vulnerabilities have different severity ratings and should be weighted accordingly. Ultimately, we would like to assign a weight  $S_{\text{cve}}(j)$  to vulnerability  $j$  as the probability that a device will fail to satisfy is specified function when it is exploited, where this includes loss of confidentiality, integrity, and availability. To simplify metric application, we estimate these weights from the base metric score provided by the Common Vulnerability Scoring System (CVSS) [23]. The base CVSS score “captures the characteristics of a vulnerability that are constant with time and across user environments” and includes components that measure where an attacker needs to be located to exploit the vulnerability, the exploit complexity, the number of times the attacker needs to provide authentication, and the level of compromise provided. The base score combines scores for these components to produce a number ranging from 0 to 10.

We analyzed the distribution of CVSS scores for 2009 and 2010 and found few low values below 4, even though vulnerabilities with these low scores should result in much lower probability of compromise compare to vulnerabilities with scores of 10. We thus applied a simple monotonic normalization to CVSS scores so they span a greater range and result in a value ranging from 0 to 1.0 that can be interpreted as a probability of compromise. We convert CVSS scores to probability of compromise using the following formula

$$s_{\text{cve}}(j) = \left( \frac{\text{cvss}(j)}{10} \right)^2.$$

This normalization results in a roughly uniform distribution of probabilities from 0 to 1.0 based on the CVE scores from 2009 and 2010. This simple approach could be improved if more detailed information on vulnerabilities and exploits were available.

The **accuracy deficit** is computed across tests as

$$AD_{\text{device}}(i) = \frac{1}{T_{\text{CVE}}(i)} \sum_{j \in \text{CVE}(i)} S_{\text{CVE}}(j) \cdot \min_{x \in \{\text{modalities}\}} [1 - (1 - TD_{3,x}) \cdot (1 - TTE_{3,x})],$$

where  $T_{\text{CVE}}(i) = \sum_{j \in \text{CVE}(i)} S_{\text{CVE}}(j)$ . Note that the  $S_{\text{CVE}}$  terms serves to focus attention on the most critical vulnerability and the min term selects the best means of detection.

### 17.3 LR-3 OPERATIONAL METRIC

The **Operational Metric** assesses risk as the probability that a device will be compromised via client-side or server-side attack. To perform this computation we fuse remote and local vulnerability detection data concerning a device to form a set of beliefs about the collection of vulnerabilities on that device as illustrated in Figure 34.

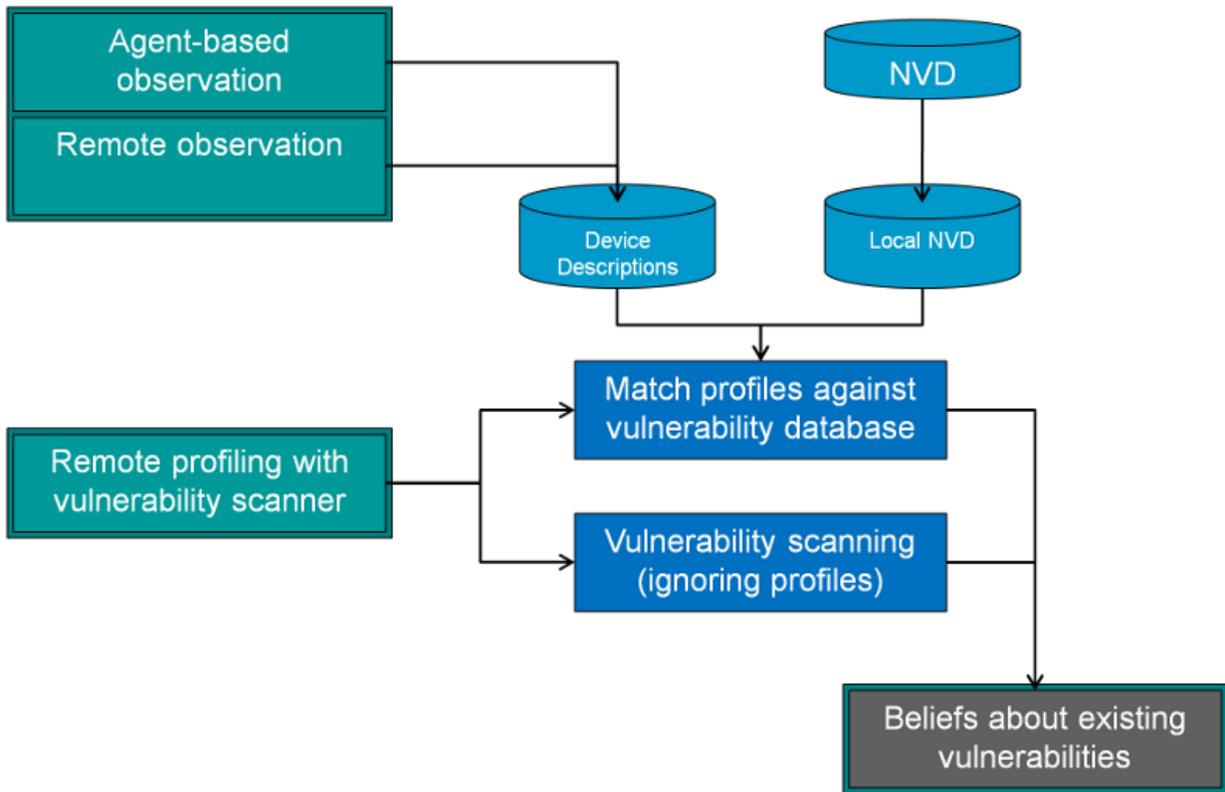


Figure 34. Vulnerability detection process for LR-3.

For each vulnerability we compute the probability that a device  $i$  will fail from a vulnerability  $v$  as the product of it failing should that vulnerability be exploited times the probability that it will be detected (and exploited) by the attacker

$$P_{\text{fail}}(v, i) = P_{\text{vuln}}(v) \cdot P_{\text{detect}}(v, i).$$

We set the probability that a device  $i$  will fail due to a vulnerability as  $P_{\text{vuln}}(v, i) = S_{\text{CVE}}(v)$  (based upon the analysis of the existing CVSS corpus). The probability of detection is derived from Eq. 1 as  $P_{\text{detect}}(v, i) = \min\left(1, \frac{W_i(V)}{\Delta_{\text{detect}}(v)}\right)$ .

A single device may have numerous vulnerabilities leading to a collection of probabilities of failure  $\{P_i\}$ . We combine these individual probabilities of failure using a complementary product  $1 - \prod(1 - P_i)$  corresponding to the attack model where the attacker tries all exploits. This results in a final operational metric

$$OM_3 = \sum_{i \in \{\text{devices}\}} S_{\text{device}}(i) \cdot \left(1 - \prod_{v \in V(i)} [1 - P_{\text{fail}}(v, i)]\right),$$

which is the expected number of compromised devices stemming from exploitation of known vulnerabilities.

**This page intentionally left blank.**

## 18. LR-4 METRICS FOR MISCONFIGURATIONS

The fourth Lincoln Risk metric concerns securely configured systems. It corresponds to both Critical Control 3 (Secure Configurations for Hardware and Software on Laptops, Workstations, and Servers) and Critical Control 4 (Secure Configurations for Network Devices such as Firewalls, Routers, and Switches) in SANS Institute [38]. Although these two Critical Controls address different classes of device, the same basic threat model and hence metric applies. This metric assumes an attacker who attempts to exploit misconfigurations on all devices in an enterprise. Table 9 summarizes the characteristics of LR-4 as extracted from Table 1.

TABLE 9

Characteristics of LR-4 Metrics

Entity with security condition	Security condition exploited by attacker	Specification that enables security condition test	Data required by operational metric
Devices	Misconfigurations	List of correct configurations on each device	List of first and last seen times and score for each misconfiguration on each device

Metrics concerning this attack type require a specification of which misconfigurations are of concern for each device. This specification should be build upon the Common Configuration Enumeration (CCE) specification [39].

### 18.1 LR-4 CHECKLIST

Assessment of LR-4 capability deficit and operational metrics requires the following checklist items:

- **Entities:** LR-1 and LR-2 should be implemented to provide an inventory of devices and software packages. The LR-2 database should be extended to specify which configuration checks and expected results should be specified for each authorized device along with for which concern they are being tested (client-side versus server-side).
- **Monitoring:** A misconfiguration scanning process must exist.
- **Resolution process:** A misconfiguration remediation process exists, such as changing configuration options, patching, or removing software.

- **Maintenance process:** A process exists to maintain appropriate configurations.

Note that the CCE standard does not attempt to specify best practices or policies but rather enumerates topics that should be addressed by best practices or policies. As such, specifications of expected values must be written so that in cases where a riskier option is permitted is is permitted rather than required. For example, CCE-4006-3 is “The USB device support module should be installed or not as appropriate” without specifying what is appropriate. If the module is permitted for a certain class of work stations, the policy should state that the acceptable values are “installed” or “not installed.”

## 18.2 LR-4 CAPABILITY DEFICIT METRIC

The **specification deficit**

$$SD_{\text{device}}(i) = 1 - I_{\text{profiled}}(i),$$

of LR-4 requires that each authorized device has a specified list of configuration items that should be checked along with permissible values. As with vulnerabilities, different configuration checks vary in importance.

Unlike vulnerabilities, there is not a significant corpus of existing, scored configuration checks. We thus nominally adopt a normalization

$$S_{\text{CCE}}(j) = \left( \frac{ccss(j)}{10} \right)^2$$

based on the assumption that CCSS behaves somewhat like CVSS.

Unlike vulnerability testing, almost all configuration testing is done directly on the device. As such, LR-4 is simpler than LR-3 in that there is no need to combine different information sources for a single device.

The **timeliness deficit** for a test  $j$  on device  $i$  is then given by

$$TD(i, j) = P_{\text{miss}}(i, j) = \max \left( 0, 1 - \frac{W_t(i, j)}{\delta_d(i, j)} \right).$$

The **total testing error** of a test  $j$  on device  $i$  LR-4 will assumed to be zero, but could be determined empirically. This results in an **accuracy deficit** of

$$AD(i) = \frac{1}{T_{\text{CCE}}(i)} \sum_{j \in CCE(i)} s_{\text{CCE}}(j) [1 - (1 - TD_4(i, j)) (1 - TE_4)],$$

were  $T_{\text{CCE}}(i) = \sum_{j \in CCE(i)} S_{\text{CCE}}(j)$ .

### 18.3 LR-4 OPERATIONAL METRIC

The operational metric for LR-4 follows the structure of LR-3. We compute a probability that a device  $i$  will fail due to a misconfiguration  $j$  as

$$P_{\text{fail}}(j, i) = P_{\text{vuln}}(j) \cdot P_{\text{detect}}(j, i),$$

where  $P_{\text{vuln}}(j, i) = S_{\text{CCE}}(j)$ . The probability of detection is  $P_{\text{detect}}(j, i) = \min\left(1, \frac{W_i(V)}{\Delta_{\text{detect}}(v)}\right)$ . As with LR-3, a single device may have numerous misconfigurations. These are combined using a complementary product to produce the operational metric

$$OM_4 = \sum_{i \in \{\text{devices}\}} S_{\text{device}}(i) \cdot \left(1 - \prod_{v \in V(i)} [1 - P_{\text{fail}}(v, i)]\right),$$

which is the expected number of compromised devices stemming from misconfigurations.

**This page intentionally left blank.**

## 19. LIMITATIONS

The new critical control metrics have a number of limitations. First, we focus only on technical controls that can provide continuous and automated risk assessment. We purposefully do not try to assess the effectiveness of security training or policy compliance when this has no direct effect on a specific attack or when automated measures are impossible.

Second, we use simple attacker models. We assume attackers have exploits for all vulnerabilities, that they use all exploits, and that they can be characterized by the interval between scans that search for insecure conditions. We could adjust the interval between attacker scans based on measured data in future work, but currently adjust the interval to motivate improvements. We could also use more complex attacker models in the future as described in section 12, but use a simple attacker model to maintain simplicity. Instead of computing likelihoods for different threats, we also assume attacks occur and determine which devices are directly compromised by each type of attack. Scores for different attacks can be reported separately to assess which attacks could be most damaging or combined with weights that express the importance of each attack type. New metrics can be easily developed and added for new threats. For example, to assess the danger posed by a previously analyzed advanced persistent threat [6] that exploits known security weaknesses it would be relatively easy to add a new metric.

A third limitation is that we only assess the devices that are directly compromised by a specific attack type. We do not create detailed attack graphs that demonstrate how an attacker can progress beyond this initial foothold through a network using additional vulnerabilities, trust relationships, and other exploitable security properties as described in [16]. This type of attack graph analysis builds on the types of measures used to compute our metrics and is planned as future work.

A fourth limitation of these metrics, that must be addressed by any security metric, is that implementation may require new tools, procedures, and specifications that extend current practice. Our experience to date, is that current SCAP standards and the National Vulnerability Database provide a foundation for creating specifications. These standards often need to be extended to create specifications required to compute our metrics. In addition, we have found that many existing security tools can gather the data required to compute metrics, but the overall process used to collect, verify, correct, and make data available in a database often needs to be improved and more carefully designed to make sure capability deficit metrics are low and operational metrics can be trusted.

**This page intentionally left blank.**

## 20. DISCUSSION AND CONCLUSIONS

Many enterprise networks suffer from weak security postures. Enhancing these postures at an enterprise level, in a cost effective manner, is extremely difficult. We have developed metrics that support a program of continuous risk monitoring to assess and maintain enterprise network security. Our metrics directly measure whether defenders have discovered and removed known exploitable security conditions and use a simple mathematical model to estimate the probability that an attacker discovers and exploits existing security conditions.

This threat-based approach to metric development has many advantages over simpler counts of security conditions. First, each metric focuses on a specific important threat. Second, operational scores that assess risk from a threat are easy to interpret and represent the expected number of devices that are directly compromised by that threat or the total asset value of those devices. Third, it is easy to combine scores across a network and avoid double counting by determining the total number of hosts compromised across all threats. Finally, the methodology developed makes it relatively easy to add metrics for new emerging threats.

We also describe a three-stage metric maturity approach that guides implementation. It includes (1) a checklist stage where tools, processes, databases, and other components are developed, (2) an operational metric stage to guarantee that specifications required to test security properties are available and test measurements can be trusted because they are made rapidly, accurately, and across all relevant network structure, and (3) an operational stage to assess the risk or expected number of devices directly compromised by a specific attack.

Metrics are designed to motivate enterprise-wide security improvements. Low-level risk scores provided for devices, software packages, configuration settings, trust relationships, and other network properties and components make it simple to infer necessary corrective actions while high-level subnet and enterprise scores make it possible to assess overall risk. Incremental credit to operational and capability scores is provided when measurement processes or controls improve incrementally. Finally, the overall difficulty of obtaining a good low metric score increases over time so that initially it is relatively easy to get a good score, but this becomes more difficult as capabilities, controls, and responses mature over time.

The metrics we have developed can provide continuous risk monitoring in any enterprise network and also support continuous risk monitoring suggested by NIST and DHS for government networks[10, 24]. They use data that can be obtained with many existing commercial and open-source security tools such as vulnerability scanners and make use of many existing SCAP standards [34]. The four metrics described in this report are being implemented and tested on actual networks in the Department of Defense and MIT Lincoln Laboratory. We have a detailed simulation for CC-1 that has been used to demonstrate and validate the computations described in this report and an initial reference implementation. This is being extended to model the other metrics. In addition, we are developing additional metrics to cover additional threats that critical controls in [38] are designed to mitigate.

**This page intentionally left blank.**

## 21. BIBLIOGRAPHY

- [1] Terje Aven. *Quantitative Risk Assessment, The Scientific Platform*. Cambridge University Press, 2011.
- [2] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, New York, NY, USA, 1998. ACM. ISBN 0-89791-982-3. URL <http://doi.acm.org/10.1145/277851.277897>.
- [3] Wayne Boyer and Miles Mcqueen. Ideal Based Cyber Security Technical Metrics for Control Systems. In *Proceedings 2nd International Workshop on Critical Information Infrastructures Security*. 2007. URL <http://www.if.uidaho.edu/~amm/faculty/Ideal%20Based%20Cyber%20Security%20Technical%20Metrics%20for%20Control%20Systems.pdf>.
- [4] Donald L. Buckshaw, Gregory S. Parnell, Willard L. Unkenholz, Donald L. Parks, James M. Wallner, and O. Sami Saydjari. Mission Oriented Risk and Design Analysis of Critical Information Systems. *Military Operations Research*, 10, 2005.
- [5] Center for Internet Security. The CIS Security Metrics, December 2010. URL <http://benchmarks.cisecurity.org/en-us/?route=downloads.show.single.metrics.110>.
- [6] Command Five Pty Ltd. Advanced Persistent Threats: A Decade in Review, June 2011. URL [HTTP://www.commandfive.com/papers/C5\\_APT\\_ADecadeInReview.pdf](HTTP://www.commandfive.com/papers/C5_APT_ADecadeInReview.pdf).
- [7] David Cox and Valerie Isham. *Point Processes*. CRC Press, 1980.
- [8] David Dagon, Cliff Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proceedings of the 13th Network and Distributed System Security Symposium NDSS*, 2006.
- [9] Damballa. Damballa Inc. Threat Report - First Half of 2011. Technical report, 2011. URL <http://landing.damballa.com/20110908-1H2011ThreatReport.html>.
- [10] Kelly Dempsey, Arnold Johnson, Alicial C. Jones, Angela Orebaugh, Matthew Scholl, and Kevin Stine. Information Security Continuous Monitoring for Federal Information Systems and Organizations. NIST Special Publication SP 800-137, NIST, 2010. URL <http://csrc.nist.gov/publications/drafts/800-137/draft-SP-800-137-IPD.pdf>.
- [11] Danny Dhillon. Developer-Driven Threat Modeling: Lessons Learned in the Trenches. *IEEE Security and Privacy Magazine*, 9:41–47, 2011. URL <http://www.computer.org/portal/web/csdl/doi/10.1109/MSP.2011.47>.

- [12] Stephan Frei. Fixing the Fundamental Failures of End-Point Security. Presented at ISF World Conference 2011, October 2011. URL [http://secunia.com/?action=fetch&filename=Secunia\\_ISF\\_2011\\_Presentation.pdf](http://secunia.com/?action=fetch&filename=Secunia_ISF_2011_Presentation.pdf).
- [13] Tom Gjelten. For Recent Cyberattacks, Motivations Vary. *National Public Radio News*, June 16, 2011. URL <http://www.npr.org/2011/06/16/137210246/for-recent-cyberattacks-motivations-vary>.
- [14] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover Security Design Flaws Using the STRIDE Approach. *MSDN Magazine*, November 2006. URL <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx>.
- [15] Douglas Hubbard. *The Failure of Risk Management*. John Wiley and Sons, 2009.
- [16] Kyle Ingols, Matthew Chu, Richard Lippmann, and Stephen Boyer. Modeling Modern Network Attacks and Countermeasures Using Attack Graphs. In *ACSAC Proceedings*, 2009. URL [http://www.ll.mit.edu/mission/communications/ist/publications/2009\\_12\\_09\\_Ingols\\_ACSAC\\_FP.pdf](http://www.ll.mit.edu/mission/communications/ist/publications/2009_12_09_Ingols_ACSAC_FP.pdf).
- [17] Andrew Jaquith. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007. ISBN 0321349989.
- [18] Gene Kim, Paul Love, and George Spafford. *Visible OPS Security: Achieving Common Security and IT Operations Objectives in 4 Practical Steps*. IT Process Institute, 2008.
- [19] Ravi Kumar and Andrew Tomkins. A Characterization of Online Browsing Behavior. In *Proceedings of the 19th International Conference on the World Wide Web, WWW 2010*, pages 561–570, 2010.
- [20] David Levin. Lessons Learned in Using Live Red Teams in IA Experiments. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX II)*. IEEE Press, 2003. URL <http://www.bbn.com/resources/pdf/RedTeamExptsPaper-Levin10-02.pdf>.
- [21] Gregor Maier, Anja Feldmann, Vern Paxson, Robin Sommer, and Matthias Vallentin. An Assessment of Overt Malicious Activity Manifest in Residential Networks. In Thorsten Holz and Herbert Bos, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 6739 of *Lecture Notes in Computer Science*, pages 144–163. Springer Berlin / Heidelberg, 2011. ISBN 978-3-642-22423-2. URL [http://dx.doi.org/10.1007/978-3-642-22424-9\\_9](http://dx.doi.org/10.1007/978-3-642-22424-9_9).
- [22] Robert Martin. Managing vulnerabilities in networked systems. *IEEE Computer Society Computer Magazine*, 34(11):32–38, 2001. URL <http://cve.mitre.org/docs/docs-2001/CVEarticleIEEEcomputer.pdf>.

- [23] Peter Mell, Karen Scarfone, and Sasha Romanosky. CVSS: A Complete Guide to the Common Vulnerability Scoring System Version 2.0. Technical report, National Institute of Standards and Technology, 2007. URL <http://www.first.org/cvss/cvss-guide.pdf>.
- [24] Peter Mell, David Waltermire, Harold Booth, Timothy McBride, and Alfred Ouyang. Caesars framework extension: An enterprise continuous monitoring technical reference architecture. NIST Interagency Report 7756, NIST, 2011. URL [http://csrc.nist.gov/publications/drafts/nistir-7756/Draft-nistir-7756\\_feb2011.pdf](http://csrc.nist.gov/publications/drafts/nistir-7756/Draft-nistir-7756_feb2011.pdf).
- [25] Metasploit. Metasploit, March 2012. URL <http://www.metasploit.com>.
- [26] MITRE. Common Platform Enumeration, December 2011. URL <http://cpe.mitre.org/>.
- [27] George Moore. Personal Communication, U.S. Dept of State, 2011.
- [28] NIST. National Vulnerability Database, 2011. URL <http://nvd.nist.gov/>.
- [29] NSA. Manageable Network Plan, April 2011. URL [http://www.nsa.gov/ia/\\_files/vtechrep/ManageableNetworkPlan.pdf](http://www.nsa.gov/ia/_files/vtechrep/ManageableNetworkPlan.pdf).
- [30] Frans Osinga. *Science Strategy and War, The Strategic Theory of John Boyd*. Rutledge, 2006.
- [31] C. Park, H. Shen, J.S. Marron, F. Hernandez-Campos, and D. Veitch. Capturing the Elusive Poissonity in Web Traffic. In *14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 189 – 196, 2006. URL <http://www.unc.edu/~haipeng/publication/poisson-short-gs.pdf>.
- [32] Vern Paxson and Sally Floyd. Wide Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, 1995. URL <http://portal.acm.org/citation.cfm?id=208390>.
- [33] Niels Provos. Spybye web note, <http://www.provos.org/index.php?/categories/1-spybye>, April 2011. URL <http://www.provos.org/index.php?/categories/1-SpyBye>.
- [34] Stephen Quinn, Karen Scarfone, Matthew Barrett, and Chris Johnson. Guide to Adopting and Using the Security Content Automation Protocol (SCAP) Version 1.0. NIST Special Publication 800-117, NIST, 2010. URL <http://csrc.nist.gov/publications/nistpubs/800-117/sp800-117.pdf>.
- [35] Moheeb Abu Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. Trends in circumventing web-malware detection. Google Technical Report rajab-2011a, Google, July, 2011. URL <http://research.google.com/archive/papers/rajab-2011a.pdf>.
- [36] Michael Riley and Sandring Rastello. IMF State-Backed Cyber-Attack Follows Hacks of Lab, G-20. *Bloomberg News*, 13 June, 2011. URL <http://www.bloomberg.com/news/2011-06-11/>.

- [37] Ronald Rudman. iPost: Implementing Continuous Risk Monitoring at the Department of State. Technical report, U.S. Department of State, Information Resource Management, Office of Information Assurance, 2010. URL <http://www.state.gov/documents/organization/156865.pdf>.
- [38] SANS Institute. Twenty Critical Controls for Effective Cyber Defense: Consensus Audit Guidelines, Version 3.0, 15 August, 2011. Technical report, SANS Institute, 2011. URL <http://www.sans.org/critical-security-controls/cag2.pdf>.
- [39] Mell Scarfone, Karen Peter. The Common Configuration Scoring System (CCSS): Metrics for Software Security Configuration Vulnerabilities. NIST Interagency Report 7502, NIST, National Institute of Standards and Technology, 2010.
- [40] Tenable Network Security. Tenable Nessus. Web Site, December 2011. URL <http://www.tenable.com/products/nessus>.
- [41] G.J. Serrao. Network access control (NAC): An open source analysis of architectures and requirements. In *2010 IEEE International Carnahan Conference on Security Technology (ICCST)*, pages 94–102, Oct. 2010. doi: 10.1109/CCST.2010.5678694. URL <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5678694>.
- [42] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association. ISBN 1-931971-00-5. URL <http://portal.acm.org/citation.cfm?id=720288>.
- [43] Sal Stolfo, Steven M. Bellovin, and David Evans. Measuring Security. *IEEE Security and Privacy*, 9:60–65, 2011. ISSN 1540-7993. doi: <http://doi.ieeecomputersociety.org/10.1109/MSP.2011.56>.
- [44] Gary Stonebumer, Alice Goguen, and Alexis Feringa. Risk Management Guide for Information Technology Systems. Special Publication SP 800-30, National Institute of Standards and Technology (NIST), 2002. URL <http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf>.
- [45] TippingPoint. TippingPoint Zero Day Initiative, February 2012. URL <http://www.zerodayinitiative.com/>.
- [46] David Waltermire, Paul Cichonski, and Karen Scarfone. Common Platform Enumeration: Applicability Language Specification, Version 2.3. NIST Interagency Report 7698, NIST, August 2011. URL <http://csrc.nist.gov/publications/nistir/ir7698/NISTIR-7698-CPE-Language.pdf>.

<b>REPORT DOCUMENTATION PAGE</b>			<i>Form Approved</i> <i>OMB No. 0704-0188</i>		
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. <b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b>					
<b>1. REPORT DATE</b> (DD-MM-YYYY) 22 May 2012		<b>2. REPORT TYPE</b> Project Report		<b>3. DATES COVERED</b> (From - To)	
<b>4. TITLE AND SUBTITLE</b>  Continuous Security Metrics for Prevalent Network Threats: Introduction and First Four Metrics			<b>5a. CONTRACT NUMBER</b> FA8721-05-C-0002		
			<b>5b. GRANT NUMBER</b>		
			<b>5c. PROGRAM ELEMENT NUMBER</b>		
<b>6. AUTHOR(S)</b>  R.P. Lippmann, J.F. Riordan, T.K. Yu, and K.K. Watson			<b>5d. PROJECT NUMBER</b> 2020		
			<b>5e. TASK NUMBER</b> 271		
			<b>5f. WORK UNIT NUMBER</b>		
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b>  MIT Lincoln Laboratory 244 Wood Street Lexington, MA 02420-9108			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>  IA-3		
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> Department of Defense Suite 6701 9800 Savage Road Ft. Meade, MD 20755-6701			<b>10. SPONSOR/MONITOR'S ACRONYM(S)</b>		
			<b>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</b> ESC-TR-2010-099		
<b>12. DISTRIBUTION / AVAILABILITY STATEMENT</b>  Approved for public release; distribution is unlimited.					
<b>13. SUPPLEMENTARY NOTES</b>					
<b>14. ABSTRACT</b> The goal of this work is to introduce meaningful security metrics that motivate effective improvements in network security. We present a methodology for directly deriving security metrics from realistic mathematical models of adversarial behaviors and systems and also a maturity model to guide the adoption and use of these metrics. Four security metrics are described that assess the risk from prevalent network threats. These can be computed automatically and continuously on a network to assess the effectiveness of controls. Each new metric directly assesses the effect of controls that mitigate vulnerabilities, continuously estimates the risk from one adversary, and provides direct insight into what changes must be made to improve security. Details of an explicit maturity model are provided for each metric that guide security practitioners through three stages where they (1) develop foundational understanding, tools and procedures, (2) make accurate and timely measurements that cover all relevant network components and specify security conditions to test, and (3) perform continuous risk assessments and network improvements. Metrics are designed to address specific threats, maintain practicality and simplicity, and motivate risk reduction. These initial four metrics and additional ones we are developing should be added incrementally to a network to gradually improve overall security as scores drop to acceptable levels and the risks from associated cyber threats are mitigated.					
<b>15. SUBJECT TERMS</b> Security metrics, risk, threat, critical controls, consensus audit guidelines, controls, maturity model					
<b>16. SECURITY CLASSIFICATION OF:</b>			<b>17. LIMITATION OF ABSTRACT</b> Same as report	<b>18. NUMBER OF PAGES</b> 104	
<b>a. REPORT</b> Unclassified	<b>b. ABSTRACT</b> Unclassified	<b>c. THIS PAGE</b> Unclassified			<b>19a. NAME OF RESPONSIBLE PERSON</b>
			<b>19b. TELEPHONE NUMBER</b> (include area code)		

