Towards a Theory of Software Diversity for Security

By

Saran Neti Maruti Ramanarayana

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario, Canada

September 2012

Abstract

Although many have recognized that software monocultures are a major impediment to improving security, it is currently unclear how much and what kind of diversity is needed to counteract this problem. This thesis provides a framework for investigating software diversity in the context of security. We propose a simple model of a software ecosystem using sets of hosts and vulnerabilities represented as a bipartite graph, and we exhibit a few examples of software security problems formulated precisely enough in this model to admit rigorous analysis. We propose software diversity metrics using entropy, illustrate a calculation using real world data, and analyse six popular defence techniques for their effect on inhibiting large scale attacks. We then employ game theory to understand why computer host owners would choose to diversify and propose a slight variation of the popular security game Capture the Flag to determine users' underlying utility functions.

Acknowledgements

First and foremost I would like to thank my adviser, Anil Somayaji, for giving me an opportunity to work with him. During our many discussions on contrasting living systems with software systems, my attitude towards biology fluctuated wildly from an utterly hopeless endeavour to a deeply fascinating phenomena and it was his seasoned forbearance and an unswerving intuition that kept me grounded. Anil often expressed his desire to see more autonomous and self-and-situation aware computers by likening them to cats and dogs, and although this work is light years away from creating pets out of laptops, I hope it scratches the right itch. Throughout my stint at Carleton, I've been a recipient of generous support and a non-deterministic amount of space and time in which to intellectually wiggle my tail, and for these, my gratitude towards Anil is beyond ability to embellish.

I would like to thank Jeremy "Leiske" Clark and members of the Carleton Computer Security Lab for useful discussions and pointers to relevant conferences. This work has been largely tangential to popular lab interests but this thesis might not have met the deadline if not for David Barrera's daily morning greeting: "finished your thesis, Saran?". Conversations with Michael Locasto have been instrumental in explicating some of the notions in software diversity. I would also like to thank Anil Maheshwari for his guidance and frequent drop-in chats.

Finally, I would like to thank my friends and family for putting up with my eccentric arguments, intelligently counter arguing, and constantly keeping me engaged. I hope that some of my mom's convivial nature and my dad's intellectual abilities have percolated down to me. Along with my sisters, Moti and Mami, my family constitutes what I can best describe as a fixed point of my life. I thank you all for your support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Software systems that power billions of computers around the world have grown enormously in size and complexity over the past two decades. Ensuring the security of these systems is critical to the functioning of virtually all major organizations in a modern society. Despite efforts to the contrary, a large number of security incidents occur each year costing the community billions of dollars.

A fundamental property of software that makes it economical to produce is the virtually zero cost of making copies. This enables a developer to amortize development costs and is responsible in no small part for the explosive growth of the computer industry. Unfortunately, this write-once-copy-endlessly property of software has a flip-side for security. It enables a single piece of malware to compromise a large number of systems by exploiting vulnerabilities in a widely replicated piece of code. Malware writers are conferred an asymmetric advantage over defenders: an attacker needs to find just one vulnerability to compromise millions of hosts while the defender is tasked with eliminating all possible vulnerabilities to secure a system.

Security researchers have proposed that increasing the diversity of software running on the installed hardware base would prevent large scale catastrophic security breaches [82, 71, 79]. In other words, if computer systems appear different to exploits, many more exploits would need to be created to compromise the same number of systems. The resulting increase in malware creation costs would contribute to levelling the efforts expended by attackers and defenders.

A few defences have been proposed with the intent of increasing diversity. The purpose of this thesis is not to develop another technique to make systems diverse, but rather to make a first step towards a more formal understanding of diversity in software systems in the context of security.

1.2 Motivation

Traditionally, security breaches have been regarded as one-off events with potentially severe undesirable consequences. Budgetary estimates often exclude them due to their unpredictable nature in occurrence and impact. But the onslaught of attacks endured by software year after year forces us to treat security breaches as part and parcel of maintaining a software system. Annual security incident reports and their corresponding financial cost estimates provide us with evidence of the systemic problem of security in software [62]. Software vulnerabilities in popular software continue to be discovered at a prodigious rate and exploits taking advantage of them continue to find ways of getting around increasingly sophisticated defensive mechanisms. In the process, what has been left effectively unchecked is the scale of damage a single exploit can cause.

Software monocultures have been recognized as a major impediment to improving the security of software systems. Researchers, both in opinion papers [71, 29, 79, 28] and peer reviewed work [82, 27, 25], have observed that software systems in practice suffer from a lack of diversity. However, only a relatively modest amount of research has gone into developing mechanisms to address this problem and of all the techniques proposed, only one class of techniques have gained widespread adoption. Randomization techniques aim to eliminate needless consistency among programs with no incremental cost in deployment. Address space layout randomization (ASLR) [52] is one such example that works at run time by modifying the base memory location of critical data structures like the stack and the heap. When used with sufficiently high entropy, ASLR can render some exploits relying on code running in specific memory locations ineffective, but when the entropy is insufficient, it is susceptible to brute-force derandomization attacks [64]. Randomization techniques are diverse in one sense because they increase effective variability in software but are not diverse in another sense because the difficulty of attack is uniform across the population. In a diverse ecosystem, attack difficulty varies between members of the population.

The concept of diversity is used in theory and practice to a much greater extent in other disciplines. In finance, diversification of stock holdings and investments are employed to reduce risk. In biology, a myriad of different mechanisms for regulating life processes have been discovered that enable differential survival of some organisms in harsh conditions and natural calamities. Diversity is a central theme in ecology, where it has witnessed the most amount of research. Measuring diversity of natural ecosystems and understanding interrelationships between species that affect the stability of such systems are some of the primary problems in biodiversity research. At a smaller scale, studying the potency of pathogens to take down entire populations of species is critical to both epidemiology and agriculture.

Software systems, despite growing to comparable levels of complexity as natural systems, have not been subjected to rigorous diversity analysis. It remains unclear to what extent the defensive strategies developed to protect individual software components are effective in improving the security of a software ecosystem treated as a whole. We believe that investigating the notion of diversity of software systems from a security standpoint would help us analyse current defences, predict the population level success or failure of newly proposed defences, and provide us with metrics for measuring progress in improving the diversity of systems.

1.3 Main Contributions

This thesis has five main contributions to the analysis of the concept of software diversity. Parts of these ideas have been published (jointly with Anil Somayaji and Michael Locasto) [46].

1. **A model a software ecosystem that enables rigorous diversity analysis.** We propose a model of a software ecosystem that emphasizes relationships between hosts and vulnerabilities and allows them to be conveniently represented as a bipartite graph. We discuss five plausible assumptions that enable viewing real world software systems as an extension of the model. We formulate and describe three problems of vulnerability discovery, prediction, and coordination that become amenable to rigorous analysis in this model.

2. **Introducing diversity indices for software.** We formalize the intuitive notion of diversity for software systems by drawing upon the extensive literature of ecology. We discuss two components that comprise diversity indices and propose a generalization of Shannon entropy as a measure of diversity. We show an illustration of how to calculate approximate diversity values of operating

systems and browsers running on commodity hardware using real world data. We discuss implications of diversity on stability measures including resistance, resilience and invasibility.

3. **Analysis of the diversity-impact of current defences.** We analyse the scalability of six popular defences widely used in real world computer systems. These include firewalls, anti-malware signatures, anti-malware heuristics, no-execute memory, address space layout randomization and software updates. We study their impact on the host-vulnerability graph and classify them based on the properties of the graph they modify. We observe that none of these defences alter the distribution of edges in the host-vulnerability graph and thus have no significant effect on diversity.

4. **Introducing game theory in diversity.** Owing to the success of the explanatory power of game theory in economics [75], evolutionary biology [66], and network security [78] we construct and analyse games relevant to diversity that reflect choices available to users. Specifically, we formulate an anti-coordination game and a dispersion game with additional costs to make changes to strategies. Equilibrium values for these suggest that in the absence of external enforcement, suboptimal strategies are likely to be employed.

5. **A methodology for capturing diversity related user preferences in security.** In the real world, choices made by consumers and developers are not often driven by a primary focus on security. Other factors like usability and functionality play an important role. This inhibits employing real world data to deduce user preferences regarding diversity parameters for security. Therefore, we present a simple modification to the popular security game, Capture the Flag, that will enable us to obtain data about revealed preferences in a security-

focused setting.

1.4 Organization of Thesis

The remainder of this thesis is structured as follows. Chapter 2 provides the relevant background needed to analyse diversity. It summarizes research findings on diversity in Finance, Agriculture and Ecology and also discusses previous attempts to introduce diversity in computer systems. (Game theoretic background and related work are presented in Chapter 6.) Chapter 3 introduces the software ecosystem model that is useful for analysing diversity. A list of simplifying assumptions and some open problems are also discussed in this chapter. Chapter 4 introduces the reader to the concept of vulnerability overlap, an essential aspect of diversity. We propose generalised entropy as a measure of diversity, and using real-world data, show an illustration of one such calculation. We also discuss diversity-stability relationships. Chapter 5 systematically classifies and analyses six popular defensive strategies and their effect on diversity. Basics of game theory are introduced in Chapter 6. Analysis of two simple games provides a descriptive justification of why maximal diversity is unlikely to naturally arise. We also proposes diversity-inspired modifications to a popular playable security game to enable us to collect real world data on user preference in the context of a cyber-conflict. Chapter 7 discusses limitations and ideas for future work.

Chapter 2

Background and Related Work

This chapter provides background on research related to diversity in computer science and other disciplines. In the first part of this chapter we review literature on analysis, experiments, and practical use of diversity in three different fields that have studied it in depth: finance, agriculture, and ecology. In finance, diversification strategies are often employed to reduce risk and some past work provides a mathematical justification for desiring increased diversity. In agriculture, we learn about the massive financial impact of monocultures. Several decades of ecological research aimed at gaining a descriptive theory of diversity in species, their interactions, and ecosystem stabilities provides a foundation for diversity analysis of computer systems.

The second part of this chapter reviews diversity techniques proposed for computer systems. We first look at early literature on fault tolerance and approaches to mitigate failures by simultaneously building and running alternative pieces of software. We then review randomization techniques that aim to improve security by eliminating unnecessary consistency among programs running on different systems. We also review some explicit diversification techniques recently proposed and briefly describe a paper that attempts to analyse optimum diversity in organizations. We defer describing background and related work on game theory and its application to computer science until Chapter 6.

Towards the end of this chapter, the reader should note that although diversity has been formally researched in some fields, the software security literature lacks a general framework to understand most aspects of diversity including measures, strategies, outcomes and testing methodologies.

2.1 Diversity in other fields

Diversity as a mechanism for managing risk under uncertain situations is studied and practised in many fields.

2.1.1 Finance

In the financial industry, diversification is important strategy used to reduce risk. Distributing funds in a variety of investments ensures that the aggregate risk is smaller than the weighted average of the risks of individual investments. As long as investments are not strongly correlated, an investor with a diversified portfolio is protected from the failures of a few constituent investments. A mutual fund is an example of a collective investment scheme that relies on diversity to provide increased assurance of returns. A mutual fund provides opportunity for individual investors with limited funds to pool money and purchase a variety of financial securities like bonds, stocks or derivatives. In a volatile market, this scheme facilitates investors to trade off risk for reduced returns.

The financial literature provides a formal justification for preferring increased diversity by generalizing the following observation: investing a fixed total wealth into identically distributed investments will leave the mean unchanged but will minimize the variance. Assuming a concave utility function, it follows that all independently distributed investments that have a common mean must enter the optimal portfolio [60]. Diversification is effective when asset returns have near zero or somewhat positive correlations. Brumley et al. give a precise condition for the case where independence assumption of investments is dropped [11]. Consider an investor who has two investment options represented as random variables X and Y. She chooses to invest a proportion, λ, of her total wealth in X. If U is a concave utility function, then $E\left(U(\lambda X + (1-\lambda)Y)\right)$ doesn't achieve a maximum at $\lambda = 1$ (i.e some funds should also be invested in Y) if and only if $E\left(XU'(X)\right) < E\left(YU'(X)\right)$. This provides the necessary and sufficient condition for choosing between one investment or distributing the total wealth in both investments even if their returns are correlated.

2.1.2 Agriculture

Monoculture, or a lack of diversity, was identified as a major problem in the agriculture industry. A single variety of seed planted extensively to produce uniform yield made it possible to automate much of processing but also made the entire harvest susceptible to complete destruction by a single pest [22]. For example, in the 1970s, over $1 billion worth of harvestable corn crop was destroyed in the U.S by a single pathogen [33].

Experiments conducted in European grasslands to measure effects of plant diversity on productivity reveal that communities with fewer functional groups are less productive [31]. Two primary factors responsible for over-yielding, where the total biomass of a mixture of species exceeds the most productive monoculture biomass, were determined to be niche complementarity and positive mutual interactions. Coexistence of ecologically different species occupying different niches leads to a more thorough utilization of resources.

Polyculture is an agricultural movement that aims to contain and possibly reverse arable land degradation and soil erosion by planting multiple crops in the same space. The hypothesis is that by imitating diverse natural systems, agriculture can be made sustainable over longer periods of time. Self regulating features of natural ecosystems are believed to be responsible for decreased incidence of plant diseases and corresponding increase in yields [39]. An experimental study in China reported over 80% increase in yields of rice crops when several varieties of rice were planted in the same fields [83]. This result was primarily attributed to a dramatic 94% decrease in disease rates making pesticides redundant.

2.1.3 Ecology

Ecology is the study of interrelationships between organisms and the environment in which they coexist. Meticulous observations and efforts to classify species around us has produced large data sets of valuable information that has not only led to great conceptual breakthroughs [21] but also initiated research into stability, richness and sustainability of natural ecosystems. Creating ecosystem models and theories for predicting habitat survival rates, corroborating theories with empirical data mined from large data sets on geographically dispersed ecosystems, and more recently, a desire to understand the effect of humans on the loss of biodiversity has made the field of ecology a leader in understanding and experimenting with diversity. We borrow heavily from this literature.

Fischer initiated the study of diversity in 1940s in a paper that studied the relation between the number of species and the number of individuals in an animal population [24]. Soon, researchers began investigating correlations between several other parameters of a variety of ecosystems. This resulted in the creation of a diverse set of diversity indices, with often inconsistent interpretations, by combining various aspects of ecosystems including species richness, relative abundance, interaction strengths and sampling size. It wasn't always clear what diversity measure to use or what each of them represented. Anne Magurran provides a consistent assimilation and simplification of a large number of diversity measures and their interpretations [43]. We also use Hill's unifying notion of information theoretic measures [32] as a measure of diversity in software ecosystems.

Quantifying the utility of diversity is still an active area of research in ecology. Relationships between diversity, stability, resistance, resilience and other potentially desirable outcomes are highly complex [54, 72]. McCann provides a comprehensive overview of the diversity-stability debate [44]. It is unclear how observations on diversity-stability relationships in ecology could be directly applied to computer systems. We explore some possibilities in Section 4.4.

2.2 Diversity in Computer Science

2.2.1 Fault tolerance

Even before computer security was a major concern, the problem of working with single implementations was recognized in the fault tolerance community. In the mid-1980s, the n-version programming approach to fault tolerance was advocated where multiple development teams would develop versions of the same software. These would be executed in parallel and provided the same inputs [3]. Outputs were aggregated for each input, and when they weren't identical, a majority vote would serve as an arbitrator. This approach attempted to reduce the probability of failure due to a single bug, but experiments showed that different versions created continued to exhibit considerably higher similarity than expected: the independence assumption was called into question [37].

In implementations, a few technical challenges were observed, including comparing floating point outputs [10], while other experiments showed that using different programming languages or development environments had no major impact on the incidence of correlated failures [9].

2.2.2 Randomization techniques

Applications of principles of diversity for computer security was initially inspired by biological processes [26]. Their implementations focused on removing unnecessary consistency in software without affecting usability. An early example of this technique was able to thwart simple buffer overflow attacks by randomly increasing the amount of memory allocated on a stack frame [25]. A more robust compile-time improvement, Stackguard, that places and checks for random canary values before returning from a function call is now a standard feature in GCC [19]. Address Space Layout Randomization (ASLR) extends this concept to various memory structures and is widely deployed in many current operating systems.

Instruction set randomization is another technique that follows the same philosophy but prevents code injection attacks. By using emulators [4] or hardware support [35], instruction-opcode mapping is altered in a pre-determined way using keys. The same keys are then used to encode machine independent executable binaries into their corresponding randomized instruction set. Compile-time and run-time randomization techniques take programs that might run on millions of systems and make their instances individually distinct in some aspects. While they are usually inexpensive to implement for the increased security they offer, a single point of failure can negate most of the protection provided, just like in cryptography. For example, ASLR on 32-bit systems was found to be susceptible to de-randomization attacks due to insufficient entropy [64].

Along with the above mentioned preventive measures, automatically detecting and generating patches for infected systems has been proposed [65]. In some limited cases it is even possible to automatically generate patches for zero-day exploits using genetic algorithms [77]. While these patching techniques are not explicit diversity type defences, they do monitor and alter program behaviour differentiating themselves from other instances at runtime.

2.2.3 Diversification techniques

As early as 1993, Cohen observed that the static nature of defences cannot hold up against the dynamic nature of attacks and proposed several techniques for modifying programs to achieve security through obscurity [16]. After describing many techniques like instruction reordering, variable substitutions, garbage insertion, instruction encodings and so on he concludes that a mathematical analysis of attacks and defences is needed to understand what we're trying to conceal in security and to what extent evolutionary techniques are effective in concealing them.

Explicit use of algorithms to increase diversity by assigning software packages to nodes in communication networks was proposed by O'Donnell and Sethu [49]. Their contention is that by ensuring neighbouring nodes run distinct software as far as possible, the spread of worms and viruses through email networks or client server file shares would be minimized. This problem nicely translates into the graph colouring problem, which is to assign colours from a given set to vertices of a graph in order to minimize the number of pairs of vertices that share a colour. In the distributed algorithms they propose, nodes communicate with their neighbours to choose one among the available software packages that are as disparate as possible. Interestingly, in their simulations they observe that hybrid algorithms outperform the rest indicating that methods used to introduce diversity in systems themselves need to be diverse.

More recently Cox et. al. proposed n-variant systems, a framework for introducing diversity in software systems, that enables simultaneous execution of automatically modified versions of programs while preserving semantics [20]. By loading different versions in non-overlapping memory addresses or by using different tags for instructions in each application, some attacks that use memory access or code injection are prevented. It has also been suggested that compiling programs along multiple code paths to create several different versions might be used to improve security through diversity [27].

2.3 Software Diversity Analysis

The software engineering literature includes several methodologies of design, implementation, analysis and verification of software variants in product lines. Schaefer et. al. provide a comprehensive overview of the state of the art of software diversity in software engineering [61] including variability modeling, diversity in architectures and implementation, quality assurance and model checking.

The software security literature, however, has witnessed little research on analysis of diversity. Chen et al. argue that software monoculture can be understood as a balance between positive externalities, like interoperability, and negative externalities, like increased risk of attack [14]. By assuming that the loss experienced by a firm increases quadratically with the number of computers compromised and that the scale of the incident (i.e. number of computers affected) follows an exponential distribution, they calculate the proportion of new software with respect to incumbent software that computers in a firm must use to minimize loss. They claim that not just the variance of loss, but even expected loss can be reduced by increasing diversity.

Having made the observation that security is often employed as an after thought by a majority of computer users and administrators, Cowan et al. analyse the effectiveness of several *post-hoc* security techniques by classifying them into four major categories [18]. They argue that interfaces and implementations are two major parts of software that are subject to obfuscation and access restriction techniques for improving security. By categorizing security techniques like file system access controls, firewalls, static and dynamic type checking, chaffing and winnowing, random code or data layout into four quadrants of interface restrictions, interface obfuscation, implementation restriction and implementation obfuscation, they claim that restriction techniques, which enforce strict rules to control known pathways of communication between different components of software are almost always more effective and cheaper to implement than obscurity techniques, which aim to deter a larger class of attacks but succeed only a majority of the time as opposed to all the time.

2.4 Discussion

This chapter has outlined two distinct themes that have emerged from a survey of diversity literature. In fields other than computer science where diversity is studied, there is scepticism, but appreciation for the concept that has stimulated much empirical and mathematical research. In contrast, computer science has witnessed very little research on diversity, especially from a security standpoint. A significant number of researchers resonate with the views of Cowan et. al. who suggest the use of restriction based techniques for improving security almost exclusively. This is most certainly appealing from a practical perspective, and perhaps effective in the short-term when attack methods are known *a priori*; but in the long term, this only escalates the arms race of creating and breaking increasingly higher barriers. A stalemate situation is reached where the following two major problems of computer security remain unsolved:

- Preventing large scale catastrophic damages because of new attacks that can surmount restrictions.
- Making computer systems more autonomous by not requiring human intervention every time there is a security breach.

Attempting to resolve both these problems requires a balance of flexible but sometimes unreliable adaptive techniques and the more rigid but reliable restriction based techniques. Security researchers familiar with complex adaptive systems like natural or biological systems argue for the former [67, 68], while those coming from a practical viewpoint typically have a strong preference for the latter [18]. Unfortunately very little progress has been made towards a resolution because of a lack of adequate tools in our current security literature. The primary purpose of this thesis is to provide some of these tools: diversity measures on software ecosystem models, game theoretic arguments for analysing strategies and expected outcomes, and a way to gather real-world data about utilities related to diversity.

Chapter 3

Model and Assumptions

3.1 Introduction

In the real world, a software ecosystem is composed of hardware developers, software and firmware developers, alpha and beta testers, end users and devices. Each entity operates under functional and financial constraints and plays a part in affecting the security of the ecosystem. Their complex interrelationships make it extremely hard to model all aspects of an ecosystem simultaneously.

In this chapter we propose a simplified model of a software ecosystem that enables us to analyse diversity from a security standpoint. This model is composed of the two most important entities that are related to security: hosts and vulnerabilities. These entities can be highly interconnected in many kinds of distributions allowing a range of possibilities for the diversity of the ecosystem.

The rest of this chapter is divided into three parts. We first describe our model both in terms of sets and as a bipartite graph. Then we list assumptions under which this model is valid while commenting on the plausibility of these assumptions in the real world. Finally we illustrate the utility of this model by posing various problems of interest to the security community in the context of this model.

3.2 Model

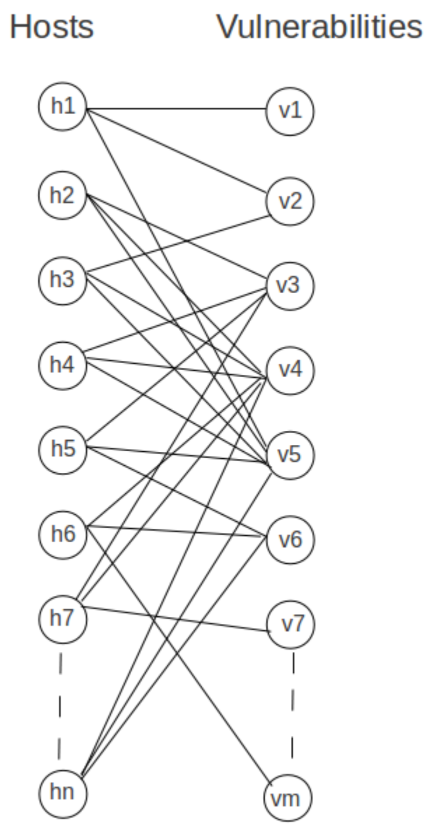


Figure 3.1. An example of non-uniform vulnerability distribution with n hosts, m vulnerabilities and 3 vulnerabilities per host

Our model consists of a set of n hosts H, a set of m vulnerabilities V and a mapping from every host to a subset of V. This is easily visualized as a bipartite graph, an example of which is shown in Figure 3.1. Hosts and vulnerabilities comprise vertices of the bipartite graph and an edge between vertices corresponding to a host h_i and a vulnerability v_j indicates that h_i runs software that contains v_j.

This model contains several simplifications. First, by associating hosts directly with vulnerabilities, we've avoided the need to model software containing those vulnerabilities. Although we are interested in software diversity, our focus is primarily on the effect of diversity on security and to that end we treat software as a set of vulnerabilities.

When a vulnerability is exploited, all hosts connected to it can be compromised. For example, in Figure 3.1 vulnerabilities v_4 and v_5 are highly connected and have the potential to affect 6 hosts each, while vulnerabilities v_1 and v_2 can only affect 1 and 2 hosts respectively. In the real world, where the number of computers, n, might be in millions and the number of vulnerabilities, m, might be in thousands, the number of hosts that vulnerabilities can compromise can exhibit high variability. In other words, the distribution of the degree of vulnerability vertices could be highly skewed or highly even. Quantifying this variability gives us diversity measures as we will see in Chapter 4.

3.3 Assumptions

Several simplifying assumptions are implicit in this model. Here we list all those assumptions, discuss their plausibility and their necessity.

3.3.1 Computer equivalence

We assume that no host in the host-vulnerability graph is privileged or has control over other hosts. For example, in automatic software updates, if an attacker compromises the central software distribution server, he may gain control of all clients that pull software from that server. In our model, we represent this as a vulnerability connected to hosts corresponding to the software update server and all clients that depend on it. There is no special distinction for the host that physically contains the vulnerability. This assumption holds for other hierarchical network configurations like subnets behind a NAT or a firewall.

3.3.2 Software-vulnerability equivalence

We assume that vulnerabilities cannot be completely eliminated from a large software stack and that to bring a computer to a usable state a significant amount of code needs to run. In other words, it is assumed that techniques like effective formal verification on the entire software stack of commodity computers is infeasible. Reliability literature indicates that this might indeed be the case [12]. This assumption enables us to avoid modelling complex software-vulnerability relationships and denote a host's choices only in terms of vulnerabilities. Instead of saying a host chooses software, we say a host chooses vulnerabilities.

It is to be noted that this assumption places constraints on the applicability of the model to real world software systems because in reality hosts do not get to choose individual software components each containing a single vulnerability and mix them in any manner to produce desired functionality. Moreover, software is layered into firmware, kernel, user space libraries, applications and so on and while it is possible to trade off functionality by moving components between layers, alternatives of software available for users to choose from are often written to work in only one layer. These two factors reduce the choice granted to a user in the real world as compared to our model.

This assumption can be relaxed by introducing a *component* layer between hosts and vulnerabilities so that hosts don't choose vulnerabilities directly, but only components which are connected to vulnerabilities. The resultant reduction in choice would reflect the real world case better, but it would also necessitate introducing a *granularity* parameter in our model to control the extent to which software can be divided into inter-operable components. We discuss this further in Section 7.3.1.

3.3.3 Vulnerability criticality

In order to create a successful exploit, an attacker might have to rely on not just one vulnerability in one software, but a series of vulnerabilities leading up to a host's compromise. In our model, we assume that all vulnerabilities can be exploited individually and any one is sufficient to compromise hosts connected to it. In the case where several vulnerabilities need to be chained to create an exploit, we combine all those vulnerabilities and treat them as one vertex in the host-vulnerability graph.

The other aspect of the *Vulnerability criticality* assumption is that the result of exploiting a vulnerability is binary: either an attacker achieves his goal or he does not. We do not model the various degrees to which a host can be compromised. There is some practical evidence to support this assumption. The goal of many exploits is to return a shell prompt to the attacker where arbitrary code execution is allowed with either user or root privileges. For example Metasploit, an offensive security toolkit, provides a mechanism whereby most exploits written in it result in dropping the user to a shell [1].

3.3.4 Vulnerability equidistribution

Every host in our model is required to choose the same number of vulnerabilities—every host vertex $h_i \in H$ has the same degree k. This statement is consistent with both *Computer equivalence* and *Vulnerability criticality* assumptions but is independent of them. It asserts that two different alternatives of software that can be substituted for one another have the same number of vulnerabilities.

This assumption is also an idealization of the real world as there are several cases where security-hardened alternatives to mainstream software are available which trade off usability, configuration complexity or performance for a reduced number of vulnerabilities. If we were to accommodate this in our model, we'd need a measure to evaluate software security. In the context of our model, this translates to sourcing a non-uniform distribution of host vertex degrees from the real world in order to reflect the market share values of computers as their effective security varies. Despite significantly increasing the complexity of our model, it is currently unclear to what extent diversity related measures and outcomes would be effected if this is taken into account. In our preliminary investigation of diversity for security we're more interested in the distribution of vulnerabilities among hosts than the relative security of alternatives available. Therefore we do not believe that this assumption cripples our model significantly.

3.3.5 Steady State assumption

Our analysis is confined to steady state where the number of vulnerabilities that exist do not change over time. This assumption highlights the ontological status of vulnerabilities in our model: they will continue to exist and can be found and exploited by expending effort.

Implication of this assumption in our model is that if software corresponding to a vulnerability v gets compromised then $deg(v)$ hosts are affected, but in the next iteration of analysis, everything remains unchanged— v is not removed from V. One interpretation of this is that a security patch might itself have a vulnerability or that new vulnerabilities might be discovered in the same software.

There has been some past work to empirically check if the security of software in the real world is indeed improving over time. Rescorla argues that software security does not get better with time [56], but Ozment and Schechter report that vulnerabilities introduced prior to the initial version release of OpenBSD have continued to decrease albeit with a median lifetime of over 2 years [51]. However, a security report indicates that the exponential growth in the number of vulnerabilities reported until 2006 has essentially flattened [62].

While the answer to *"Is security of computer systems improving?"* remains vague at best, our interest in this model is not primarily to study the deletion of host-vulnerability edges but to understand the causes and effects of their re-organization.

3.4 Open Problems

While our model and assumptions are a great simplification of the real world software ecosystem, they do allow us to capture the dynamics of commodity systems running mainstream software that are in a constant state of code flux. In this section, we discuss three possible broad questions of interest to security that might have implications for diversity. We do not attempt to solve these problems here but merely express them using our model and discuss a few observations.

3.4.1 Vulnerability Discovery

To discover vulnerabilities and create exploits, attackers need to spend non-trivial amount of effort either by directly investing time and expertise in investigating software or by spending money to buy zero-days. Some security firms estimate that zero-day vulnerabilities and exploits can cost attackers between \$10K-\$100K, making it a fairly serious investment [2].

One of the questions attackers face is which software to investigate for vulnerabilities given the expected number of hosts each software can be used to compromise and the expected difficulty in finding vulnerabilities in that software. Suppose we relax part of the *Vulnerability criticality* assumption so that some vulnerabilities might be easier to find than others: let the random variables $T_f(v_i)$ and $C_f(v_i)$ be the time and cost of finding a vulnerability v_i in software. Using estimates from previous vulnerabilities, attackers might have a way to estimate the expected values and maybe even the distribution. Their task is to allocate a given amount of resource to attempt to discover one or more vulnerabilities to maximize the total number of hosts that can be compromised. Attackers would have to predict how $deg(v_i)$ would change with time, subject to a constant $\sum_i deg(v_i)$. This problem of allocating resources to discovering vulnerabilities is also applicable to white hat security researchers.

3.4.2 Vulnerability Prediction

Just as attackers have to figure out how vulnerability distribution among hosts changes with time, defenders can actively switch one software for another to avoid being the target of attacks as much as possible. In our model, we reduce the problem of predicting which software might get compromised in the future to the well known paging problem.

Suppose a host keeps track of vulnerabilities compromised over time $w_1, w_2, w_3 ..., w_t$ where $w_i \in V$. In our model, by the *Software-vulnerability equivalence* assumption, a host's requirement to run software can be viewed as selecting k vulnerabilities from V to fill its *cache*. It's objective is to minimize the probability that the cached vulnerabilities will be compromised in the future.

Contrast this with the paging problem: as the CPU requests a series of pages $p_1, p_2, ..., p_t$ the caching algorithm has to store k pages in its cache so as to maximize the probability that they will be requested in the future. This problem is well studied in operating systems literature because it is expensive to make fast memories in large sizes. Modern CPUs have several levels of cache memory in increasing order of size and decreasing order of speed. When the CPU requests a page that isn't in the cache, a page miss occurs at which point the page replacement algorithm decides to evict a page in the current cache. The page replacement problem is an online problem for which several strategies are known: First in First out, Least Recently Used, Random and other variants that use clocks and counters. While the optimal algorithm requires complete knowledge of future requests and hence can't be implemented, strategies like Least Recently Used can be shown to be worse off than optimal only by a constant factor [7].

The vulnerability prediction problem can be very simply translated into the paging problem by considering every host to have a cache of vulnerabilities that are not connected to it. From this viewpoint, it is easy to see that against oblivious adversaries, a strategy of evicting the *Least recently used* page when a page fault occurs corresponds to getting rid of the *Most recently attacked* vulnerability when an attack occurs. The difference between these problems is that the CPUs are passive and usually not malicious towards memory whereas attackers can adaptively change their strategies. Further analysis is required to investigate if these two seemingly unrelated problems have any deeper connection.

3.4.3 Vulnerability Selection and Coordination

In the Vulnerability prediction problem each host was tasked to select vulnerabilities based on history of attacks. In addition to that, if hosts have the ability to select vulnerabilities by coordinating with each other, the total expected number of attacks could be reduced.

The problem is what protocols would hosts use to distribute vulnerabilities among themselves. A central point of distribution can lead to a complete security breach if compromised. Therefore a single point of failure cannot be an integral part of a system that aims to facilitate increased diversity. We look at a simple problem where two hosts cooperate to choose vulnerabilities in a distributed fashion. Consider the case where two hosts h_1 and h_2 decide to pick k vulnerabilities each from V, $2k << |V| = m$, such that they end up with no vulnerability in common. That is, $V_{h_1} \cap V_{h_2} = \emptyset$, where V_h is the set of vulnerabilities chosen by host h. Assume that both hosts know all elements of V and can uniquely identify them. If both hosts are honest, then they can agree to pick the first k and the second k vulnerabilities respectively. But it might be the case that one of the hosts is malicious and is attempting to trick the other honest host to choose vulnerabilities it wishes to propagate. Therefore what is required is a protocol where two honest hosts should be able to choose distinct vulnerability sets, but if one of the hosts is malicious, the best it can do is a random choice, i.e $\Pr(|V_{h_1} \cap V_{h_2}| = x) = \binom{k}{x}\binom{m-k}{k-x}$.

Firstly, if a host has a deterministic strategy, then it is clear that a malicious host can choose the same set of vulnerabilities and cause $|V_{h_1} \cap V_{h_2}| = k$. One suboptimal solution is the following simple strategy: h_1 chooses k vulnerabilities randomly from the first $m/2$ vulnerabilities in V, and h_2 chooses k vulnerabilities randomly from the second $m/2$ vulnerabilities in V. This assures no vulnerabilities are in common. If one of the hosts is malicious, the only thing it knows is which half of V the other host's vulnerabilities are from, i.e $\Pr(|V_{h_1} \cap V_{h_2}| = x) = \binom{k}{x}\binom{m/2-k}{k-x}$. Further work is needed to investigate if h_1 can communicate with h_2 to choose different vulnerabilities without revealing its random bits. Investigating variants of zero-knowledge protocols might provide improved results.

Chapter 4

Diversity measures

In this chapter we review work on measuring diversity in ecological literature and propose diversity measures for software ecosystems based on the simplified model described in Chapter 3. Measuring diversity of species in various habitats is of tremendous interest to researchers in ecology. Study of diversity in ecology started with Fischer in the early 1940s. As increasing amount of information on species and their relative abundance was becoming available, there was a need to fit the data collected using theoretical distributions [24]. Several decades of effort to aggregate and explain ecological data has resulted in a diverse set of diversity indices [43]. Although the concept of diversity might seem intuitive to many, a concrete definition has proven so elusive that it has even been called a 'non-concept' [34]. However, owing to its importance in understanding other phenomena, unification of various measures and their interpretations are still actively researched [57]. Here we aim to provide clarity to the word *diversity* in relation to software security.

This chapter is divided into four sections. In the first, we examine possible questions about software security that might have answers related to diversity. The second part introduces diversity indices that could be used to calculate diversity numbers for software ecosystems and the third part provides a simplified illustration of such a calculation using real world data. Finally, we discuss the impact of software diversity on security by drawing upon extensive work on diversity-stability relationships from ecological literature.

4.1 Vulnerability overlap

An important aspect of the diversity of a software ecosystem from a security standpoint is the number of vulnerabilities that hosts have in common. When several hosts share vulnerabilities, an attack that exploits any common vulnerability can compromise all those hosts. In fact, as the extent to which vulnerabilities are shared increases, there is a concomitant increase in the scalability of attacks that utilize those vulnerabilities. The phrase *vulnerability overlap* aims to capture this intuitive notion. In this section, we explore how vulnerability overlap motivates the statistical interpretation of entropy as a measure of diversity (as discussed in Section 4.2).

More concretely, by vulnerability overlap, we mean the number of vulnerabilities that a given set of hosts have in common. Potency of vulnerabilities critically depends on this number and one might ask how security of an ecosystem would be affected as we vary the overlap while keeping all other parameters constant. Suppose we fix the number of hosts, n, the number of vulnerabilities m, and also the number of vulnerabilities per host, k (i.e effectively fixing the number of vertices and edges in the graph). The number of ways in which we can distribute m unlabelled vulnerabilities among n indistinguishable hosts, so that each host gets k vulnerabilities is the same as the number of ways to partition the integer nk into m non-negative integers, which is $\binom{nk-m-1}{m-1}$. Even for relatively small numbers, like $n = 100,000$ hosts, $m = 300$ vulnerabilities and $k = 10$ vulnerabilities per host, this number is a gigantic 10^{1182}.

If we want to label and distinguish vulnerabilities, this number is even larger by a factor of $m!$. Of all these possible configurations of the software ecosystem, those that have a skewed distribution are far fewer than those that are more evenly distributed leading us to the statistical interpretation of entropy as a measure of diversity. For example, if 80% of the vulnerabilities are distributed evenly only among 20% of hosts, as opposed to 80% of the hosts there are 10^{168} fewer possible configurations.

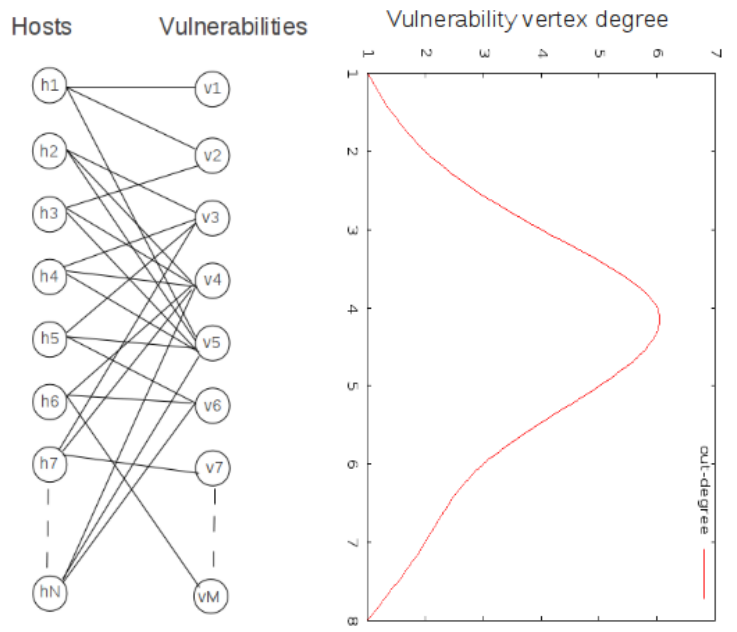


Figure 4.1. A host-vulnerability graph where vulnerabilities are normally distributed among hosts.

One might be interested to know how drastically vulnerability overlap changes as the ecosystem becomes more evenly distributed. For example, we can ask on an average, how many vulnerabilities two hosts have in common and how does it change as vulnerabilities are increasingly dispersed?

Consider a software ecosystem as shown in Figure 4.1 where our n and m values are as above. We vary the number of vulnerabilities per host k between 5 and 60 by varying the degrees of vulnerability vertices using a normally distribution, i.e $deg(V) \sim \mathcal{N}(\mu, v^2)$. Using a Gaussian distribution allows us to change the amount of vulnerability overlap by changing a single parameter, the variance, v. As v is varied between 10 and 90, the average vulnerability overlap between any two hosts decreases. As shown in Figure 4.2, each curve corresponds to one Gaussian distribution. For two randomly chosen hosts h_x and h_y, the curve is a graph of $V_{h_x} \cap V_{h_y}$ on the y-axis and the number of vulnerabilities per host, k, on the x-axis, where V_h is the set of vulnerabilities connected to host h. Computing vulnerability intersection between random hosts was done by selecting 1000 pairs of hosts randomly and calculating the average number of shared vulnerabilities. A higher variance implies more uniform distribution of vulnerabilities, where as a lower variance represents a spikier curve. With a lower variance, a small set of vulnerabilities exist among a significant population of hosts, i.e if we pick two hosts at random the number of vulnerabilities they have in common is high. In fact for the same average vulnerability overlap between two randomly chosen hosts, the number of vulnerabilities per host can vary significantly with variance. For example, as shown in Figure 4.2, a distribution with a variance of 10 (narrow, spiky) can have only 20 vulnerabilities per hosts where as one with a variance of 70 (broader, more uniform) can afford 50 vulnerabilities per host for the same average vulnerability overlap, 10, between two random hosts. In other words, it is possible that a software ecosystem can have a lower average vulnerability overlap, restricting attack scalability, despite each host individually having a higher number of vulnerabilities as long as the vulnerabilities are more or less evenly distributed.

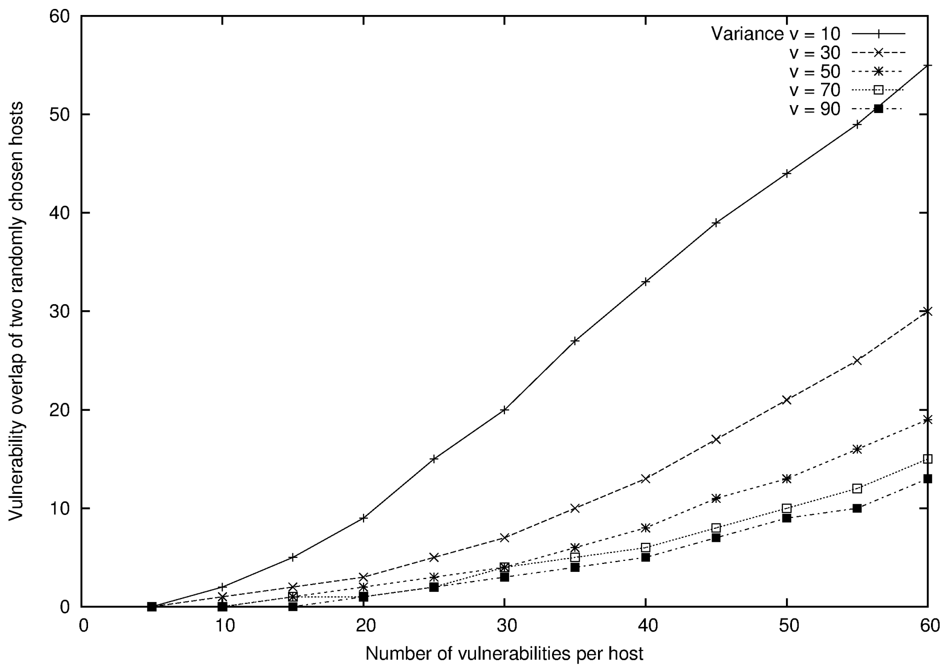


Figure 4.2. Effect of change in the number of vulnerabilities per host on the vulnerability overlap between two randomly chosen hosts for various values of the variance of normally distributed vulnerabilities

4.2 Diversity Indices

It is important to note that that diversity, like randomness, is not an absolute concept but is relative to an observer. In our case, observers are attackers and our software ecosystems need to have just enough diversity to protect against attacks. For example, if all hosts run a common piece of formally verified code which is guaranteed to have no vulnerabilities, then that should not have any effect on diversity measures. This is primarily the reason why we measure diversity for security in terms of vulnerabilities and not some parameters of software like its code density.

Measuring diversity is not an unambiguously straightforward process because of a simple reason—it contains two separate components. First, the absolute number of alternatives available or *richness* and second, the proportion in which each alternative is distributed or *relative abundance*. It computer security terms, it matters that there are 20 different alternatives of software that offer the same functionality as opposed to 2, and it also matters that these alternatives are more or less equally distributed among hosts. The ratio in which these two components are combined leads to a continuum of diversity measures corresponding to Renyi entropy. A special case of Renyi entropy is the well known Shannon entropy.

We now use the bipartite graph G (see Figure 3.1) to represent diversity as follows. If vulnerabilities from V are uniformly distributed among hosts, diversity is high and a compromise of any one vulnerability affects nk/m hosts, while a skewed distribution leads to reduced diversity and, in an extreme case, would let an attacker take down all n hosts using one vulnerability. A diversity index should capture both these extremes and all possibilities in between.

Definitions

If a host h chosen at random finds a vulnerability v connected to it, let p_i be the probability that $v = v_i$. We have $p_i = deg(v_i)/kn$ where $deg(v)$ is the degree of the vertex associated with vulnerability v.

Let the fraction of edges connected to each vulnerability vertex, v_i be p_i. Note that $p_1 + p_2 + ... + p_m = 1$. Simpson's index measures the abundance of more common vulnerabilities and is regarded as a measure of dominance concentration [80]. It is defined as

$$q = \frac{p_1^2 + p_2^2 + ... + p_m^2}{p_1 + p_2 + ... + p_m} \tag{4.1}$$

Shannon entropy is a popular information theoretic entropy measure and is defined as

$$H = -\frac{p_1 \log p_1 + p_2 \log p_2 + ... + p_m \log p_m}{p_1 + p_2 + ... + p_m} \tag{4.2}$$

Both these measures can be expressed as a special case of the following equation.

$$N_a = (\frac{w_1 p_1^{a-1} + w_2 p_2^{a-1} + ... + w_m p_m^{a-1}}{w_1 + w_2 + ... + w_m})^{\frac{1}{1-a}} \tag{4.3}$$

N_a, called the Diversity number [32], is the reciprocal of the $(a-1)^{th}$ root of weighted mean of the $(a-1)^{th}$ powers of p_i. If we set weights w_i themselves as p_i, then we obtain

$$N_a = (\sum_{i=1}^{m} p_i^a)^{1/1-a} \tag{4.4}$$

where the parameter a controls the relative importance given to less connected vulnerabilities and highly connected vulnerabilities.

Renyi entropy is simply the logarithm of the diversity number: $H_a = log(N_a)$. Further, we note the following special cases:

- $N_{-\infty}$ is the reciprocal of the proportional abundance of the rarest vulnerability.
- N_0 counts the number of vulnerabilities.
- $N_1 = \lim_{a \to 1} N_a$ corresponds to Shannon entropy. $log(N_1) = H = -\sum_{i=1}^{m} p_i \log(p_i)$ as in Equation 4.2
- N_2 is the reciprocal of the Simpson's Index defined in Equation 4.1
- N_∞ is the reciprocal of the proportional abundance of the commonest vulnerability.

As the value of a increases from $-\infty$ to $+\infty$, the diversity number N_a decreases the weight assigned to the least connected vulnerabilities while increasing the weight of highly connected vulnerabilities. Before proceeding to discuss diversity-stability relationships in the context of software security, an example of how one could calculate these values in the real world is shown.

4.3 Illustration of a diversity calculation

For an illustration on how to approximately calculate diversities, we look at a real world example. Figure 4.3 shows a host-vulnerability graph of desktop operating systems during 2011. Market share data is taken from [45] and publicly disclosed vulnerability statistics are taken from NVD [48]. We only consider vulnerabilities in two classes of software - operating systems and browsers - because estimated market shares for each are easily available. Every host has to choose one OS and one browser and for simplicity we assume that all three browsers run on all operating systems. Figure 4.4 plots N_a for various values of a.

The first graph $G1$ is computed by assuming that all vulnerabilities from each category, e.g., “Linux”, “Firefox”, are lumped together. This is done to reduce the additional choice hosts would have if there were allowed to pick one vulnerability from “Chrome” and one from “Firefox”. Since this isn’t possible in the real world, and also because it is hard to estimate the total number of computers each software is installed in, as opposed to just market share, we just have 6 vulnerabilities. If n is the number of hosts, there are $2n$ edges. Therefore, vulnerability degrees are

$$\{p_i\} = \{\frac{0.91n}{2n}, \frac{0.03n}{2n}, \frac{0.06n}{2n}, \frac{0.56n}{2n}, \frac{0.23n}{2n}, \frac{0.21n}{2n}\}$$

from which N_a can be computed.

In $G2$, we weight each vulnerability category (e.g., “Windows”), with the number

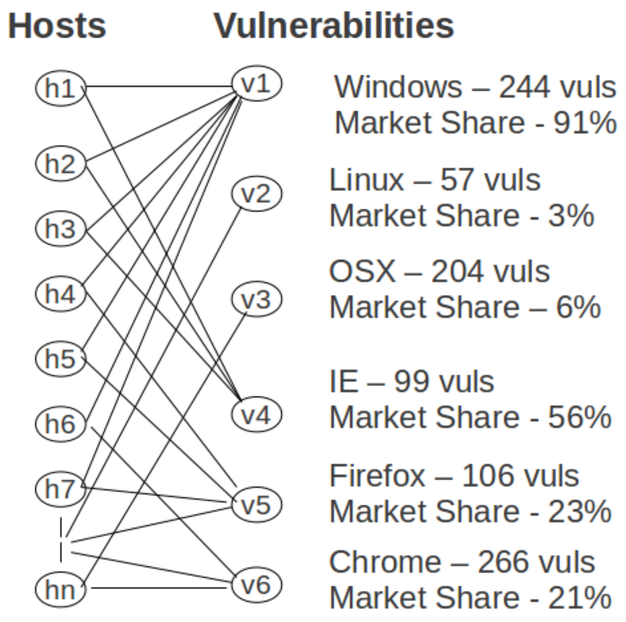


Figure 4.3. Market share data and the number of vulnerabilities reported in NVD for Operating Systems and Browsers in 2011. Every host chooses an OS and a browser. Graph (number of vertices and edges) is not to scale.

of vulnerabilities reported (e.g., 244). In this case different combinations of OS and Browser will lead to different number of vulnerabilities per host. We have

$$\{p_i\} \;=\; \{244 * \frac{0.91n}{tot}, 57 * \frac{0.03n}{tot}, 204 * \frac{0.06n}{tot}, 99 * \frac{0.56n}{tot}, 106 * \frac{0.23n}{tot}, 266 * \frac{0.21n}{tot}\}$$

where tot is the total number of edges. This results in lower entropy because the number of vulnerabilities reported for each category is distributed more evenly than the corresponding market share resulting in increased skewness. $G3$ is the hypothetical case for comparison where all operating systems have the same market share, resulting in the highest diversity. If browsers are also assumed to be equally distributed, we obtain a horizontal straight line passing through N_0.

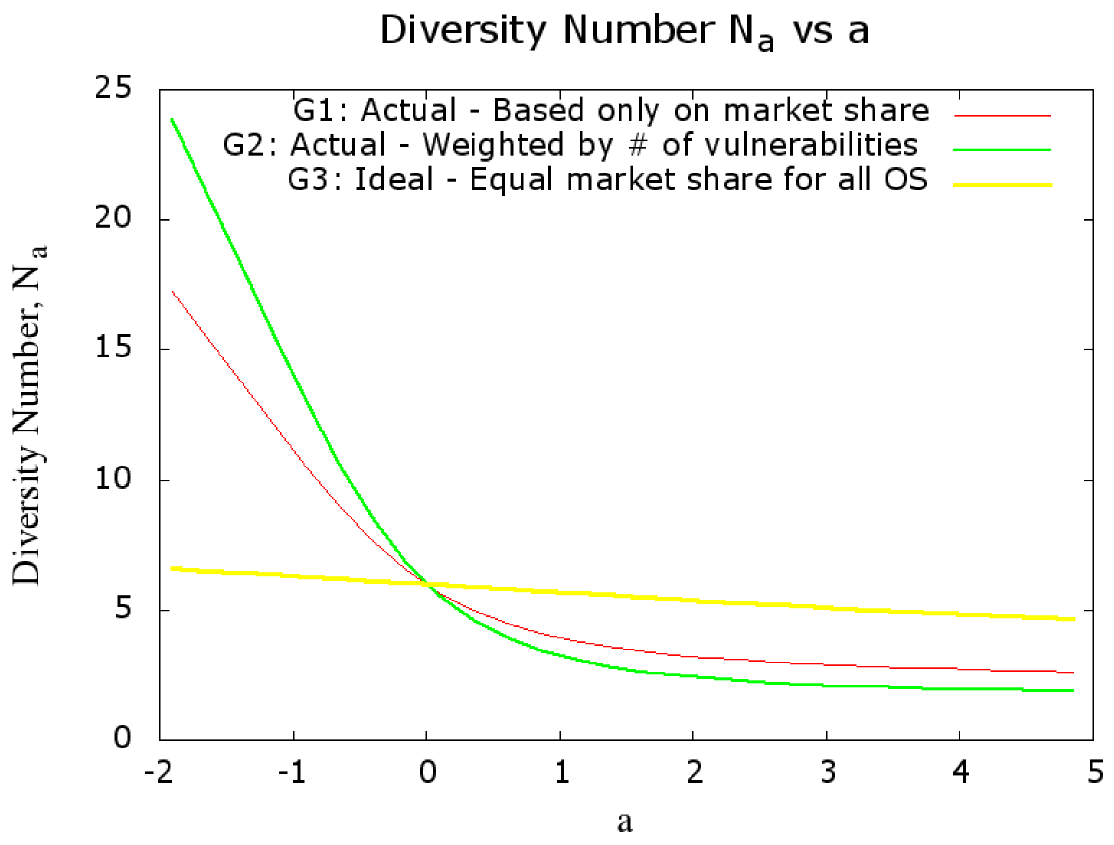


Figure 4.4. Calculating diversity numbers N_a for the software ecosystem shown in Figure 4.3. Note that Renyi entropy $H_a = log(N_a)$ and a special case of that when $a = 1$ is Shannon entropy. $G1$ is obtained by treating all vulnerabilities from each category as just one vulnerability. $G2$ is calculated by weighing each category with the number of vulnerabilities. $G3$ is for comparison where all OS have equal market share.

Which specific measure to use depends on relative weights in the exponent one wishes to use for the least connected or highly connected vulnerabilities. Although our example considers only browser and OS-specific vulnerabilities, it is possible to include a larger variety of software if market share for each is known. An interesting exercise would be to calculate diversity values for Unix-like distributions at the kernel, userspace and application level especially since several variants and forks exist. However, obtaining accurate install base figures for every software is a formidable task. Granularity of choice is a crucial difference in this example from our ideal model and entropy as a diversity measure is highly sensitive to it. In order to increase diversity, we not only have to increase the amount of choice, but also have to make software in components that can be intermixed easily. As a trivial example, suppose SSL libraries *openssl* and *gnutls* are API-level compatible and for all practical purposes can be replaced with each other, then a package manager may decide depending on market share values and security track record which library to install and use.

4.4 Diversity-Stability relationship

Stability of an ecosystem is intricately linked to its diversity. The diversity-stability debate in ecological literature has produced several interesting observations despite still being unresolved [44]. Elton first proposed in the 1950s that simple species population models tend to oscillate and are hence less stable, while more diverse models are more stable as they can withstand perturbations [23]. Later theoretical studies suggested the opposite and it was realized that just as diversity addresses two different components (species richness and relative proportions), the gross word stability can have several more precise meanings including resistance (how much variables under consideration change due to perturbation), resilience (how fast variables return to equilibrium after perturbation) and variability (long term change of variance) [54]. Another concept of interest that is linked to diversity is invasibility—the study of impact of introducing members of foreign species into an ecosystem.

The general consensus of various experiments conducted on plant communities in Minnesota, on the grazing ecosystem in the Serengeti and various replicable microcosm experiments (both aquatic and terrestrial) indicate that there is a positive correlation between stability and diversity [44]. However, there is evidence that diversity is not the prime driver of this relationship but instead ecosystem stability depends on its ability to contain a community of species that are capable of differential response. The impact of harsh environmental conditions on the diversity of an ecosystem remains an active area of research. On one hand, researchers have claimed that the result of physiological or resource stress and strong fluctuations in weather conditions reduce interactions among species favouring coexistence of ecologically very similar species. But to the contrary, Chesson and Huntly, in an important theoretical contribution, have argued that harsh and fluctuating conditions create new ecological niches that lead to increased diversity [15]. Maintenance of diversity requires the two following components: the existence of flux or variability in ecosystems and populations capable of differentially exploiting this flux or variability [44]. We shall return to see how this might apply to computer systems in our discussion on the game theoretic aspects of diversity in Chapter 6.

Several analogies can be drawn between computer and natural ecosystems with respect to stability. Obfuscation [40] and software tamper resistance [76] are static methods to inhibit unintended or malicious modification of software, while dynamic techniques like pH [69] aim to provide resistance and resilience by monitoring system calls and delaying unusual ones. The concept of invasibility carries over to computer systems in a simpler form primarily because of the relatively black and white nature of success or failure of exploit code and root or user privilege separation methods widely used as defaults in access control systems. Literature on the analysis of stability of computer systems includes the study of resistance to attacks when firewalls are misconfigured [81] and of networks to limit propagation of viruses [63], while literature on diversity predominantly includes opinion papers by security researchers [82, 71]. However, the diversity-stability debate itself has been left unexplored for computer systems. In the future, as the major operating systems establish their dominance and their core pieces of code evolve to a relatively stable state, it might be possible to conduct experiments on applications running on higher layers by treating operating systems as a model of the environment.

Although research on the diversity-stability debate in the context of computer systems remains largely unexplored, the striking similarities between ecological subsystems and their software counterparts offer reason to believe that the positive correlations between diversity and stability theorized and observed in ecology would be applicable to computer systems. In fact, several security researchers agree with the hypothesis that increased diversity has the potential to improve security [71, 29, 79]. The more contentious issues are whether current defences are effective in combating monocultures and to what extent is the short-term and the long-term cost of trying to pursue explicit diversification strategies rationally justifiable. Using the framework described so far, we address these issues in the next two chapters.

Chapter 5

Current Defences

5.1 Introduction

Using the model described in Chapter 3 and diversity measures introduced in Chapter 4 we here examine the effect on diversity of six popular defensive techniques: firewalls, signature-based anti-malware defences, heuristic-based anti-malware defences, no-execute memory, address space layout randomization (ASLR), and software updates. We briefly describe each of these defences, analyse their effect on some classes of vulnerabilities, and discuss how much, if at all, they contribute towards making attacks less effective from an ecosystem perspective.

5.2 Firewalls

Firewalls filter or block network traffic destined for specific hosts or services, either by running on routers or bridges (network firewalls), or by running on individual hosts (host-based firewalls).

Firewalls are an extremely scalable defence from a deployment perspective. Network firewalls can protect thousands of hosts by sitting at an ingress point for a shared network connection. Host-based firewalls, when configured with standard rules, can also be deployed on many hosts for very little cost: the software and policies simply have to be propagated.

Firewalls are also very effective at raising the cost of exploiting vulnerabilities in network services that need not be accessible from the open Internet. But the very ability of a network firewall to protect many hosts at once also means that once that firewall is bypassed, it provides no further protection. The cost of having to bypass the network firewall can be amortized over all of the hosts that it protects. Similarly, the cost of developing a technique to bypass a host-based firewall can be amortized over many hosts so protected.

In the host-vulnerability graph, let $V_f \subset V$ be the set of vulnerabilities a that a firewall running on $H_f \subset H$ protects. Then, in the best-case scenario where an attacker cannot exploit any vulnerability from V_f, a fraction of $|V_f||H_f|/kn$ edges disappear from the graph. In the worst case where an attacker has managed to bypass the firewall all those edges remain and the graph is unchanged.

Firewalls thus act as a coarse grained control system over what data is transferred to and from a network. To an attacker this translates to a one-time cost for morphing the exploit code into data which a firewall allows. After this transformation, he can potentially access hundreds or thousands of hosts.

5.3 Anti-Malware Signatures

Signature-based anti-malware systems, whether implemented as network traffic, data (email/document), or host-based scanners, all search for specific patterns of bytes in order to detect known pieces of malware. While scanners may employ a number of techniques, such as unpackers and machine emulators, to circumvent malware obfuscation, these are all used to expose behaviour for signature matching.

Signature-based anti-malware systems can be deployed relatively easily: after installations, automated update mechanisms keep their signatures updated. The per-host cost of deployment is thus quite minimal: it is software that is installed and then largely forgotten.

Anti-malware systems can incur a cost to the defender, however, every time they detect something that is suspected to be malicious. This is in contrast to firewalls, where potentially malicious network connections are silently blocked. Vendors, however, devote significant resources to tuning their signatures to ensure that these costs, particularly those caused by false alarms, are sufficiently low to preserve their deployment scalability.

As it turns out, the true role of these systems is actually to limit attack scalability, rather than detect attacks, per se. Attackers have access to most anti-malware packages; thus, they can design their malware so that they will not be detected. Once a piece of malware becomes widely enough dispersed, though, it will be captured for analysis by an anti-malware vendor who will then craft and distribute signatures to detect it. Thus, the only attacks that signature-based anti-malware systems detect reliably are the ones that have been widely deployed. If an attacker is willing to forego attacking many targets that could have been attacked, or if attacks can be mounted in a sufficiently covert way, it is possible for no signature to be created for their malware for an indefinite period of time.

Anti-malware signatures can be viewed as having a “recycling” effect on the vulnerability set. As new vulnerabilities are discovered and added to V, anti-malware signatures ensure that old well known vulnerabilities are deleted from V. This does open up the possibility for hosts to choose different vulnerabilities and redistribute their edges, but only if we have defences that enable intelligent redistribution. If we follow conventional ways, the steady state assumption holds and the graph remains unaltered.

Signature-based defences can be largely circumvented through the use of polymorphism engines. By creating a variety of obfuscated binaries, the anti-malware vendor has to devote resources to crafting signatures that can recognize each variant. With effort, vendors can develop signatures for the parts of the malware that do not change; unfortunately, this arms race does not favour the defenders [70].

5.4 Anti-Malware Heuristics

Heuristic-based anti-malware defences attempt to block malware by detecting known patterns of behaviour that have been found to correlate with malicious activity. However the behaviour pattern is generated, these systems can notice suspicious behaviour—relaying keystrokes to the network, oddly-timed registry edits—and can then quarantine the responsible piece of code.

These systems are more comparable to firewall-type defences than signature-based anti-malware defences. They do not block specific pieces of malware, and they do not eliminate vulnerabilities; instead, they generally make it more difficult for an attacker to exploit a set of vulnerabilities.

Heuristic-based defences have similar per-host incremental costs to signature-based anti-malware defences; indeed, in practice the two are generally combined. However, heuristic-based defences are more like firewalls than signature-based defences. Unless heuristics are collected and adapted on a per host basis they present a relatively static challenge to attackers. An exploit that is carefully created to evade detection by heuristics on one machine works on all the machines using those heuristics.

5.5 No-execute memory

No-execute memory, as for example implemented in the DEP protection on Windows and segment-based no-execute stacks on Linux [19], interferes with stack-based buffer overflow attacks and, depending on implementation, other code injection attacks, by preventing code that has been injected from being executed. Because no-execute memory defences normally have to be incorporated into operating system kernels and operate with no user intervention and very few false alarms, they incrementally cost almost nothing to deploy.

When first developed, no-execute memory made certain types of buffer overflow attacks infeasible—they seemed to effectively remove an entire class of vulnerabilities from the attacker's repertoire. Unfortunately, those vulnerabilities were not eliminated; they just needed to be exploited by a new attack technique. Return-to-libc [53] and then return-oriented programming [13] made these vulnerabilities accessible again, albeit at a higher one-time cost to the attacker.

In the host-vulnerability graph, no-execute memory, in the best case, deletes vulnerabilities from V, and in the worst case adds a one-time cost to the creation of exploits.

5.6 ASLR

Address-space layout randomization (ASLR) randomly permutes the layout of a process's address space in such a way so as to defeat code injection attacks that have rigid dependencies on there being a predictable layout to the target process's address space. Like no-execute memory, ASLR provides significant protection against simple buffer overflow attacks; also, like no-execute memory, it can be circumvented, in this case through the use of a “derandomization” attack (i.e., by brute forcing the possible variations in a process’s layout) [64]. Circumvention is only feasible if the attacker is allowed to make many tries (i.e., they can crash a program thousands of times), and if the target does not randomize enough. On 64-bit hosts, ASLR can potentially offer enough entropy to keep hosts secure for years, even in the face of a derandomization attack.

ASLR, like no-execute memory, can be deployed for almost no incremental cost per host because it is a feature incorporated into the operating system. Also, like most of the already discussed defences, ASLR does not have an impact on attack scalability. If the implementation provides sufficient entropy or other protections to prevent brute-force attacks, certain types of attacks are simply blocked. Yet, once it is circumvented on one host, all hosts are now equally vulnerable.

In the host-vulnerability graph, ASLR used on a subset of hosts (like 64-bit ones) simply disallows those hosts from choosing certain classes of vulnerabilities. As the number of hosts on which ASLR is an effective defence increases, the corresponding buffer overflow attacks it prevents against vanish from the vulnerability set. Thus, in the best case, ASLR changes the graph by removing once and for all some vulnerabilities from the vulnerability set, while in the worst case, leaves the graph intact.

5.7 Software Updates

With a software update, code is replaced or patched in such a way that the updated program (ideally) no longer has a vulnerability that the previous version had. If we assume that most systems have many vulnerabilities, the removal of one vulnerability does not significantly change the security situation of the host: there still exist many vulnerabilities that can be used to compromise the updated host. Nevertheless, if the vulnerability is being actively exploited, eliminating that vulnerability can offer significant practical protection.

Software updates used to be very expensive: they required downtime and manual time and effort to apply. Updates now are increasingly automated, with patches being propagated and often even installed in the background. These changes have greatly reduced the incremental cost of deploying software updates: once the infrastructure is in place, each new update can incur almost no cost per host. However, it is still expensive to develop updates so that they are safe to deploy in an automated fashion. And, not everyone can or does automatically install software updates.

Software updates have a rather paradoxical effect on attack scalability. In general they serve to limit scalability by enabling defenders to quickly eliminate vulnerabilities that are being actively targeted. Thus, any exploit that is sufficiently widespread will quickly become much less effective as patches are developed and distributed. However, the price of this is the creation of infrastructure that can quickly change the code running on millions of hosts. If compromised, software update mechanisms can become one of the most efficient ways of propagating an attack. Thus software updates serve to limit attack scalability, except when they greatly enhance it.

On the host-vulnerability graph, software updates can been seen as having the same recycling effect as Anti-malware signatures. They remove old vulnerabilities from V and potentially introduce new ones. But unlike in the case of anti-malware signatures, the automatic software update system does not provide an opportunity for hosts to redistribute their edges. The single point of failure problem is discussed in Section 7.3.1 and a simple potential solution to lack of opportunistic diversification is proposed in Section 7.3.2.

5.8 Conclusion

We've looked at six popular defensive techniques and their effect on reducing the scalability of attacks. Several independent parameters are needed to describe these defences in full detail, greatly increasing the complexity of analysis. Hence, we compromise accuracy for simplicity, and flatten the multi-dimensional space that characterizes the wide range of exploits in into one single vulnerability set. The effect of these defences on the host-vulnerability graph can be classified into the following three categories:

- Remove a subset of vulnerabilities from V; e.g., ASLR, No-execute memory.
- Remove a fraction of host-vulnerability edges; e.g., Firewalls, Anti-malware heuristics.
- Recycle vulnerabilities in V; e.g., Anti-malware signatures, Software updates.

None of these defences appear to cause any significant changes to the host-vulnerability edge redistribution and consequently have no effect on the diversity indices defined in Section 4.2. If we believe that diversity can indeed be an effective defensive strategy, we need to design new kinds of defences that not only protect against individual attacks but also reduce the scalability of attacks.

Chapter 6

Games for diversity

Increasingly, game theory is proving to be a useful tool in understanding strategies of attackers and defenders and outcomes of the games they're playing. In this chapter we discuss games related to diversity. We provide a brief overview of game theory, its scope and validity in explaining and predicting agent behaviour and review computer science literature where game theory has been successfully employed. We then study anti-coordination games and argue that they capture an important aspect of choice in diversity. Analysis of equilibrium conditions reveals why rational agents might not always end up choosing the most diverse outcomes. Finally, we propose a small modification to a popular cyber security game, capture the flag, that would enable us to collect interesting real world data about choices made in security-focused situations.

6.1 A brief overview of Game theory

In this section we introduce the reader to basic concepts of game theory that we'll use in the rest of this chapter. For a detailed overview on this topic, the reader may refer to any of the several excellent books [50, 5, 55]. A reader familiar with game theory may skip this section.

6.1.1 History and Applications

Origins of formal game theory can be traced back to 1944 when John von Neumann and Oskar Morgenstern published *Theory of Games and Economic Behavior* [75]. Their book laid the foundations of utility theory for consistently associating payoffs with strategies and proved the minimax theorem which guarantees that every two person zero-sum game with finitely many strategies has an equilibrium value. Since then, game theory has been generalized in several different directions and applied in a variety of fields.

The concept of Nash equilibrium was introduced for non-zero sum games and it was shown that even non-cooperative games have an equilibrium value if probability distributions over pure strategies were allowed. The field of economics was revolutionized by game theory because of its ability to model competition, markets, interacting agents, bargaining and auctions. Design of a good auction system for the sale of British 3G Telecom licenses brought in a revenue of $34 billion which was several times more than what the finance ministry expected [6]. But other European countries with different auction designs including Italy and Switzerland fared far worse despite having the same value of licenses sold. Per capita revenue ranged from 650 Euros in the UK to 20 Euros in Switzerland indicating that good auction design can have a huge financial impact [36]. Evolutionary game theory was developed to explain several aspects of living systems including animal cooperation, sexual behaviour and biological altruism with surprisingly good results [66]. Game theory has also proved its use in seemingly unrelated fields like topology and foundations of set theory [42].

6.1.2 Game description

The following concepts are essential to all games :

- **Players:** Individuals who make decisions to maximize their utility by choice of actions. A game may contain one or several players which could be humans, algorithms pitted against each other or just hypothetical agents. In all cases it is assumed that all players are rational in that they always try to maximize their payoffs.

- **Actions:** Every player P_i has a set of actions $A_i = \{a_j\}$ she is allowed to make. The set of actions available to each player may or may not be the same. An action profile $a = \{a_i\}, i = 1, 2..., n$ is a list of actions one for each player. a_{-i} denotes the joint action taken by other players except P_i.

- **Strategies:** In a certain situation, a player may either select one action deterministically or toss a coin and randomly choose one among all possible actions available to her. A strategy s_i is a probability distribution over a player's action set. If the strategy picks out a single action deterministically then s_i is called a pure strategy and otherwise it is called a mixed strategy.

- **Payoff:** Every player has an associated payoff function $U_i(s_1, s_2, ..., s_n)$ where player P_i chooses s_i and the remaining choose other strategies. Just as in the case of actions, this can be represented succinctly as $U_i(s_i, s_{-i})$

Games may involve each player making just one move each, or could go on for several iterations. Outcomes in both these cases may be different. The **order of play** might also be important if the game models a situation where players cannot all move simultaneously. In these cases, players that move in subsequent turns might have the advantage of knowing the moves of previous players. This is typically modelled using **information sets**.

In some real world cases it might be reasonable to assume the existence of a method where players can make binding commitments. Games where such a mechanism exists are called **cooperative games** as opposed to **non-cooperative games** where players cannot make binding commitments. If they choose to cooperate, it would have to be because cooperation increases payoffs of one or both players. Games where the sum of payoffs of all players is always 0 are called **zero-sum games** as opposed to **non-zero sum games** where the sum of total payoffs may not be zero.

6.1.3 Solution concepts

Given a complete description of a game, game theorists seek to find a way to predict how a game will be played out. A solution describes which strategies are adopted by each player and what the end result of the game is expected to be. Sometimes games can have multiple solutions making it necessary to refine solution concepts until implausible scenarios are eliminated and the desired equilibrium is reached. Typically, game theorists are interested in finding actions and payoffs that occur at equilibrium conditions.

A strategy that maximizes a player's minimum expected payoff is called the **maximin strategy**. Formally, a maximin value is the following :

$$\mu_i^m = \max_{s_i} \min_{a_{-i} \in A_{-i}} U_i(s_i, a_{-i}) \tag{6.1}$$

In the case of two player zero-sum games with finitely many strategies, minimax theorem assures us that the maximin value (maximizing the minimum expected payoff) of one player is same as the minimax value (minimize the maximum expected loss) of the other player. However, in many games players can often do better than their maximin strategies by outsmarting other players or by cooperating with them.

Nash equilibrium is generally considered to be the single most important solution concept in game theory. A game is said to be in a Nash equilibrium if no player has an incentive to unilaterally deviate from his current strategy. More formally, a profile strategy s^* is a Nash equilibrium if $\forall i$,

$$U_i(s_i^*, s_{-i}^*) \geq U_i(s_i^{'}, s_{-i}^*), \forall s_i^{'} \tag{6.2}$$

It might be the case that if two or more players could change their strategies, every individual's payoff would increase. A strategy profile is said to be **pareto optimum** if there exists no other strategy profile that results in a higher payoff for every player. One of the more striking predictions of game theory is famously observed in a popular two player game, the Prisoner's Dilemma. In this game, both players can get a higher payoff if they cooperate, but Nash equilibrium occurs when one of the players does not cooperate leading to a predicted outcome that isn't pareto optimal. In real world experiments and game shows based on Prisoner's Dilemma, people often behave in a way predicted by Nash equilibrium [41]. But sometimes unexpected cooperation among players is also observed, especially in repeated versions of the game [38].

6.1.4 Interpretation

In game theory, results on existence of various solution concepts are based on mathematically sound fixed point theorems. John Nash used Kakutani's fixed point theorem to show that every non-zero sum game with finitely many strategies has at least a mixed strategy Nash equilibrium. While theoretically, a mixed strategy is just a probability distribution over pure strategies, in application it is still open to interpretation.

One major issue it raises is that in a two player game with completely determined payoffs and no pure strategy nash equilibrium, in what sense is it *rational* to probabilistically choose one strategy sometimes and another at other times. Mixed strategy Nash equilibrium can be interpreted in the following ways [59]:

1. **Purification idea.** According to this interpretation players have motives not captured by the model of the game that dictate which actions to pick when choosing strategies. This interpretation is used with some caution because it implies either that the model is incomplete or that the player is acting irrationally by basing her preference over outcomes on extraneous reasons that have nothing to do with payoffs. However, it does provide a consistent interpretation.

2. **Large population interpretation.** In this interpretation, each agent playing the game is a proxy for a large set of agents and if the game is repeated independently several times the mixed strategy is a probability distribution over all the players in the game and not over a single player's pure strategies. Thus, if a game has a mixed strategy Nash equilibrium and is played by several individuals in a population, some individuals will be worse off than others.

6.2 Game theory and computer science

Both computer science and game theory have found many useful applications in a large number of other fields. In fact their spread is now so wide that the intersection of both these fields has led to the creation of a new emerging field known as Algorithmic game theory [47].

6.2.1 Computation in games

Nash's result that every finite game has a mixed strategy Nash equilibrium is existential. The problem of how an agent can go about computing such an equilibrium and whether there exist efficient algorithms for it can be answered by computational complexity analysis. Computing Nash equilibrium does not fall into the most common class of problems regarded as intractable, the NP-complete class, because unlike *SAT*, *NASH* always has a solution. However, recent results have established that computing Nash equilibrium is indeed a hard problem; *NASH* is PPAD-complete [47, p. 29]. The most general algorithm for computing equilibrium for two players games, the Lemke-Howson algorithm, is exponential in the worst case [47, p. 33].

This discovery is of importance to games in diversity because our hope is not just to provide a descriptive theory, but also to investigate mechanisms that can improve diversity. It is important that distributed algorithms running on individual machines and negotiating software components in vulnerability markets are efficient in estimating equilibrium values. There are positive results that indicate that approximate Nash equilibrium might be computed in polynomial time.

Another area where algorithms are useful in game theory is in computing market equilibria using combinatorial and convex programming techniques. Researchers are also trying to relate the Multi-Party Communication problem in cryptography to the problem of computing equilibria in strategic games.

6.2.2 Games in computation

Game theory is a useful tool in areas of computer systems where agents operate under limited resources to maximize their output. For example, in non-cooperative routing games in which self-interested agents route traffic through a congested network, it is important to estimate the inefficiency at equilibrium as compared to the case where all traffic is routed in a centrally planned way. This is commonly referred to as the cost of anarchy [47, p. 461]. Similarly, game theory helps in understanding the formation of large computer networks by autonomous competing entities. Load balancing is another area gaining prominence due to increasing number of data centres and large Internet companies requiring scalable infrastructure.

Incentive modelling using game theory has proved to be a very important factor in deciding the success or failure of a multi-agent system. Improving the resilience of peer-to-peer systems by appropriately designing incentives to limit damage from malicious nodes has been a fruitful area of research [47, p. 593]. Game theory is also used in designing manipulation-resistant online reputation systems to prevent sybil attacks and by search engines in online advertising to optimize profits by auctioning parts of screen space.

6.2.3 Games in computer security

Misaligned incentives was one of the first problems related to game theory identified in computer security. If individuals responsible for protecting systems don't face full consequences of a security breach, they may make socially suboptimal choices. Economics of information security is an emerging sub-field whose import to security might be comparable to that of cryptography [47, p. 633]. Hal Varian analyses the problem of free riding where several agents responsible for reliable functioning of a system would prefer to shirk than to contribute resulting in a suboptimal social outcome [74].

Bootstrapping new security protocols is also conducive to game theoretic analysis. If the cost of technology is greater than its benefit until a critical threshold of adopters is reached, the improved technology might never get deployed. The essence of this problem is captured in coordination games where it is desirable to enforce coordination, but letting individuals independently choose strategies results in suboptimal outcomes. In the case of diversity, we model one of its aspects using anti-coordination games.

Our inability to estimate security of software has direct consequences for producing poor quality software. When information asymmetry exists where consumers cannot assess the value of the software, there is an incentive for sellers to distribute low quality software creating a *market for lemons* [47, p. 638]. Games for analysis of computer network security is an active area of research with several models of attackers, network administrators and hosts modelled as players in many kinds of games. Quantitative risk assessment, interactions between DDoS attackers and network administrators and intrusion detection problems in mobile ad-hoc networks have been analysed as games with imperfect information [58].

The recently proposed FlipIt game aims to analyse situations where it is unclear if the defender or the attacker is in control of a host [73]. The identity of who is in control of the host is not revealed until a player makes a move. Costs are associated with moving and the objective of a player is to maximize the total time for which she has the control of the host. Analysis reveals that aggressive play can motivate opponents to drop out and close monitoring of resources is essential to detecting attacks faster. A primary suggestion offered in the paper is that systems must be designed for repeated total compromise.

We are unaware of any work that tries to analyse diversity from a game theoretic point of view. As mentioned in Section 4.2, diversity consists of two components: the absolute number of varieties available and the relative distribution of varieties. The popular maxim, *“Don’t put all your eggs in one basket”* raises two important questions:

- How many baskets are needed?
- How should eggs be distributed in those baskets?

In terms of security, we are interested in investigating how many alternatives of software are sufficient and how a host goes about choosing one among those alternatives. The first part of this question depends on several factors and is quite complex to analyse. We discuss a few approaches in Section 7.3.1. In the rest of this chapter we study the second half of this question.

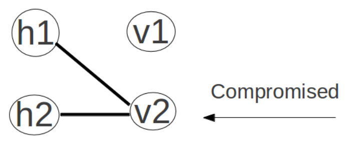
6.3 Anti-coordination games for diversity

In this section we model an essential aspect of choice in diversity from a computer security standpoint as an anti-coordination game. Anti coordination games [8] are two player, two strategy games where it is preferred that both players choose different strategies. Using these games we can investigate what would happen when new software alternatives become available enabling some hosts to make a switch. Consider two hosts h_1, h_2 and two vulnerabilities v_1, v_2, such that both h_1 and h_2 are initially connected to v_2 as shown in Figure 6.1.

To start the analysis, one can think of an event when either a new software containing v_1 is released in the market or that software containing v_2 is compromised prompting hosts to consider other alternatives. Each host now has two choices: to stay with v_2 or to switch to v_1. Since switching involves potentially installing new software, getting it to interoperate and getting a user familiar with it, we associated a cost, c_w with it. But if both players continue to use v_2, they risk another attack. So there is a cost of sharing a vulnerability, c_2 which is over and above the intrinsic cost of having just one vulnerability c_0. Since both hosts are required to select one vulnerability in all cases, it is assumed without loss of generality that $c_0 = 0$. To summarize:

- c_2 = cost of choosing a vulnerability if both hosts choose the same vulnerability.
- c_w = cost of switching to a different vulnerability

The corresponding payoff matrix for the game we described above is shown in Figure 6.1. The entries in the matrix are utilities and hence are negatives of the cost variables.



Payoffs	Stay (v_2)	Switch (v_1)
Stay (v_2)	$-c_2, -c_2$	$0, -c_w$
Switch (v_1)	$-c_w, 0$	$-(c_2 + c_w, -c_2 + c_w)$

Figure 6.1. Two player anti-coordination game with switching costs

We have the following two cases to consider:

Case 1: $c_w > c_2$

In this case it costs more to switch than to continue using the older software that is already used by the other host. An example of this game with numerical payoffs is shown in Table 6.1a. Suppose the column player chooses to stay. Then, the best response for the row player is also to stay because switching costs him 15, while staying only 10. Suppose the column player chooses to switch. Even then the best response for the row player is to stay because it would cost him nothing. Therefore unconditionally, the row player would continue to stay, and symmetrically by the same argument so will the column player. This game has only one Nash equilibrium, $\{stay, stay\}$, and the total social cost is $2c_2$ which is suboptimal if $2c_2 > c_w$. In our example a $\{stay, switch\}$ or $\{switch, stay\}$ outcome would cost a total of only 15, while the predicted outcome $\{stay, stay\}$ would cost 20. This game has no mixed strategy Nash equilibrium because for both players *stay* strongly dominates *switch.*

Case 2: $c_w < c_2$

If one were to assume that the software community managed to make inter-operable variants extremely easy to replace, then it might be that the cost of switching to a new variety is less than the expected risk of continuing to share the same variety with other hosts. An example of a game with payoffs where $c_w = 10$ and $c_2 = 15$ is shown in Table 6.1b. To analyse this, suppose the column player chooses to stay. Then the best response for the row player is to switch because it would cost him 10 and not 15. If the column player chooses to switch, the row player would prefer to stay. The analysis is similar for the column player. This game has two pure strategy Nash equilibria $\{stay, switch\}$ and $\{switch, stay\}$.

While these outcomes are socially optimal, the problem is how do hosts arrive at this equilibrium. Each host could wait for the other to make a switch, because it wouldn't have to bear the cost of switching but can reap the reward of not having to share the software containing the vulnerability. Mixed strategy Nash equilibrium is typically used as a solution concept when no Pure strategy Nash equilibrium exist. However, in this case if we calculate the probabilities, we find that with a probability of $(c_w + c_2)/2c_2$ both hosts would choose to stay and the total cost for both players is then $2(c_2 - c_w)c_w/c_2$. If $c_w > c_2/2$, then the mixed strategy Nash equilibrium is a better social outcome than either of the pure strategies. If one subscribes to the large population interpretation of mixed strategy, then in our example, $\{stay, stay\}$ is the outcome of the game 69% of the time.

Payoffs	Stay	Switch
Stay	$-10, -10$	$0, -15$
Switch	$-15, 0$	$-25, -25$

(a) Cost of switching is more than cost of sharing. $c_w > c_2$

Payoffs	Stay	Switch
Stay	$-15, -15$	$0, -10$
Switch	$-10, 0$	$-25, -25$

(b) Cost of switching is less than cost of sharing. $c_w < c_2$

Table 6.1. Two player anti-coordination games with numerical payoffs

6.4 Dispersion games for diversity

This section extends Section 6.3 by considering choice of several vulnerabilities among several hosts. Two player anti-coordination games generalized to many players and actions are called dispersion games [30]. In essence, these are games where utilities are such that given a choice of strategies for every agent, maximally dispersed outcomes are preferred. More formally, a normal form game G is a tuple $< H, A, \succeq_i >$, where H is the set of agents, A is the set of actions available and $\succeq_i$ is the preference relation over outcomes $O = A^n$ for agent h_i.

Our model described in Section 3.2 has a list of assumptions associated with it. Some of these assumptions correspond to symmetries of the game.

- **Common action:** If every agent has the same set of actions, $A_i = A$ for all $h_i \in H$, then we have a common action (CA) game. In our model, any host can pick any vulnerability and hence we consider only CA games.

- **Agent symmetry:** A CA game is agent symmetric if for all outcomes $o =< a_1, a_2, ..., a_n >$ and for all its permutations $o^{'}$, $o \succeq_i o^{'}$ and $o^{'} \succeq_i o$ for all hosts $h_i \in H$. This means that all hosts have the same preference relation over outcomes irrespective of which host choose which action. This is consistent with our *Computer equivalence* assumption where we treat all hosts equally.

- **Action symmetry:** A CA game is action symmetric iff for all outcomes o and $o^{'}$, if there exists a one-to-one mapping $f : A \rightarrow A$ such that for all $h_i \in H$, $f(a_i) = a_i^{'}$, then $o \succeq_i o^{'}$ and $o^{'} \succeq_i o$. This means no host inherently prefers one action over the other. This definition is required because hosts may have an action preference depending on how many other hosts choose those actions. In our model, *Vulnerability criticality* assumption translates into Action symmetry as it assigns equal apriori weight to all vulnerabilities.

In a fully symmetric game, agents cannot distinguish between outcomes with the same configuration of number of agents choosing actions. In a diversity calculation this is precisely what matters. To extend the problem of two player anti-coordination game to many players, we study a specific game where a maximally dispersed outcome is preferred but it costs to switch strategies that lead to it. Figure 6.2 shows n hosts, $m << n$ vulnerabilities where initially all hosts are connected to just one vulnerability v_c. The question we want to ask is: suppose v_c is compromised and it costs c_w to switch to a different vulnerability, what will the final graph look like? How many hosts switch and how many remain on v_c?

To answer these questions, we first have to introduce a scalability factor. If a host chooses a vulnerability v such that $deg(v) = x$, then the utility due to interoperability and ease of creating and sharing software might grow as $benefit(x)$ while the risk associated with increased number of hosts choosing the same software leading to higher probability of attacks might be $risk(x)$. Since the risks and the benefits of sharing vulnerabilities can depend on several factors that are hard to theoretically determine, we combine the two and treat the utility of sharing vulnerabilities as a single function. We define the scalability cost as a monotonic function $\Pi(x) = risk(x) - benefit(x)$. Let c_0 be the intrinsic cost of choosing a lone vulnerability, v such that $deg(v) = 1$. Then $c_0\Pi(x)$ is the cost of choosing v such that $deg(v) = x$.

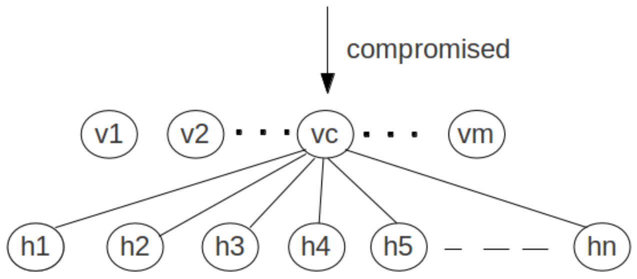


Figure 6.2. A graph of n hosts, $m << n$ vulnerabilities with all hosts connected to v_c initially. Cost of switching is c_w, while cost of choosing a vulnerability v_i is $c_0\Pi(deg(v_i))$ where $\Pi(x)$ is the risk-payoff scalability factor.

As before, let c_w = cost of switching from one vulnerability to another and $o =< a_1, a_2, ..., a_n >$ be an outcome of this game where $a_i \in V$ is the action taken by host h_i. Further, let $n^o_{a_i}$ = number of hosts choosing a_i in o. The utility of a host does not depend on the specific hosts choosing a_i, but only on the number of hosts that have selected it. Therefore, utility for a host choosing a_i in an outcome o is

$$U_{a_i}(o) = -c_0\Pi(n^o_{a_i}) - c_w(1 - \delta_{a_i a_c}) \tag{6.3}$$

where δ_{ij} is the Kronecker delta function (i.e. $\delta_{ij} = 1$ iff $i = j$ and 0 otherwise). If a host chooses a_j over a_k, then we must have $U_{a_j}(o) \geq U_{a_k}(o)$, i.e.

$$c_0(\Pi(n^o_{a_k}) - \Pi(n^o_{a_j})) + c_w(\delta_{a_j a_c} - \delta_{a_k a_c}) \geq 0 \tag{6.4}$$

If $j \neq c$ and $k \neq c$, this reduces to $\Pi(n^o_{a_k}) \geq \Pi(n^o_{a_j})$, and symmetrically, a host that chooses a_k over a_j must have $\Pi(n^o_{a_k}) \leq \Pi(n^o_{a_j})$. By monotonicity of $\Pi(x)$ we have $n^o_{a_j} = n^o_{a_k}$, i.e all $v_i \neq v_c$ will have equal preference as is to be expected.

If $j = c$ and $k \neq c$, we get $\Pi(n^o_{a_c}) \leq \Pi(n^o_{a_k}) + c_w/c_0$, and symmetrically, for $j \neq c$ and $k = c$ we get $\Pi(n^o_{a_c}) \geq \Pi(n^o_{a_k}) + c_w/c_0$.

Therefore, if n_c hosts continue to choose v_c and $(m-1)n_s$ hosts make a switch from v_c to $v_i \neq v_c$, then, observing that $n_c + (m-1)n_s = n$, we have

$$\Pi(n_s) = \Pi(n - (m-1)n_s) - c_w/c_0 \tag{6.5}$$

We, of course, do not know what the scalability factor $\Pi(x)$ is, but if we did know that and if we could estimate c_w/c_0, we could calculate how many hosts would choose to stay with v_c and how many would shift to other vulnerabilities using Equation 6.5. More interestingly, in the other direction, if we had a series of values for n, m, n_s and c_w/c_0, we could try to estimate $\Pi(x)$. In the next section we propose a modification to the popular Internet security game Capture the Flag that will make it possible to collect this kind of data in the real world.

6.5 Capture the Diversity

Diversity is a very broad concept and although one can make several assumptions and perform analysis, it is often hard to obtain data for corroboration. In the real world, choices people make about software aren't solely dictated by security. Hence we turn to popular security competitions like Capture the Flag where the incentives are primarily to secure one's system, keep services running and attack other systems [17]. Typically a team is provided with m vulnerable services, with $m \approx 10$. A Game server periodically checks every host for running services which yield points for the host. If an attacker creates an exploit for a vulnerability in a service, periodic attacks are launched on hosts running that service to capture its flag and report it to the game server.

The crucial point is that participants don't get to choose which services to run—they have to keep all m services running. In the real world, however, if a service is easily exploited, one gets to replace it with an alternative. To model this, we propose that $2m$ services be available to the participant, but only one from each of m categories of services needs to be kept running simultaneously. Everyone gets to see the host-service graph (available from the game server) and choose which of the two alternative services to run, e.g., *postfix* or *exim*; *mysql* or *postgresql* and so on.

We hypothesize that this simple modification will yield very interesting data. First, it will show the evolution of host-vulnerability graph and entropy H_a with time. Second, we get to see the strategies employed by top teams that defend and attack well. Third, it is entirely possible that some teams might score high points primarily by strategically switching one service for the other based on the game-play. Emphasizing this last point is one of the primary goals of this thesis. Although finding and fixing bugs is important, as the security and hacking community are well aware, choosing which software to run given ample choice is non-trivial and highly relevant to security.

6.6 Summary

In this chapter we've introduced game theory, briefly reviewed past work on applications of game theory in computer science and vice versa. We've noted several areas where both these fields complement each other including computer security.

Applications of game theory in security is still in its infancy, but as the complexity of the system increases and more agents begin to partake in its upkeep, security breaches have to be treated as an integral part of our model and not as extraneous events. We looked at a game, *FlipIt*, in which repeated compromise of systems is treated as the norm. We also briefly touched upon applications of game theory in understanding misaligned incentives, the problem of free riding and implications of incomplete information on software quality.

We then argued that anti-coordination games capture an essential aspect of diversity. Using our simplified model of hosts and vulnerabilities, we studied an anti-coordination game where players (hosts) have motives to choose different actions (vulnerabilities) but where it costs to make a move. In the case where it is more expensive to make a move, we note that the equilibrium condition can lead to a socially suboptimal outcome, while in the case where it is less expensive to make a move, pure strategy Nash equilibria are optimal. However, there is also a mixed strategy Nash equilibrium that might have a greater social payoff depending on cost ratios which also causes a reduction in diversity.

In generalizing our proposed version of anti-coordination games to several players, we observed that assumptions made in our model translate into symmetries of dispersion games. A specific game was analysed by assuming an unknown payoff multiplier function, and its solution at the equilibrium condition led us to an equation that could let us estimate the payoff function if other parameters were known. At this point we proposed a modification to the popular security game *Capture the flag* to include choice between services and argued that this would enable us to capture data for estimating revealed user preference for diversity in a security context.

Chapter 7

Discussion

This chapter is divided into four sections. In the first section, we discuss the main contributions of this thesis to the analysis of software diversity in the context of security. The second section examines shortcomings in our model, assumptions and analysis and also talks about general limitations of a formal theory of diversity. We suggest both theoretical and practical avenues of future work in the next section. The final section ends with a few concluding remarks.

7.1 Contributions

One of the first problems we faced was developing a software ecosystem model simple enough to admit rigorous analysis but extensible enough to be applicable in the real world. By treating software as a set of vulnerabilities and hosts as agents that choose a subset of those vulnerabilities, our model of a software ecosystem presented in Section 3.2 appeared to capture the essential aspect of scalability of attacks. In Section 3.3, we listed several simplifying assumptions under the lenses of which our model applied to the real world, while also providing justifications for some of these using real world examples.

Even without diversity measures defined on the model, we were able to investigate some problems faced by attackers and defenders. In Section 3.4 we showed a simplified formalization of the vulnerability discovery problem and reduced it to the well known paging problem. We also studied the vulnerability coordination problem to learn that even the simplest of cases can require complex algorithms to protect users against malicious hosts.

In Section 4.2, we develop diversity measures for software using a generalized notion of entropy called Renyi entropy. We show a natural derivation of the formula used for entropy and discuss some of its intricacies. We specifically noted that diversity is an amalgamation of two separate components and the ratio in which they're combined is partly responsible for a lack of clarity about the concept. In Section 4.3 we use real world market share and vulnerability data to illustrate how one could approximately calculate diversities. Finally, in Section 4.4, we review ecological literature on diversity-stability relationships and note that despite being hard to quantify, both experiments and theory reveal a positive correlation between the two. We draw some comparisons between natural and software ecosystems and note that a positive correlation between diversity and stability of software systems is similarly plausible although this would require further investigation.

In Chapter 5 we analyse six popular current defences for their effect on diversity. The effects of these defences on the host-vulnerability graph divides them into three classes, none of which alter the vulnerability distribution in the steady state. We argue that for explicit diversity-type defences to be effective, they need to be able redistribute edges in the host-vulnerability graph.

Even if maximizing diversity might be an ideal goal for an ecosystem designer, our software ecosystem composed of several entities with varying incentives, motivation and payoffs rarely have a central design. If we treat hosts choosing software as agents acting selfishly, then it is unclear if the resulting ecosystem has optimal diversity even if each host would prefer an outcome that has optimal diversity. In Section 6.3, we propose that anti-coordination games with moving costs capture an essential aspect of diversity. In both cases where moving costs are less than or more than sharing costs, we notice that selfish actions can result in socially suboptimal outcomes. Section 6.4 looks at dispersion games, a generalization of anti-coordination games to several players and strategies. One of the major difficulties in game theory is coming up with utility functions. We assume a certain risk-payoff scalability function and come up with an equation involving that function at the equilibrium condition. In order to estimate the scalability function empirically we propose an extension to a popular security game called Capture the Flag in Section 6.5. Starting with proposing the model, to creating diversity measures, to analysing the effect of user choice on diversity using games, this testing methodology completes the final part of our attempt to formalize software diversity.

7.2 Limitations

Here we discuss limitations in our approach and also comment on what we believe are some inherent difficulties in modelling such a complex system.

7.2.1 Limitations in our approach

Software-vulnerability equivalence

The assumption of *Software-vulnerability equivalence* in our model lets hosts choose any combination of vulnerabilities greatly increasing the total choice available to them. In the real world, a host gets to choose from alternatives of software or software components, while the choice of attackers are vulnerabilities themselves. This asymmetry of real world choice between attackers and defenders is lost in our bipartite model. By introducing a layer of components in the graph, we could have fixed this problem, but at the cost of complicating the process of calculating diversity measures.

Vulnerability equidistribution

Vulnerability equidistribution assumption can be critiqued on the grounds that some hosts are designed to be more secure than others. Operating systems with security in mind, like OpenBSD, or security enhancing frameworks, like SELinux, are regarded to be more effective against exploits if configured correctly. We could have distributed edges among hosts in a non-uniform manner but we weren't sure how to quantify the relative security of real world hosts and obtain data on the number of relatively *"secure"* hosts and *"insecure"* ones.

Entropy measures sensitive to sample size

We introduced entropy as a measure of diversity in Section 4.2. Diversity measures are sensitive to sample sizes. This means that in order to accurately calculate diversity in a real world ecosystem, we need actual numbers of computers on which each individual piece of software runs and not just market share values. In our real world software ecosystem these numbers are usually hard to obtain for proprietary software and hard to accurately estimate for open source software. In our study of diversity measures, we have not investigated the effect of sample sizes on entropy calculation, although we did mention that our illustrative example was only an approximation.

7.2.2 Limitations in modelling

Heterogeneity in vulnerabilities

Our model assumes that any single vulnerability can be used to create an exploit. In the real world, however, exploits may need to use a chain of vulnerabilities to compromise hosts. These vulnerabilities may be software bugs, network configurations errors, poorly chosen or stored passwords, bad cryptographic algorithms or sometimes even social networking websites. Every entity involved in the development, deployment or use of software is responsible for making choices that affect security and any model that tries to capture all possible cases and interactions quickly becomes too complex. Further, because of techniques like NAT, exploits may not be able to reach some hosts that run vulnerable software making it hard to exactly determine the potency of a vulnerability.

Utility of software

In our analysis of games in Chapter 6, agents' choice between alternatives of software was dictated only based on security concerns. But in reality, the utility of software for a majority of users can depend on several factors including features, usability, reliability and open or closed source nature. These extraneous and difficult to model incentives place an enormous burden on the part of the analyst trying to study purely security aspects of choice in a software ecosystem. One of the primary reasons we choose Capture the Flag as a platform for obtaining user's security preferences was because it is a security-focused setting where other factors of software play no role, unlike the real world where extrinsic motives interfere.

7.3 Future Work

We believe that the framework we present for formalizing diversity is only a starting point to understand the full implications diversity in security. This first attempt at formalization leaves many unanswered questions but also points to avenues for further research.

7.3.1 Theoretical improvements

Our formulation of this problem can be extended theoretically in several different directions. Here we list a few interesting ones.

Improving the model

The first step in improving the model is to introduce the software component layer between hosts and vulnerabilities. Each component would be connected to many vulnerabilities and hosts would get to choose components only. In order to simulate the layers in our software architecture, components could be chunked into categories and each host gets to pick a certain number of components from each category. For example one may enforce a kernel, a C library, two runtimes for high level languages (like Python or Java), several applications and a couple of browsers for each host. This would result in a more realistic model with several parameters. Tuning them independently may allow us to model desktop, server, mobile or even embedded software ecosystems separately and compare their diversities.

Attacker strategies

Our analysis focuses primarily on defenders (hosts) and defensive strategies while treating attackers passively. Attackers are assumed to prefer vulnerabilities that can compromise the largest number of hosts. In reality, the highly adaptive nature of attackers makes it necessary to consider both attackers and defenders simultaneously. Just like a user's choice of software depends on more factors than just security, an attacker's choice of vulnerability might depend on ease of creating exploits or making use of old ones, traceability of attacks, available computing power to launch attacks and so on. As defenders need to be only as diverse as is required to thwart attackers, it is necessary to understand attacker strategies to develop a more complete understanding of diversity.

Quantifying the amount of choice

In Chapter 6, we observed that the two components of diversity measures give rise to two different questions: how many alternatives are needed and how to distribute the alternatives. We focused on the second question, partly because it was easier to analyse. The first question is of course equally important. The problem is complicated because software development depends on finance, market requirements, perceived quality of existing software and in the case of open source software, developer interest. It remains to be seen if it's possible to abstract some parts of this and arrive at an answer to the following question: how many alternatives of software are sufficient?

Software coordination protocols

In Section 3.4, we looked at the simple problem of hosts coordinating in a distributed manner to select software to ensure increased diversity. If one of the hosts is malicious, it can influence what software gets chosen by the other host to influence the distribution of software in the ecosystem. We saw that even in the simple two host case, it is non-trivial to achieve this. For the n-agent case, consider the following formulation of the problem. Suppose there are n hosts and each host needs to select

k vulnerabilities from V. Assume that $kn > m = |V|$; hosts are forced to share some vulnerabilities. Suppose it is in the interest of each *honest* host to maximize the diversity of the resulting distribution. What distributed protocol should hosts use to negotiate who chooses which vulnerability in the face of a few *malicious* hosts trying to influence them. This problem is important because in practice virtually all software distribution and updates happen in a centralized manner. This puts a large number of hosts at risk if the distribution centre itself gets compromised. Identifying and solving this problem might be the first step towards a distributed software distribution infrastructure.

7.3.2 Practical improvements

This preliminary work on formalizing diversity has largely focused on the abstraction aspect of creating a theory. While it is still premature to make concrete recommendations to the software community, we can nevertheless suggest some mechanisms to increase diversity and propose a method to monitor diversity in the real world.

Intelligent package management

Linux distributions typically use package managers to maintain software on users' computers. A central repository of all packages is made available on the web and a user gets to choose which package is installed on her computer. Some distributions, like Gentoo, require users to select and compile all packages they need, but many popular distributions come with pre-selected packages. This introduces uniformity across installations, despite users having ample choice of alternatives in the repository. We suggest that for back-end packages that are interface-level compatible, a package manager would choose an alternative at random and install it. Users would of course be given the option to override.

For example, suppose that in Debian systems both SSL libraries, openssl and gnutls, are API-level compatible. In order to install an SSL library, a user issues *apt-get install libssl*. *apt-get* either randomly picks one of the alternatives or communicates with other installations for security track records to assess which to use for optimum diversity or total social payoff.

Interoperable components

Interoperability between software components is generally looked at as something desirable because it offers more choice to end users, encourages competition among developers, and prevents situations where users are locked into a vendor. Our analysis of diversity points out that security is another important reason to motivate development of interoperable software components. Having the ability to mix components dramatically increases the set of possible configurations in which an ecosystem can exist. Developing methods to more evenly distribute components is important to increasing diversity, but the first step is to create those components in various granularities. Security researchers who work on large scale systems should encourage development and use of interoperable software components.

Diversity observatory

Several organizations, like NIST, categorize and assign security ratings to publicly disclosed vulnerabilities. These systems can be extended with data about the number of installations of software or market share data to produce real-time diversity estimates of our software ecosystems. This would provide users with several benefits: they could query their software configuration to check if their installation is *typical* or they could receive suggestions from the observatory about possible alternatives to use for increased diversity and reduced possibility of their system to be compromised. In the reverse direction, we could gain a better understanding of how diversity is affected by the number of alternatives, the number and frequency of vulnerability compromises and users making use of choices available. Finally, the observatory would enable us to track diversity as a function of time.

7.4 Conclusion

Increasing the diversity of software running on systems has been proposed as a solution for mitigating attacks capable of a large scale compromise. Improvements in current defences have witnessed a concomitant increase in the sophistication of attacks calling into question the effectiveness of current defensive strategies to tilt scales favourably in the software security arms race. The ensuing debate on the merits and costs of diversity intermittently shows up in the sidelines of the agenda of several security researchers. Unfortunately, the broad and blanket nature of the concept of diversity offers only a shaky ground at best for arguments to stick. The primary purpose of this thesis was to affect remediations in this direction by making the notion of software diversity in the context of security more precise.

We formulated a model of a software ecosystem as a host-vulnerability graph and defined diversity metrics on it using entropy. With the support of several simplifying assumptions, this model allowed us to precisely formulate some new open problems in security. We then analysed the effect of six popular defences on the distribution of edges in the host-vulnerability graph to learn that none could alter the diversity of the ecosystem enough to deter scalable attacks. We employed game theory to explore tradeoffs in user choices that affect diversity and proposed a mechanism to emperically determine user preferences for diversifying in a security-sensitive context. In the final chapter, we discussed limitations in our model and assumptions, diversity metrics and game theoretic analysis and proposed directions for future work in improvements of theory and practical applications. We hope that our framework for understanding diversity lends clarity to the concept and stimulates research into creating diversity-type defenses that are effective against scalable attacks.

Bibliography

[1] Metasploit [last accessed: 8 august 2012]. [Online]. `http://metasploit.com/`.

[2] D. Aitel. 0days : How hacking really works. [Online]. `http://www.immunitysec.com/downloads/0days.pdf`.

[3] A. Avizienis. The n-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, 11(12):1491–1501, 1985.

[4] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 281–289, New York, NY, USA, 2003. ACM.

[5] K. Binmore. *Playing for Real: A Text on Game Theory.* Oxford University Press, 2007.

[6] K. Binmore and P. Klemperer. The biggest auction ever: The sale of the british 3g telecom licences. CEPR Discussion Papers 3214, C.E.P.R. Discussion Papers, 2002.

[7] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, New York, NY, USA, 1998.

[8] Y. Bramoull, D. Lpez-Pintado, S. Goyal, and F. Vega-Redondo. Network formation and anti-coordination games. *International Journal of Game Theory*, 33:1–19, 2004. 10.1007/s001820400178.

[9] S. Brilliant, J. C. Knight, and N. G. Leveson. Analysis of Faults in an N-Version Software Experiment. *IEEE Transactions on Software Engineering*, 16(2), February 1990.

[10] S. S. Brilliant, J. C. Knight, and N. G. Leveson. The consistent comparison problem in n-version software. *SIGSOFT Softw. Eng. Notes*, 12(1):29–34, Jan. 1987.

[11] S. L. Brumelle. When does diversification between two investments pay? *The Journal of Financial and Quantitative Analysis*, 9(3):pp. 473–483, 1974.

[12] R. Butler and G. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *Software Engineering, IEEE Transactions on*, 19(1):3 –12, Jan 1993.

[13] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 559–572, New York, NY, USA, 2010. ACM.

[14] P. Chen, G. Kataria, and R. Krishnan. Software diversity for information security. [Online], 2005. `http://www.infosecon.net/workshop/pdf/47.pdf`.

[15] P. Chesson and N. Huntly. The roles of harsh and fluctuating conditions in the dynamics of ecological communities. *The American Naturalist*, 150(5):pp. 519–553, 1997.

[16] F. B. Cohen. Operating system protection through program evolution. *Computers and Security*, 12(6):565 – 584, 1993.

[17] C. Cowan, S. Arnold, S. Beattie, C. Wright, and J. Viega. Defcon capture the flag: Defending vulnerable code from intense attack. In *DARPA DISCEX III Conference, Washington DC*, pages 22–24. IEEE Computer Society Press, 2003.

[18] C. Cowan, H. Hinton, C. Pu, and J. Walpole. The cracker patch choice: An analysis of post hoc security techniques. In *Proceedings of the National Information Systems Security Conference (NISSC)*, pages 16–19, 2000.

[19] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Symposium*, pages 63–78, 1998.

[20] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong, and J. Hiser. N-variant systems: a secretless framework for security through diversity. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[21] C. Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859.

[22] David and Andow. The extent of monoculture and its effects on insect pest populations with particular reference to wheat and cotton. *Agriculture, Ecosystems & Environment*, 9(1):25 – 35, 1983.

[23] C. S. Elton. *The Ecology of Invasions by Animals and Plants*. Methuen, London, 1958.

[24] R. A. Fisher, A. S. Corbet, and C. B. Williams. The relation between the number of species and the number of individuals in a random sample of an animal population. *Journal of Animal Ecology*, 12(1):pp. 42–58, 1943.

[25] S. Forrest, A. Somayaji, and D. Ackley. Building diverse computer systems. In *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*, pages 67 –72, May 1997.

[26] S. Forrest, A. Somayaji, and D. Ackley. Building Diverse Computer Systems. In *Proceedings of the* 6th *Workshop on Hot Topics in Operating Systems*, pages 67–72, 1997.

[27] M. Franz. E unibus pluram: massive-scale software diversity as a defense mechanism. In *Proceedings of the 2010 workshop on New security paradigms*, NSPW '10, pages 7–16, New York, NY, USA, 2010. ACM.

[28] D. E. Geer. Monopoly Considered Harmful. *IEEE Security & Privacy*, 1:14 & 17, November/December 2003.

[29] G. Goth. Addressing the Monoculture. *IEEE Security & Privacy*, 1(6):8–10, November/December 2003.

[30] T. Grenager, R. Powers, and Y. Shoham. Dispersion games: General definitions and some specific learning results. In *AAAI 2002*, pages 398–403. AAAI Press, 2002.

[31] A. Hector, B. Schmid, C. Beierkuhnlein, M. C. Caldeira, M. Diemer, P. G. Dimitrakopoulos, J. A. Finn, H. Freitas, P. S. Giller, J. Good, R. Harris, P. Hgberg, K. Huss-Danell, J. Joshi, A. Jumpponen, C. Krner, P. W. Leadley, M. Loreau, A. Minns, C. P. H. Mulder, G. O'Donovan, S. J. Otway, J. S. Pereira, A. Prinz,

D. J. Read, M. Scherer-Lorenzen, E.-D. Schulze, A.-S. D. Siamantziouras, E. M. Spehn, A. C. Terry, A. Y. Troumbis, F. I. Woodward, S. Yachi, and . J. H. Lawton. Plant diversity and productivity experiments in european grasslands. *Science*, 286(5442):1123–1127, 1999.

[32] M. O. Hill. Diversity and evenness: a unifying notation and its consequences. *Ecology*, 54(2):427–432, 1973.

[33] J. G. Horsfall. *Genetic vulnerability of major crops.* National Academy of Sciences, 1972.

[34] S. H. Hurlbert. The non-concept of species diversity: a critique and alternative parameters. *Ecology*, 52:577–586, 1971.

[35] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 272–280, New York, NY, USA, 2003. ACM.

[36] P. Klemperer. How (not) to run auctions: The european 3g telecom auctions. *European Economic Review*, 46(45):829 – 845, 2002.

[37] J. C. Knight and N. G. Leveson. An experimental evaluation of the assumption of independence in multi-version programming. *IEEE Transactions on Software Engineering*, 12:96–109, 1986.

[38] D. M. Kreps, P. Milgrom, J. Roberts, and R. Wilson. Rational cooperation in the finitely repeated prisoners' dilemma. *Journal of Economic Theory*, 27(2):245 – 252, 1982.

[39] E. Lefroy and R. Hobbs. *Agriculture as a Mimic of Natural Ecosystems.* Springer, 1998.

[40] C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 290–299, New York, NY, USA, 2003. ACM.

[41] J. A. List. Friend or foe? a natural experiment of the prisoner's dilemma. Working Paper 12097, National Bureau of Economic Research, March 2006.

[42] P. Maddy. Believing the axioms. ii. *Journal of Symbolic Logic*, 53(3):736–764, 1988.

[43] A. E. Magurran. *Ecological diversity and its measurement.* Croom Helm Limited., London., 1988.

[44] K. Mccann. The diversity-stability debate. *Nature*, 405:228–233, 2000.

[45] NetApplications. Net market share [last accessed: 10 july 2012]. [Online]. `http://netmarketshare.com`.

[46] S. Neti, A. Somayaji, and M. Locasto. Software diversty: Security, entropy and game theory. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Security*, HotSec'12, Bellevue, WA, USA, 2012. USENIX Association.

[47] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

[48] NIST. National vulnerability database [last accessed: 10 july 2012]. [Online]. `http://nvd.nist.gov/`.

[49] A. J. O'Donnell and H. Sethu. On achieving software diversity for improved network security using distributed coloring algorithms. In *Proceedings of the 11th ACM conference on Computer and communications security*, CCS '04, pages 121–131, New York, NY, USA, 2004. ACM.

[50] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, 1994.

[51] A. Ozment and S. E. Schechter. Milk or wine: does software security improve with age? In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, Berkeley, CA, USA, 2006. USENIX Association.

[52] PaX. Address space layout randomization (aslr). [Online], 2001. `http://pax.grsecurity.net/docs/aslr.txt`.

[53] A. Peslyak. return-to-libc attack, August 1997.

[54] S. Pimm. The complexity and stability of ecosystems. *Nature*, 307:321–326, Jan. 1984.

[55] E. Rasmusen. *Games and information: An introduction to game theory.* Wiley-blackwell, 2007.

[56] E. Rescorla. Is finding security holes a good idea? *Security Privacy, IEEE*, 3(1):14 –19, jan.-feb. 2005.

[57] C. Ricotta and L. Szeidl. Towards a unifying approach to diversity measures: Bridging the gap between the shannon entropy and rao's quadratic index. *Theoretical Population Biology*, 70(3):237 – 243, 2006.

[58] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu. A survey of game theory as applied to network security. In *Proceedings of the 2010 43rd*

Hawaii International Conference on System Sciences, HICSS '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[59] A. Rubinstein. Comments on the interpretation of game theory. *Econometrica*, 59(4):909–24, July 1991.

[60] P. A. Samuelson. General proof that diversification pays. *The Journal of Financial and Quantitative Analysis*, 2(1):pp. 1–13, 1967.

[61] I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela. Software diversity: state of the art and perspectives. *International Journal on Software Tools for Technology Transfer (STTT)*, 14:477–495, 2012. 10.1007/s10009-012-0253-y.

[62] Secunia. Secunia yearly report 2011. [Online]. `http://secunia.com/?action=fetch&filename=Secunia_Yearly_Report_2011.pdf`.

[63] G. Serazzi and S. Zanero. Computer virus propagation models. In M. Calzarossa and E. Gelenbe, editors, *Performance Tools and Applications to Networked Systems*, volume 2965 of *Lecture Notes in Computer Science*, pages 26–50. Springer Berlin Heidelberg, 2004.

[64] H. Shacham, E. jin Goh, N. Modadugu, B. Pfaff, and D. Boneh. On the effectiveness of address-space randomization. In *CCS 2004: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 298–307. ACM Press, 2004.

[65] S. Sidiroglou and A. Keromytis. Countering network worms through automatic patch generation. *Security Privacy, IEEE*, 3(6):41 – 49, Dec 2005.

[66] J. M. Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1st edition edition, Dec 1982.

[67] A. Somayaji. How to win an evolutionary arms race. *Security Privacy, IEEE*, 2(6):70 – 72, Dec 2004.

[68] A. Somayaji, M. Locasto, and J. Feyereisl. The future of biologically-inspired security: is there anything left to learn? In *Proceedings of the 2007 Workshop on New Security Paradigms*, NSPW '07, pages 49–54, New York, NY, USA, 2008. ACM.

[69] A. B. Somayaji. *Operating system stability and security through process home-ostasis*. PhD thesis, University of New Mexico, 2002. AAI3058952.

[70] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo. On the Infeasibility of Modeling Polymorphic Shellcode. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 541–551, 2007.

[71] M. Stamp. Risks of Monoculture. *Communications of the ACM*, 47(3):120, March 2004.

[72] D. Tilman. Biodiversity: Population versus ecosystem stability. *Ecology*, 77(2):pp. 350–363, 1996.

[73] M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. Flipit: The game of "stealthy takeover". Cryptology ePrint Archive, Report 2012/103, 2012. `http://eprint.iacr.org/`.

[74] H. R. Varian. System reliability and free riding. In *Economics of Information Security*. Kluwer Academic Publishers, 2004.

[75] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior (Commemorative Edition) (Princeton Classic Editions).* Princeton University Press, May 2004.

[76] C. Wang, J. Hill, J. Knight, and J. Davidson. Software tamper resistance: Obstructing static analysis of programs. Technical report, Charlottesville, VA, USA, 2000.

[77] K. Wang, G. Cretu, and S. Stolfo. Anomalous payload-based worm detection and signature generation. In A. Valdes and D. Zamboni, editors, *Recent Advances in Intrusion Detection*, volume 3858 of *Lecture Notes in Computer Science*, pages 227–246. Springer Berlin / Heidelberg, 2006. 10.1007/11663812_12.

[78] K. wei Lye and J. M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005.

[79] J. A. Whittaker. No Clear Answers on Monoculture Issues. *IEEE Security & Privacy*, 1(6):18–19, November/December 2003.

[80] R. H. Whittaker. Dominance and diversity in land plant communities. *Science*, 147:250–260, 1965.

[81] A. Wool. A quantitative study of firewall configuration errors. *Computer*, 37(6):62 – 67, june 2004.

[82] Y. Zhang, H. Vin, L. Alvisi, W. Lee, and S. K. Dao. Heterogeneous networking: a new survivability paradigm. In *Proceedings of the 2001 workshop on New security paradigms*, NSPW '01, pages 33–39, New York, NY, USA, 2001. ACM.

[83] Y. Zhu, H. Chen, J. Fan, Y. Wang, Y. A. N. Li, J. Chen, J. Fan, S. Yang, L. Hu, H. E. I. Leung, T. W. Mew, P. S. Teng, Z. Wang, and C. C. Mundt. Genetic diversity and disease control in rice. *Nature*, 406:718–722, Aug. 2000.