Defining a Model for Defense-In-Depth

James Sullivan
University of Calgary
jfsulliv@ucalgary.ca

Michael E. Locasto
University of Calgary
locasto@ucalgary.ca

ABSTRACT

The principle of Defense-in-Depth in secure system design is a widely accepted one; by layering the defensive mechanisms in a security posture, the system is seen as more fault-tolerant and more resistant to adversaries. Despite the wide adoption of this practice, there is a gap in the formal understanding of its abilities to protect a computing system.

Given the wealth of work that has demonstrated the unreliability of security mechanisms under composition, this lack of a formal treatment of Defense-in-Depth as a security principle is concerning. In this paper, we present the Defense Graph model, a formal model for Defense-in-Depth.

Defense Graphs are graph-theoretic tools for the modeling of a system's security properties, focusing attention on data flow and compositional relationships in a system. By constructing a Defense Graph, hidden anti-patterns and weak design choices can be made apparent.

Through a number of case studies, we demonstrate the applicability of our model to system analysis. By describing existing systems in the formal language of Defense Graphs, we show that our model is able to explain measurable phenomena of a system and provide formal foundations for a number of 'best practices' in system design.

1. INTRODUCTION

Defense In Depth is a strategy for securing information systems that is characterized by the use of layered and redundant security measures. The goal of Defense in Depth is to improve fault-tolerance of a system– that any given attack path will be covered by multiple measures in case any of them fail. This is succinctly characterized in the Byzantine Generals problem [14], in which Lamport et. al demonstrate the importance of redundant and distinct mechanisms for fault-tolerance.

Though the strategy of Defense In Depth is recommended by many organizations (e.g. [2–5]), there is a gap in the understanding of the limitations and benefits of Defense In Depth from a formal perspective. An overly simplistic model of security would claim that the ideal security posture is one that covers any attack possible, possibly by using many different specialized security mechanisms. However, this stands against the intuition and advice that simplicity is a desirable property of a security posture, and that the layout of a system is an important factor in its security.

A preliminary study of the effectiveness of layering Antivirus programs (detailed in Section 3) revealed evidence of emergent behavior arising from a complex and layered security posture. We aim to better model the undocumented effects of Defense in Depth with an empirically supported and formal model. To this end, we define the notion of *Defense Graphs.*

The focus of Defense Graphs is on the compositional relationships in a system. It is known that security policies and security softwares do not compose reliably [13,18], even with a deterministic order of application. Worse yet are instances of non-deterministic orders of composition, in which policies are composed in an inherently unpredictable way. Identifying these regions is essential to understand the limitations of the guarantees offered by a layered security posture.

In our model we propose four properties arising in a layered security posture. *Independence* refers to the degree to which individual security measures in a system will not interact with one another. *Coverage* refers to the extent of the instrumentation provided by a given security measure, or the language that it instruments. *Redundancy* is the degree to which security measures overlap in their coverage. *Cost* is a measurement of the cost of a given system, including both financial and computational resource costs.

These properties were selected to be sufficiently expressive of basic properties of a system, while still remaining grounded in empirical measurements. They are also extracted from data that is commonly logged, so that an organization could begin to measure these properties with marginal overhead or specialized hardware and software.

A model's strengths arise from its predictive and explanatory powers of natural phenomena. A model that can both predict the outcome of natural phenomena given initial conditions, and can explain these outcomes in reverse, is a useful one for describing the properties of its subject. In this paper we establish the explanatory power of the Defense Graph model through a number of case studies, outlined in Section 6.

2. RELATED WORK

2.1 Language and Automata-Theoretic Security

Our model defines Security Mechanisms as an automaton that allows a subset of its input data to pass through as output, reducing the data which does not comply with its security policy. This approach is captured in past literature such as [21], in which Schneider et. al propose a formal model for security policies and the mechanisms that enforce them. In particular, they define a subset of Security Policies which are *Execution Monitor* enforceable, corresponding to a set of mechanisms that operate only by monitoring a target's execution. This is similar to our notion of *Reducers* introduced in Section 4.1.

Work in [22] further expands the language theoretic security model, formally defining the properties of type-safe languages and code analysis techniques as forms of policy enforcement. The authors indicate in their model that these different classes of mechanism can be composed to be greater than the sum of their parts; from the perspective of our model, these compositions succeed precisely for this diversity of mechanism. We explore in Section 6 the consequences of composing similar security mechanisms, and of composing them in unreliable order.

2.2 Graph-Theoretic Security Models

A number of security models have utilized graphs and trees to analyze a system's security. Attack graphs [24] (also Scenario Graphs [12] and Attack Trees) are tools to model a set of attack vectors against a system, where an attack on a system is represented by a finite chain of atomic attacks (a sequence of state transitions) resulting in the system entering an unsafe state.

Attack Graphs are useful and widely adopted both for vulnerability analysis of a system, and to provide practical security recommendations for a system. For example, Sommestad et. al extend the Attack Graph with nodes representing security measures in [25], and establish a probabilistic heuristic to assess the risk of a system given its security layout. In [15], Lippmann et. al applied attack graphs to layered SCADA defense networks, using the graphs to provide security recommendations.

2.3 Security Policy Composition

It is known that in general, security policies are not secure under composition. Early work from Abadi [6] defined the composition principle, conditions under which policy enforcing modules are reliably composable. In particular, they establish that protocols are composable only when the result of one protocol implies the environmental conditions the other requires. This principle is applied by Heckman in [10], to prove the correctness of a two-server compositional model within a secure microkernel.

Heintze and Tygar [11] established a formal model of security policies and demonstrated their unreliability under composition, and defined conditions of a model to guarantee reliable policy composition. These conditions are twofold- firstly, that the policies are message independent (the messages sent by one policy are not valid messages for the other and will not be interpreted), and are directed secret-secure (if a policy under a model is secure from some initial state, then it is secure).

Another formal treatment of protocol composition was given in [13], in which the Chosen-Protocol Attack is described. Given two security protocols which share some information (in this case, a cryptographic key), it is possible for an adversary to convince some parties that they are interacting with one protocol when in fact they are interacting with the other. This work shows the danger in the sharing of information between protocols, and more generally in composing their operation.

Universal Composability has been explored by [7,17]. While this is similar in name and concept to our work, it is distinct in that our model of composition is over mechanisms represented as automaton, rather than over protocols (in particular, cryptographic protocols). Our work does not present itself as an extension to the concept of universal composability or conditions for successful composability applied to software.

A more similar treatment to composability is given in [16], in which McCullough uses the notions of non-interference and non-deterministic compositions to define conditions in which policy enforcing softwares may compose, and in particular showed that there can be composition only when there is a reliable ordering of mechanism and when there is no feedback between two mechanisms.

Work by Datta et al. in [9] further explores conditions of reliable composition; the general idea of composition is broken into the two categories of sequential and parallel composition, and individual conditions that must be satisfied are presented. In particular, sequential composition requires that the post-conditions of one mechanism satisfy the preconditions of the other, and all types of composition require non-interference or secret sharing between mechanisms.

Technical discussion of Linux Security Modules (LSMs) in [18] shows that this problem is not restricted to formal domains, and is a concern in the practical domain as well. In particular, composition of LSMs is a technical challenge because of the absolute kernel access that these LSMs use and require (for instance, they often instrument system calls or kernel data structures, and this in itself creates a point of interfering functionality between LSMs). This creates significant interference between LSMs.

A recent Linux patch [23] implements LSM stacking in the Linux kernel by having each of the possible LSM security hooks contain a list of function pointers rather than a single pointer. Each of these function pointers (installed by various LSMs such as SELinux or TOMOYO) must agree on a system call's legality for it to be permitted.

2.4 Fault Tolerance of Security Systems

A classic piece of work by Lamport et. al [14] defined the Byzantine Generals problem, in which it was shown that a composition of decision-making subsystems can tolerate only m incorrect decisions for every $3m+1$ correct decisions. This is to say that a system can tolerate failure of its mechanisms only up to one third of the total number it employs. This informs a core tenet of Defense in Depth- by increasing the redundancy of mechanism, we increase the system's ability to recover from a byzantine failure.

While this is a desirable property of a system, we posit that its benefits diminish with overuse. We will explore in Section 6 cases in which the composition of security mechanisms in a system does not increase its fault tolerance, but instead introduces complexity which in turn creates an environment for emergent behavior and unpredictability to occur.

3. MOTIVATION

To motivate the need for a careful consideration of the compositional relationships between security mechanisms, we performed a series of tests on commodity Antiviruses to measure the effects of composition on a system's effectiveness. A set of popular Antiviruses were combined on a host system, and their effectiveness against a standard set of test files was measured. In particular, we measured two simple indications of effectiveness- the ability of the security configuration to detect a threat, and its ability to protect from a threat.

Detection here is defined as a notification being made to the user or to the system that a threat was identified by the Antivirus. Protection is an extension of this behavior, but also requires that the Antivirus prevented access to the malicious data by removing it or quarantining it.

Two further notions are defined to measure the compositional effects of these Antiviruses. We define the notion of an induced failure to be the case where Antivirus A passed an EICAR test by itself, but failed the test when paired with Antivirus B. An absolute induced failure is the case where both A and B failed, which would result in a system breach. Induced detection failure is similarly defined.

A Windows 7 Professional instance hosted in a virtual machine was used as the testing environment. A series of five test scenarios, provided by the Anti-Malware Testing Standards Organization ([1]) were carried out in this environment. We measured the effectiveness of each Antivirus against each test file in isolation, and of each Antivirus pairs. The tests were performed three times each for each configuration. The five scenarios were as follows:

1. Manual Download of a malicious file (EICAR File)
2. Drive-By Download of a malicious file (EICAR File)
3. Manual Download of a Potentially Unwanted Application
4. Accessing a Phishing Page
5. Cloud Protection - Test the ability of the Antivirus to consult a cloud signature database.

Table 1 contains the results of our analysis, as the proportion of tests in which each event occurred to the total number of tests.

We observed that in 24.7% of all pairings, an induced failure to protect against the malicious file was observed. In 10.8% of cases this failure was absolute; in the other cases, the threat was still prevented by the paired mechanism. Detection was less significantly affected by the pairing. 5.4% of cases exhibited an induced detection failure.

	Trial 1	Trial 2	Trial 3	Average
IDF	0.056	0.056	0.048	0.054
IPF	0.234	0.266	0.242	0.247
IAPF	0.113	0.113	0.097	0.108

Figure 1: Results from the Antivirus case study. "IDF, IPF" refer to Induced Detection/Protection Failure. "IAPF" refers to Induced Absolute Protection Failure.

The induced failures that were observed in this case study supports a claim made by our model: that non-deterministic compositions (defined in Section 4 can diminish security guarantees. By introducing a second mechanism into the system, security was not enhanced, because the two mechanisms destructively interfered with each other's input.

In contrast, a multi-scanning system such as [4] gives each antivirus a copy of the data to scan and allows them to work independently of one another. In such a system, the interference between mechanisms is eliminated and their operation becomes more consistent. This is strong evidence that the layout of a system has an effect on its security, which is the key insight behind our model.

A further effect that was noted in these experiments was a significant performance drop. When a second Antivirus was installed on the system, there was a noticeable performance degradation in almost all tasks, such as desktop navigation or opening an application. We did not, however, measure these effects due to the constraints of our test architecture. Our scenarios took place in virtual machines, which makes a fair assessment of performance degradation challenging. Future experiments will require a more isolated environment to better assess the performance degradation.

This small sample size, tested against a limited range of test files (which were industry standard test files that most signature-based Antiviruses will detect) does not constitute a realistic scenario or a complete analysis of all commodity Antiviruses. However, even within this small case study, mechanism composition was shown to affect the security of the system, motivating a formal approach to considering how the structure of a system affects its security.

4. MODELLING DEFENSE-IN-DEPTH

Our case study in the composition of Antivirus software has demonstrated that security mechanisms do not reliably compose under some circumstances, and can fail to correctly enforce their policies. This emergent behavior prevents policy-centric models of security composition from fully describing their actual implementation. Simple models that ignore these properties can lend a false sense of security to organizations by teaching that applying more security software will always provide better security.

We model these complex relationships by introducing the notion of **Defense Graphs**. A Defense Graph is a tool that enables empirically founded descriptions and analysis of the security properties of a system, allowing for predictions to be made about problematic patterns in a system's layout. Defense Graphs not only account for the language of individual security mechanisms, but also for the data flow and the layout of the system.

4.1 Definitions

Defense Graphs provide a formal language for the analysis of security postures. In particular, they provide insight into the relationships shared by security mechanisms, the direction of data flow, and the emergent properties of composed security mechanisms. We first formally define the Defense Graph and its components.

Definition 1. A **Defense Graph** is a directed, acyclic graph $D = (V, E)$ where

- V is a set of vertices, with $|V| \geq 2$.

- $\alpha \in V$ is the **Entry Point** to the system, with no in-bound edges and a unique out-bound edge. α is connected to every vertex in D.
- $\beta \in V$ is the **Target** in the system, with no out-bound edges and a unique in-bound edge.
- $M \subseteq V$ is a set of **Policy Enforcers** $\{m_1 \dots m_n\}$ with $n \geq 0$.
- $P \subseteq V$ is a set of **Policy Selectors** $\{p_1 \dots p_k\}$ with $k \geq 0$.
- $M \cap P = \emptyset$.
- $E \subseteq V \times V$ is a set of directed edges, with $|E| > 0$.

A Defense Graph D satisfies the following properties:

A system's security policies are enforced in practice by some security measures, such as a firewall or an access control list, which instrument some data stream in the system and remove input that is deemed to be malicious. These security measures become the Policy Enforcers of the Defense Graph.

Definition 2. A **Policy Enforcer** $m = (S, \tau, s_0)$ is a finite automaton with a set S of states, a set of transition relations $\tau \subseteq S \times S$, and an initial state s_0.

A policy enforcer has an input language I_m, corresponding to all valid data for the mechanism, and an output language O_m corresponding to data that the mechanism accepts and allows to continue through the system.

Security Mechanisms can be placed into a number of categories based on their semantics. We outline a number of these common categories, but note that the description may not capture all unique properties of security mechanisms- a total taxonomy of the effects and side effects of security software is not feasible.

The first policy enforcement classification is the filter. In this case, the output language O_m is a subset of I_m, that is to say that all input strings $i \in I_m$ will either be *passed through* m or they will be *rejected by* m. This corresponds to many typical security measures; for example, a Firewall will have an input language corresponding to the set of valid network packets, and an output language of the packets that comply with its policies (a subset of the input).

Note the correspondence of this type of mechanism to the Execution Monitoring mechanisms defined in [21], which provides a rich (but by necessity, incomplete) specification of many common security mechanisms. These mechanisms form the basis for our model, from their sound and describable theoretic properties.

A mechanism may transform data to meet a specification, or to append metadata, or to encode the data. This type of mechanism produces an output language O_m that may be disjoint from the input language I_m. Such mechanisms could be said to *Append* to or *Modify* a language; while these mechanisms are not a focus of our model, we note their existence.

A second type of vertex in these graphs is the Policy Selector, which directs, collects, or multiplexes data streams based on the type of the input. The clearest example of such a device is a switch or a router in a network, which directs packets to the correct machine on the network. A more subtle instance of a policy selector is a dispatch table such as a system call table; if a security module in a kernel instruments some particular system call to check for validity of input, then the dispatch table acts as a policy selector.

Let $\mathcal{L}$ be the set of all possible input and output languages.

Definition 3. A **Policy Selector** $c \subseteq \mathcal{L} \times \mathcal{L}$ is a function mapping an input language to an arbitrarily transformed output language.

$|c| > 0$, that is to say that c contains at least one (I_0, O_0).

4.2 Constructing Defense Graphs

The first step in the construction of a Defense Graph is to identify the target of the system, and the adversary's entry point. If multiple such points exist in the system, then it is necessary to construct multiple Defense Graphs- however, they may share many similar subsections. In an example of a network, the entry point to the system is the internet-facing gateway, from which an adversary would originate their attacks. The adversary would likely target particular hosts on the network, such as an internal file server, which would become the target.

Following this, the analyst must enumerate the security measures of a system, their point of instrumentation, and the type of data they instrument. The security measures become the vertices in the Defense Graph, and edges between vertices correspond to a direct data channel between two mechanisms. The more detailed a description of a mechanism's language is, the more accurate the Defense Graph will be. However, it is sufficient to give broad descriptions of input, as long as it is known when two mechanisms do or do not overlap in the types of data that they analyze.

For example, a network may employ security mechanisms such as a perimeter firewall, an intrusion detection system, or a number of host-based access control mechanisms. Identifying the type of data instrumented by the security mechanism can be simple (the firewall, for instance, instruments valid network packets), or somewhat more involved (an Access Control List instruments a request for access to an asset by a given individual– formally, pairs of individuals and asset requests).

There are some cases where two mechanisms do not operate on data in a consistent order. These mechanisms are said to compose non-deterministically (described in detail in Section 4.3), and are treated as a single vertex with a well defined entry and exit point but with some internal non-deterministic state. Typically these constructs correspond to concurrent access to shared data, which is known to be problematic even from a non-security standpoint. Avoiding these constructs in a system is advisable, particularly when the mechanisms operate on similar types of data.

Finally, there are instances where some form of policy selection occurs, which simply corresponds to a point in the system where data can take one or more different paths based on its properties. An example of this is a switch in a network which directs packets to a particular host on the network, or a dispatch table which calls a particular function in its table based on the input given to the dispatcher. These can also multiplex and demultiplex data- for example, a hub broadcasts input to many hosts, and an emerging technique known as multi-scanning (e.g. Metascan [4]) demultiplexes the scan results from a number of antivirus suites into a single result.

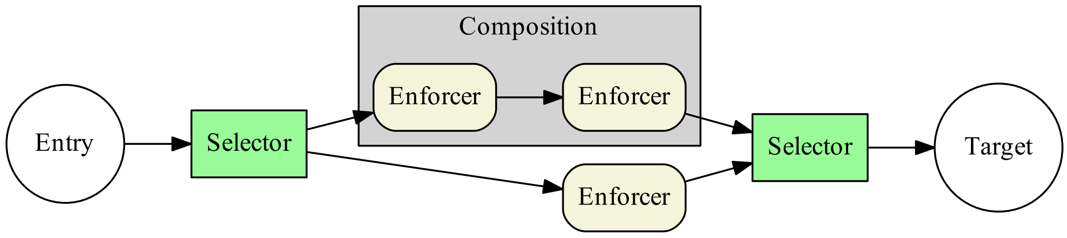


Figure 2: A simple Defense Graph. Note that the top two Policy Enforcers are said to be composed.

It is vital that the analyst identifies the system's features correctly and completely. An unspecified path through a system can be used by an adversary to bypass sections of the security architecture, which can lead to an unexpected compromise. Similarly, an ill-specified language of a security mechanism can be problematic- the mechanism's actual and predicted properties will not be equal, and the Defense Graph's properties discussed in Section 5 will be incorrect.

4.3 Mechanism Composition

Adjacent security mechanisms in a system form a composition of security policies. A great deal of literature on secure policy composition exists, but there has been little discussion of the order of policy application, which Defense Graphs enables. In particular, two mechanisms can be composed in one of two ways, with different resulting properties.

Given two mechanisms m_i, m_j, a composition of these mechanisms is exhibited when the output of one m_i becomes the input to the other m_j (or vice versa). This composition is either *deterministic*, where a consistent ordering is exhibited, or *non-deterministic*, where the order of execution is not consistent (such as in a concurrent system where two mechanisms execute in distinct and non-locking threads, but operate on the same data).

Definition 4. Given two security mechanisms m_i, m_j with $O_{m_i} \subseteq I_{m_j}$, a **Deterministic Composition** m_{ij} is a security mechanism constructed by concatenating m_i and m_j in sequence.

m_{ij} satisfies the following properties:

1. $I_{m_{ij}} = I_{m_i}$ (The set of valid inputs is that of the first mechanism), and
2. $O_{m_{ij}} \subseteq O_{m_j} \cap O_{m_i}$ (The output of the composed mechanism is all inputs that both m_i and m_j accept).

Note that $m_{ij} \neq m_{ji}$.

An example of a deterministic composition is a system with a network-based perimeter firewall and an intrusion detection system. For any given packet i originating from outside of the network, i must traverse the firewall before the Intrusion Detection System can execute on i.

Note that the second (*dependent*) mechanism must have an input language that is a subset of the output of the first (*preceding*) mechanism. If this is not satisfied, then the two mechanisms cannot be readily composed; instead there must be some arbitrary computation that takes place between the two mechanisms to facilitate data flow from one to the other. Data that does not conform to the language of the latter security mechanism may be treated as valid when it is not, or vice versa. Such an instance could occur if a secure channel were opened by an adversary, allowing them to send encrypted data to the target that bypasses a signature based malware detector. Despite the malicious nature of the data, its new form could prevent its identification by the latter security measure.

Definition 5. Given two security mechanisms m_i, m_j, a **Non-deterministic Composition** n_{ij} is a security mechanism constructed by concatenating m_i and m_j in a non-deterministic order.

n_{ij} satisfies the following:

1. $I_{n_{ij}} = I_{m_i} \cup I_{m_j}$ (The set of valid inputs is the union of valid inputs for both mechanisms)
2. $O_{n_{ij}} \supseteq O_{m_i} \cup O_{m_j}$ (The language of acceptance for the non-deterministic composition is minimally the union of its mechanisms' languages of acceptance, and possibly contains other data).

Note that $n_{ij} = n_{ji}$.

Non-deterministic compositions of security mechanisms is clearly exhibited in the case study of Antivirus cooperation (Section 3), in which we observed that the order in which mechanisms operated on data was not deterministic. This inconsistent ordering created an environment for emergent behavior, which was reflected in the increased rate of failure for composed antiviruses. This non-deterministic configuration is an undesirable one, as fewer guarantees can be made about the behavior of the composition and its effectiveness cannot be reliably predicted due to emergent behavior.

If data that originates from the entry point to the system will always traverse one mechanism first, and the second immediately after, then they are deterministically composed. Should this ordering or path not be consistent, then they exhibit non-deterministic composition. In all other cases, there is no composition exhibited between the two mechanisms. These conditions of nondeterminism are similar to those given in [16]. Methods such as system instrumentation, execution tracing, and network scanning are available to extract this information, as well as basic system analysis.

5. PROPERTIES OF DEFENSE GRAPHS

Defense Graphs describe the security properties of a system in terms of four distinct properties- the *Coverage* of the input to the system that is provided by its set of mechanisms, the *Redundancy* of this coverage, the *Independence* of the mechanism's operation and the *Cost* of the arrangement.

We first define Coverage and Redundancy, two tightly knit properties of a system, in terms of the languages of its security mechanisms. In rough terms, the coverage of a security mechanism is the set of inputs which it is capable of processing and enforcing its policies on. More precisely, the coverage of a mechanism m_i is its input language.

Note that coverage is not related to whether the mechanism m_i accepts or rejects the input; coverage is a measure of instrumentation and not of successful policy enforcement. To measure successful policy enforcement presupposes an effective method of distinguishing malicious and non-malicious input. Since an exploit is a program provided as input that induces unexpected computation (see [20]), and since it is undecidable to determine if a program is malicious (see [8]), this too is an undecidable problem.

Definition 6. The **Coverage** of a mechanism m is $C(m) = I_m$, the set of input strings that the mechanism interprets.

The coverage of the system is given by:

$$C(D) = \bigcup_{i=0}^{|M|} C(m_i) \tag{1}$$

Or the set of all inputs for which some mechanism exists in the system that interprets the input.

For a system to exhibit Defense in Depth, it clearly must have some overlap in the coverage offered by its mechanisms. We model this overlap in terms of the intersection of coverage- Redundancy is precisely this notion.

Definition 7. The **Redundancy** of two mechanisms m_1, m_2 is given by:

$$R(m_1, m_2) = \frac{|C(m_1) \cap C(m_2)|}{|C(m_1) \cup C(m_2)|} \tag{2}$$

Or the proportion of the overlapping coverage provided by the mechanisms to their total coverage.

The total redundancy of the system is given by

$$R(D) = \frac{2 \sum_{i=0}^{|M|} \sum_{j \neq i}^{|M|} R(m_i, m_j)}{|M|^2 - |M|} \tag{3}$$

Or the proportion of the redundant (distinct) mechanism pairs to the number of possible pairs.

A system with minimal redundancy incorporates many distinct security mechanisms, while a system with maximal redundancy would be composed of a number of security measures with similar coverage.

The notions of coverage and redundancy speak to the language of mechanisms, but not to their arrangement. The arrangement of two mechanisms depends on the composition that they exhibit.

Given two mechanisms m_1, m_2, there are four ways in which the mechanisms can relate when placed into a Defense Graph.

1. An edge $e = (m_1, m_2)$ (or the reverse) exists. In this case, the mechanisms are *in sequence.*

2. m_1, m_2 compose to n_{12}. In this case, the mechanisms exhibit some non-deterministic structure.

3. m_1, m_2 have input edges e_1, e_2 with $e_1 = (c, m_1)$ and $e_2 = (c, m_2)$ for some $c \in C$. In this case, the mechanisms are *in parallel.*

4. If none of the above are satisfied, the mechanisms are *not adjacent.*

The first two cases are of interest because they exhibit a *Dependence* between mechanisms, in which the policy enforcement provided by one mechanism modifies the data stream that is then passed to the second mechanism.

Definition 8. Given two mechanisms m_1, m_2, we say that m_1 **depends** on m_2 if and only if any walk from $\alpha \dots m_1$ must also contain m_2. Independence is the converse notion.

$$I(m_1, m_2) = \begin{cases} 0: & w = \alpha \dots m_1 \implies m_2 \in w \\ 1: & \text{otherwise} \end{cases} \tag{4}$$

The **Independence** of the system $I(D)$ is given by

$$I(D) = \frac{2 \sum_{i=0}^{|M|} \sum_{j=0, j \neq i}^{|M|} I(m_i, m_j)}{|M|^2 - |M|} \tag{5}$$

i.e., the proportion of independent (distinct) mechanism pairs to the possible mechanism pairs.

A system with minimal independence chains together many security measures, while a system with maximal independence would distribute the flow of data across many security measures that do not pass data to each other.

The fourth metric accounts for the cost of a given security posture. Each mechanism is introduced at a cost to the system, be it financial or in terms of resource usage. This metric can be measured in a variety of ways. As an example we measure cost in terms of the time taken for a computational task to complete by a security-laden system compared to the base system with no security measures. Any metric that results in a proportional value of cost (where higher is better) can be used in place.

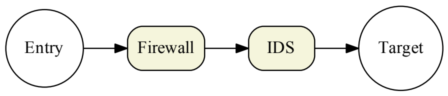
Definition 9. Let D_0 be the Defense Graph D modified to be such that $M_0 = \emptyset$; i.e., the unprotected system. Let J be a set of computational tasks, and let $T : \mathcal{D} \times J \to (0, \dots)$ be the time function given by $T(D, j) = t$ where t is a positive number of milliseconds that the task j took to complete on the system represented by Defense Graph D.

The **Cost** function $P : \mathcal{D} \to [0, \dots)$ is given by

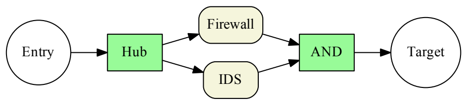
$$P(D) = \sum_{i=0}^{|J|} \frac{T(D, j_i)}{T(D_0, j_i)} \tag{6}$$

If any given task $j \in J$ is such that D cannot complete j (i.e., the system represented by D crashes during j), then $P(D) = 0$.

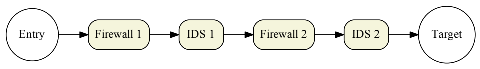
The above metrics are combined in a 4-dimensional vector describing the system's measured security properties. This enables a simple comparison between security postures, detailed in Section 5.2.



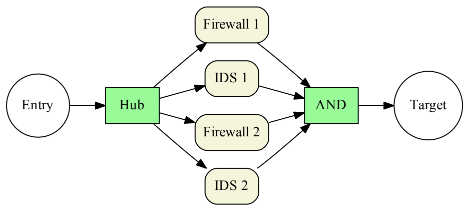
(a) D_1 - High Coverage, Low Redundancy, Low Independence



(b) D_2 - High Coverage, Low Redundancy, High Independence



(c) D_3 - High Coverage, High Redundancy, Low Independence



(d) D_4 - High Coverage, High Redundancy, High Independence

Figure 3: Four principal patterns in Defense Graphs.

5.1 Building Blocks of Defense Graphs

From the above four notions of Coverage, Redundancy, Independence, and Cost, we construct a number of idealized Defense Graphs that correspond to a high or low value of the first three properties. Note that the last property (Cost) is a dependent property- that is to say that the performance of the system arises from its configuration.

There are four principal patterns which we will examine in this section. These patterns are of interest to note because they are typical constructs in any Defense Graph and can be composed to form arbitrarily complex graphs. See figure 3 for each of these constructs.

It seems apparent that the most desirable type of construct is D_4, maximizing Coverage, Redundancy, and Independence. While this ideal is of note for its strong properties, it is not a typical structure in system design. Most systems tend to process data linearly and in discrete steps, rather than the massively parallel and independent manner shown in this construct.

5.2 Analyzing Defense Graphs

A common intuition about a system is that it is only as strong as its weakest link. An attacker does not need to enumerate all vulnerabilities of a system, instead needing only to define a chain of events that lead to some attack goals being met. Attack Graphs [24] are a useful tool for the attacker and the defender alike, helping to visualize and enumerate these chains of attack targets. Defense Graphs also provide a similar notion that is useful to both parties.

Definition 10. An **Execution Chain** C is a walk through the Defense Graph D, starting at α and ending at β, traversing zero or more mechanisms $m_1 \dots m_n$.

Execution Chains simplify the analysis of a Defense Graph, and thus the analysis of a complex system. We argue that an analysis of Execution Chains in a Defense Graph suffices to provide a lower boundary for the overall effectiveness of the system by considering the relationships possible between two adjacent mechanisms.

The notions of *Redundancy* and *Independence* are vital in defining the relationship between two adjacent security mechanisms. Informally, redundancy is the degree to which each mechanism's coverage of the vulnerabilities in the system overlap (concerned with the language of the mechanisms), and independence is the absence of a direct data flow from the output of one mechanism to the input of the other (concerned with the layout of the mechanisms).

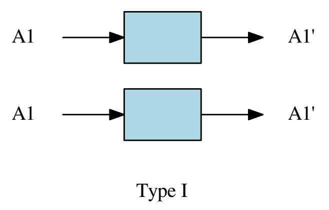
Given any two mechanisms, there are four key classes of arrangement that can be exhibited, visualized in Figure 4. If two mechanisms m_1, m_2 are arranged as in patterns III or IV, then they exhibit a *sequential* arrangement (and thus, exhibit some type of composition). If they are instead arranged in patterns I or II, then they exhibit a *parallel* arrangement.

Mechanism	Input Language	Output Language
m_{I_1}	I_I	O_I
m_{I_2}	I_I	O_I
m_{II_1}	I_{II_1}	O_{II_1}
m_{II_2}	I_{II_2}	O_{II_2}
m_{III_1}	I_{III}	O_{III}
m_{III_2}	I_{III}	O_{III}
m_{IV_1}	I_{IV_1}	O_{IV_1}
m_{IV_2}	I_{IV_2}	O_{IV_2}

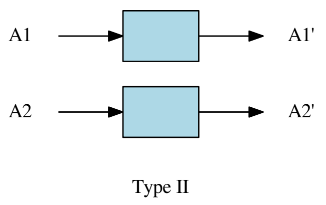
Figure 5: The eight mechanisms of the four example Defense Graphs.

Execution chains are defined by a series of Type III or IV arrangements. No execution chain C can contain any Type I or II arrangements; at this point the execution chain C branches to two distinct $C_1 \neq C_2$.

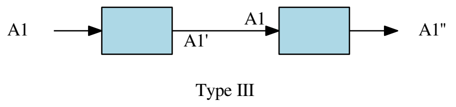
To show that Type III and IV arrangements are in general the worst case (and thus provide a lower bound for



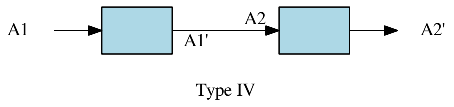
(a) Parallel, Redundant



(b) Parallel, Non-redundant



(c) Sequential, Redundant



(d) Sequential, Non-redundant

Figure 4: Four classes of mechanism arrangement.

the overall effectiveness of the security posture), we consider four simple Defense Graphs $D_I \ldots D_{IV}$, each of which contains two mechanisms m_{I-IV_1}, m_{I-IV_2} arranged in patterns $I - IV$. Refer to Table 5 for the specifications of the eight mechanisms. Suppose that the size of coverage is equal for all mechanisms (though they are possibly disjoint). Suppose that the cost of each system is fixed.

The *Coverage, Redundancy, and Independence* of each Defense Graph $D_I \ldots D_{IV}$ is given in Figure 6, with an example calculation given for D_I below. (We use, without proof, that $C(m_1) = C(m_2) = I_I \setminus O_I$ by applying definition 6).

$$C(D_I) = C(m_{I_1}) \cup C(m_{I_2}) = I_I \setminus O_I$$

$$R(D_I) = \frac{|C(m_{I_1}) \cap C(m_{I_2})|}{|C(m_{I_1}) \cup C(m_{I_2})|} = \frac{|I_I \setminus O_I|}{|I_I \setminus O_I|} = 1$$

$$I(D_I) = \frac{2 * I(m_{I_1}, m_{I_2})}{|M|^2 - |M|} = \frac{2 * 1}{4 - 2} = \frac{2}{2} = 1$$

Observe that in Figure 6, there is a marked advantage held by the two parallel Defense Graphs (D_I and D_{III}) when compared with the sequential arrangements. In particular, the parallel systems exhibit independence between their mechanisms, while the sequential systems do not. This observation supports the hypothesis that a correctly implemented parallel structure may be more fault-tolerant than a sequential one, at the cost of reduced total coverage.

D	Coverage	Redundancy	Independence
D_I	c_I	1	1
D_{II}	c_{II}	0	1
D_{III}	c_{III}	1	0
D_{IV}	c_{IV}	0	0

Figure 6: Calculated values for each Defense Graph. $0 \leq |c_I, c_{III}| < |c_{II}, c_{IV}|$, since D_I, D_{III} use the same mechanism twice, but D_{II}, D_{IV} use two distinct mechanisms.

Note that we emphasize the importance of non-interference in a parallel implementation- if there is unexpected and undefined interaction between two mechanisms, then their Defense Graph cannot be in parallel by definition, and so such a Graph would not correctly describe the system. This reiterates the importance of a correct and complete analysis of the data flow and interactions in a system.

From this, we see that the optimal local strategy in this case is to select a *Parallel* arrangement of mechanisms; whether the mechanisms should be highly redundant (Type I) or have broad coverage (Type II) depends on the desired properties of the system.

This local optimality implies that a lower bound on the Coverage, Redundancy and Independence of a system can be found by analyzing its sequential Execution Chains, which are constructed from locally non-optimal arrangements. In other words, our model matches the intuition that the system is only as strong as its weakest link.

6. APPLICATIONS OF DEFENSE GRAPHS

In their seminal paper [19], authors Saltzer and Schroeder present a set of security principles that directly inform modern design strategies. These eight principles, which to this day are cited and referenced, arise from observations of the challenges in secure system design and from similar challenges in other domains, such as in military operational security.

A design focus of our Defense Graph model is to explain the fundamental benefits of these common design principles in a formal manner. We demonstrate that our model is suited to describe these principles and to argue for their effectiveness purely from its own language. We will in particular discuss three principles provided in [19].

Economy of Mechanism - The design of a system should be kept as small and simple as possible. Unwarranted complexity does not increase the effectiveness of a security system, and in fact may undermine it by increasing the size of the trusted code base, possibly introducing new vulnerabilities.

The mechanisms of a Defense Graph should impose a minimal cost on a system to maximize its effectiveness. Complex structure (such as non-deterministic compositions) should be avoided in favor of simple and reliable constructs. These complex arrangements may reduce the effectiveness of the system, since they are prone to unpredictability and emergent properties.

Similarly, the data flow through the system should be simple, to ensure a fuller understanding of the attack vectors against a system. Complex data paths cannot easily be described as Defense Graphs and thus do not lend themselves to modeling.

Complete Mediation - Every access to every object must be checked for authority. Enforce a system-wide view of access control.

There is a direct causal relationship between the increasing coverage of a system and its overall effectiveness. Ensure that when feasible, all vectors are protected against by one or more mechanisms. Moreover, care must be taken to allocate security measures to the weakest points in the system. An unprotected data path should be the exception, and not the rule.

Least Common Mechanism - Minimize the amount of mechanism common to many users or depended on by all users. Any shared mechanism is an information path that may be compromised and abused.

Ensure that security mechanisms operate and fail independently when possible. Dependence relationships in a system reduce its effectiveness and can create situations where a failure in a subsystem becomes a total failure.

By redefining these fundamental security concepts in the language of our model, we show that reasonable and informed decisions about the benefits and costs of a security posture can be made by applying our model. These principles which are commonly accepted and applied can be formalized by our model, and the underlying reasoning behind their adoption can be explained.

7. DISCUSSION

The model that we have presented here is useful for concisely describing the compositional relationships present in a computer system. However, our model makes a number of simplifying assumptions, as with most models, to make it more practically applicable and to reduce the cost of adopting the model. Because of these assumptions, there are several limitations of our model that must be acknowledged and discussed. We will address some of these limitations in this section, and discuss the practicality and the contribution of the Defense Graph model.

What practical contributions do the Defense Graph model offer?

The primary contribution of our model is that it expands the focus of secure system design beyond simply selecting some set of security mechanisms. It is clear that an enumerated list of security mechanisms is a low-fidelity information source; knowing that a firewall exists in a network is very different than knowing where in the network the firewall is, and what input streams it instruments. There are practical security consequences to the location in a system that a mechanism performs its work, which are not captured by the list-of-mechanisms approach.

If a network IDS is monitoring a wide input stream that connects the majority of the network traffic to a wider network (e.g. the Internet), then it will be able to examine most of the traffic. This has benefits for security (since the NIDS have an opportunity to view most attack attempts), with a possible increase of cost since the NIDS may require more robust hardware to handle the increased payload. That the behavior and effectiveness of individual mechanisms will in part depend on their layout is intuitive and was acknowledged by Uribe et. al in [26], who presented a constraint-based approach to configuring NIDS and firewalls in a network to improve security guarantees.

The need for modeling techniques for the effect of the layout of security mechanisms is clear. Defense Graphs offer an initial framework for the analysis of this aspect of security, which has traditionally been only considered in informal terms. Our model formalizes several intuitive notions of secure systems (as was seen in Section 6), which is valuable for the research community as it spearheads more principled approaches to security.

The four properties of Defense Graphs that we introduced in Section 5 show one way that a formal model can be used to make comparative statements about the quality of security layouts. These properties are simple but capture some important notions in Defense-in-Depth, while avoiding the pitfalls of having a ‘security metric’. In particular these properties can be used for simple cost-benefit analysis, and answering practical questions such as *“Which security mechanism can I remove from my system with the least security impact?”* and *“Where is the ideal location in a system to install a new security mechanism to improve my system’s redundancy and fault-tolerance?”*

Do the four properties of Defense Graphs adequately capture whether a system is secure or not?

As with any metric in security research, there are two potential pitfalls to be aware of when defining a metric. If metrics are too broad (for example, the ‘security metric’, which has no universally agreed-on definition), it is difficult to make strong claims using the metric. It may not be clear that an overly broad metric models what it intends to, and there may be too many factors in the measurement for the metric to be properly isolated from other variables.

On the other hand, an insufficiently expressive metric provides little practical use for researchers. Metrics are used to extract useful information from models, simulations and experimentation, so any metrics that are defined should aim to capture some essential kernel of information about its underlying system.

The metrics that we have provided attempt to find a balance between these two pitfalls. We selected a set of properties that are well-defined and can be reasonably extracted from a Defense Graph, while still conveying some useful information about the system. These properties may not be the only useful metrics that can be established for the Defense Graph model, and future work may establish more robust and expressive metrics for Defense Graphs or validate the existing properties as accurate models of system behavior.

What are the issues and limitations with this approach?

As the Defense Graph model is still relatively young, there are some details that have not been considered yet. One such detail is the different behaviors of other types of security mechanisms. As was mentioned in Section 4, our model considers all security mechanisms to be filters that take some input stream and either accept or reject each string, with an output language that is a subset of the input language.

Clearly not all security mechanisms behave in this way, and there are mechanisms that transform or augment data. An example here is a cryptographic cipher, which transforms its input language in a way that can be reversed only by the same cipher and key (under the usual cryptographic assumptions). This simplifying assumption was useful to allow us to focus our attention on other details, such as the composition of mechanisms and the layout of the system. However, it may be the case that capturing these details is important for future work in Defense Graphs, and future iterations of the model may consider these differences.

A limitation of any model for a complex system is that it loses some fidelity of information. The system that we are attempting to model is a composition of already complex (Turing-complete) systems, and so the system as a whole is at least as complex as its constituents. Any model that we can practically use to make predictions about a system must simplify the underlying system, if the system is sufficiently complex. These simplifications inevitably cause some information to be lost.

In our case, we lose information about the individual mechanisms, such as whether they are properly implemented and devoid of software flaws. Other techniques exist for analyzing the individual components of a system (such as static and dynamic analysis to verify some properties about software, or parse-tree differentials [20] to verify that two implementations of recognizers for a formal language actually do implement the same language), so our model leaves these details outside of its scope.

8. CONCLUSION

In this paper we present a graph-based model for Defense in Depth, Defense Graphs. Our model makes explicit the compositional relationships between security mechanisms, and can reveal points of weakness and emergence in the system. As the composition of security mechanisms is known to be unreliable in many cases, identifying these points is vital to understand the limitations of a layered security posture.

Four properties are exhibited in a Defense Graph- Independence, Redundancy, Coverage, and Cost. These properties concisely describe the essentials of the system in question, and arise from the layout of the system, the languages of the mechanisms used, and an empirical analysis of the system's cost. The properties can be used for simple analytical comparison between security architectures, and can be used to predict some of the behavior of the system (assuming the Defense Graph is correctly instantiated).

Composition of mechanism is a core focus of the model, and in particular the model identifies a number of compositional structures with different properties. Composition can be either deterministic, with a reliable order of mechanism, or non-deterministic, where this order may not be consistent. Though in either case the security policies may not be correctly enforced (a known result in policy composition), the latter case of non-deterministic composition may exacerbate this incorrect behavior due to emergent properties.

By identifying these points of composition, a system designer can better understand the limitations of their security posture, and can design systems in a more principled manner. A principal contribution of this model is to force an explicit look at the system's design, and the points where these compositions may be exhibited.

We apply our model to a number of different scenarios to establish its explanatory and predictive capabilities, and show that a number of security first principles can be derived and explained from within our model. In doing so, we show that our model is useful not only from a formal perspective, but also from a practical one, and can be a vital tool to support principled system design.

Acknowledgements

We appreciate the feedback of the anonymous reviewers, who helped clarify the structure of our arguments and pointed out a number of ways to improve the model. This work was partly sponsored by Canada's NSERC Discovery Grant program.

9. REFERENCES

[1] AMTSO feature settings check. `http://www.amtso.org/feature-settings-check.html`.

[2] Defense in depth. `http://www.nsa.gov/ia/_files/support/defenseindepth.pdf`.

[3] Defense in depth. `http://www.sans.org/reading-room/whitepapers/basics/defense-in-depth-525`.

[4] Metascan. `https://www.opswat.com/products/metascan`.

[5] Recommended practice: Improving industrial control systems cybersecurity with defense-in-depth strategies. `https://ics-cert.us-cert.gov/sites/default/files/recommended_practices/Defense_in_Depth_Oct09.pdf`.

[6] Abadi, M., and Lamport, L. Composing specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS) 15*, 1 (1993), 73–132.

[7] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on* (2001), IEEE, pp. 136–145.

[8] Cohen, F. Computer viruses: theory and experiments. *Computers & security 6*, 1 (1987), 22–35.

[9] Datta, A., Derek, A., Mitchell, J. C., and Pavlovic, D. Secure protocol composition. In *Proceedings of the 2003 ACM Workshop on Formal*

Methods in Security Engineering (New York, NY, USA, 2003), FMSE '03, ACM, pp. 11–23.

[10] Heckman, M., and Levitt, K. Applying the composition principle to verify a hierarchy of security servers. In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on* (1998), vol. 3, pp. 338–347 vol.3.

[11] Heintze, N., and Tygar, J. A model for secure protocols and their compositions. *Software Engineering, IEEE Transactions on 22*, 1 (Jan 1996), 16–30.

[12] Jha, S., Wing, J., Linger, R., and Longstaff, T. Survivability analysis of network specifications. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on* (2000), IEEE, pp. 613–622.

[13] Kelsey, J., Schneier, B., and Wagner, D. Protocol interactions and the chosen protocol attack. In *Proceedings of the 5th International Workshop on Security Protocols* (London, UK, UK, 1998), Springer-Verlag, pp. 91–104.

[14] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS) 4*, 3 (1982), 382–401.

[15] Lippmann, R., Ingols, K., Scott, C., Piwowarski, K., Kratkiewicz, K., Artz, M., and Cunningham, R. Validating and restoring defense in depth using attack graphs. In *Military Communications Conference, 2006. MILCOM 2006. IEEE* (2006), IEEE, pp. 1–10.

[16] McCullough, D. Noninterference and the composability of security properties. In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on* (Apr 1988), pp. 177–186.

[17] Prabhakaran, M., and Sahai, A. New notions of security: achieving universal composability without trusted setup. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing* (2004), ACM, pp. 242–251.

[18] Quigley, D. P. Re: Linux security *module* framework (was: LSM conversion tostatic interface), 2007.

[19] Saltzer, J. H., and Schroeder, M. D. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (1975), 1278–1308.

[20] Sassaman, L., Patterson, M. L., Bratus, S., and Locasto, M. E. Security applications of formal language theory. *Systems Journal, IEEE 7*, 3 (2013), 489–500.

[21] Schneider, F. B. Enforceable security policies. *ACM Trans. Inf. Syst. Secur. 3*, 1 (Feb. 2000), 30–50.

[22] Schneider, F. B., Morrisett, G., and Harper, R. A language-based approach to security. In *Informatics* (2001), Springer, pp. 86–101.

[23] Shaufler, C. LSM: Multiple concurrent LSMs, 2015.

[24] Sheyner, O., Haines, J., Jha, S., Lippmann, R., and Wing, J. M. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on* (2002), IEEE, pp. 273–284.

[25] Sommestad, T., Ekstedt, M., and Johnson, P. Cyber security risks assessment with bayesian defense graphs and architectural models. In *HICSS* (2009), pp. 1–10.

[26] Uribe, T. E., and Cheung, S. Automatic analysis of firewall and network intrusion detection system configurations. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering* (2004), ACM, pp. 66–74.