Software diversity: Security, Entropy and Game Theory

Saran Neti
Carleton University
saran@ccsl.carleton.ca

Anil Somayaji
Carleton University
soma@ccsl.carleton.ca

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

Abstract

Although many have recognized the risks of software monocultures, it is not currently clear how much and what kind of diversity would be needed to address these risks. Here we attempt to provide insight into this issue using a simple model of hosts and vulnerabilities connected in a bipartite graph. We use this graph to compute diversity metrics as Renyi entropy and to formulate an anti-coordination game to understand why computer host owners would choose to diversify. Since security isn't the only factor considered when choosing software in the real world, we propose a slight variation of the popular security wargame Capture the Flag that can serve as a testbed for understanding the utility of diversity as a defense strategy.

1 Introduction

Monocolture first became a significant problem in the agriculture industry where use of a single variety of seed can make an entire harvest susceptible to complete destruction by a single pest [3]. The software ecosystem faces a similar problem as noted by several researchers [14, 5, 18]. Specifically, the over-reliance on certain pieces of software, whether they be operating systems or applications, has been cited as increasing the likelihood and severity of widespread security compromises.

Software diversity is an intuitive but imprecise term used to express the idea that variability can improve the survivability of a population. Diversity as a mechanism for fault tolerance was put forth as n-version programming [1] in mid-1980s. Diversity for security exists in various forms including compile time diversity [4], run time diversity [8], automatic patch generation [13] and automatic signature detection [16]. While diversity-inspired defense strategies can have limitations (e.g., ASLR derandomization attacks [12]), they can also be very effective in practice, particularly to mitigate memory corruption attacks. To date, however, diversity/randomization strategies have not been shown to provide general increases in software security.

To better understand in what circumstances diversity is an effective defense strategy and the kind of diversity that is needed to provide protection, we believe we need a more rigorous understanding of the concept of diversity. There has been some work on formalizing diversity in the past. For example, O'Donnell and Sethu analyzed distributed coloring algorithms to assign packages to systems on a network for increased diversity [11], and Chen et. al. estimated diversity based on tradeoffs of interoperability and security [17].

In this paper we take a new approach to formalizing diversity by modeling it as the entropy computed using a bipartite graph interconnecting *hosts* and *vulnerabilities*. While our model is built upon many simplifying assumptions, as we will show it maintains enough richness to enable a plausible, entropy-based measurement of diversity of deployed systems. It also allows us to explore diversity from a game theoretic perspective.

This paper makes three contributions. First we introduce entropy as a measure of diversity for software and use it to numerically estimate the diversity of software ecosystems. Second, we introduce an anti-coordination game that captures the interplay of choice, diversity, and the scalability of risk. Finally, we present a slight modification to a popular online security game "Capture the Flag" that could potentially help explore the utility of diversity as a strategy in the context of cyber conflict.

2 Model and Assumptions

To create an analytically tractable model of software diversity, we need to make a number of simplifying assumptions. We have chosen to model diversity by focusing on the interactions between hosts and vulnerabilities. The rationale for this choice is that all hosts must run some set of software, and any such software

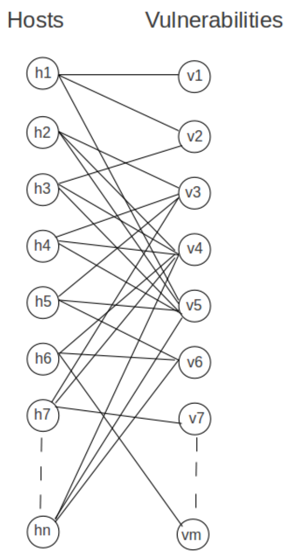


Figure 1: An example graph G of non-uniform vulnerability distribution with n hosts, m vulnerabilities and 3 vulnerabilities per host

will have some number of vulnerabilities. We are specifically not specifying what it means for two hosts to have the "same" vulnerability. In prinicple, two very different pieces of software could be compromised by the same exploit even if they share no code in common. In practice, however, common vulnerabilities generally derive from common code or shared specifications. Thus, by focusing on hosts and the vulnerabilities they have, we can avoid the complexities of modeling software explicitly while still capturing the essence of diversity, namely that multiple targets will not all be compromised by the same attacks.

More precisely, 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. These relationships can be visualized as a bipartite graph as shown in Figure 1. Hosts and vulnerabilities comprise vertices of a bipartite graph and an edge between h_i and v_j indicates that host h_i runs software that contains vulnerability v_j.

Implicit in this model are a number of simplifying assumptions:

1. **Computer equivalence:** We assume that each host is of equal value to the attacker. This assumption is meant to approximate attackers targeting large numbers of consumer machines, rather than targeted attacks against corporate or military systems.

2. **Software vulnerability equivalence and equidistribution:** Vulnerabilities cannot be completely eliminated from a large software stack. So, instead of saying a host chooses software, we say a host chooses vulnerabilities. Further, we assume every host has to choose k vulnerabilities—every host vertex $h_i \in H$ has the same degree k.

3. **Vulnerability criticality and reachability:** We assume that all vulnerabilities are equally hard to find and the result of exploiting any such vulnerability is the same: complete compromise of the host.

4. **Steady State assumption:** Our analysis is confined to steady state where the number of vulnerabilities that exist do not change over time.

Clearly these assumptions are significant simplifications. But consider this: high value targets on the Internet are, by their nature, rare relative to numerous low value consumer systems. Less critical vulnerabilities are often chained together to produce attacks that completely breach system security. Major operating systems and applications all consist of comparable volumes of code, and all of these code bases have been shown to have significant numbers of vulnerabilities. And, the rate of vulnerability disclosure in popular code bases continues to be large, even though software is being patched at an ever increasing rate. Thus, while our assumptions preclude our model from applying to every computer system, it does allow us to capture the dynamics of commodity systems running mainstream software that are in a constant state of code flux due to frequent patching—the situation for most hosts connected to the Internet today.

3 Diversity measures

We now use the bipartite graph G (see Figure 1) to represent diversity as follows. If the 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.

We now wish to formalize this notion. Diversity measures as used in ecology literature are an obvious choice. However, there is no one unified measure that captures everything about diversity [7]; instead, we have a continuum of measures. 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.

Definitions. (See [7]) Given the fraction of edges connected to each vulnerability vertex, p_i, Diversity number N_a is defined as :

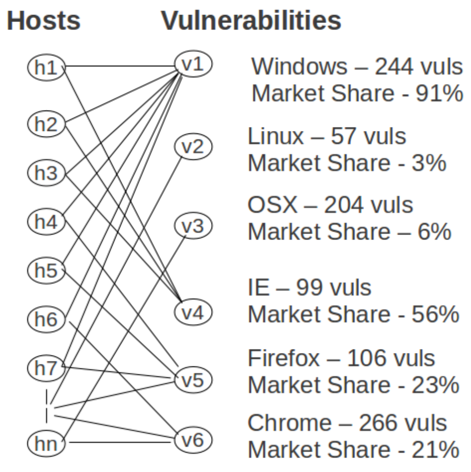


Figure 2: Market share data and the number of vulnerabilities reported in NVD for Operating Systems and Browsers in 2011. Every host chooses an OS and a browser. Note, to simplify presentation in this figure each v_i represents the set of vulnerabilities associated with a given piece of software.

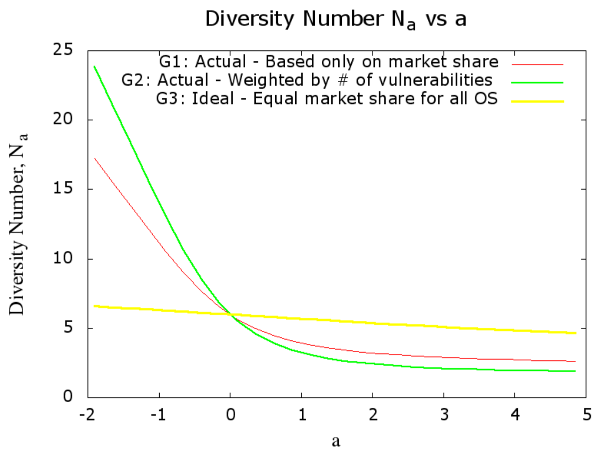


Figure 3: Calculating diversity numbers N_a for the software ecosystem shown in Figure 2. Note that Renyi entropy $H_a = log(N_a)$ and a special case of that when $a = 1$ is Shannon entropy. $G1$ is obtained by treating all vulnerabilities from each category as just one vulnerability. $G2$ is calculated by weighing each category with the number of vulnerabilities. $G3$ is for when all OSs have equal market share.

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

N_a is the reciprocal of the $(a-1)^{th}$ root of weighted mean of the $(a-1)^{th}$ powers of p_i where weights themselves are the relative proportion of edges connected to vulnerabilities. N_0 counts the number of vulnerabilities while for $a = 1$, we get $N_1 = \lim_{a\to 1} N_a$ which corresponds to Shannon entropy, i.e $log(N_1) = H = -\sum_{i=1}^{m} p_i log(p_i)$. As the value of a increases from $-\infty$ to $+\infty$, the diversity number N_a changes the weightage assigned from the least connected vulnerabilities to highly connected vulnerabilities.

For an illustration on how to calculate diversities, we look at a real world example. Figure 2 shows a host-vulnerability graph of desktop operating systems during 2011. Market share data is taken from [9] and publicly disclosed vulnerability statistics are taken from NVD [10]. We assume that every host has to choose one OS and one browser. Further, for simplicity we assume that all three browsers run on all operating systems[1] Figure 3 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 supported by the fact that a host cannot choose some vulnerabilities from "Chrome" and some from "Firefox", it has to choose software as a whole. 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 one can compute N_a. In $G1$, the number of vulnerabilities/host is constant as in our original model, but for that we disregared the actual number of vulnerabilities reported.

In $G2$, we weight each vulnerability category (e.g "Windows") with the number of vulnerabilities reported (e.g 244). In this case different combinations of OS and Browser will lead to different numbers of vulnerabilities/host. $\{p_i\} = \{244 * \frac{0.91n}{tot}, 57 * \frac{0.03n}{tot}, 204 * \frac{0.06n}{tot}, 99 * \frac{0.56n}{tot}, 106 * \frac{0.23n}{tot}, 266 * \frac{0.21n}{tot}\}$, 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 comparision 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.

Which specific measure to use depends on relative weighting in the exponent one wishes to use for the least

[1]Note that technologies such as WINE for running Windows binaries on Linux and MacOS make this assumption not completely arbitrary.

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 systems at the kernel, userspace, and application level especially since several variants and forks exist. Granularity of choice is a crucial difference in this example from our ideal model, and entropy as a diversity measure is highly sensitive to this kind difference. 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 the SSL libraries *openssl* and *gnutls* are 100% API-level compatible so each can be replaced with the other. If this were the case, then a user or a package manager, depending on market share values and security track records, could decide which library to install and use.

4 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 play. 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 [15]. In diversity, we would like to analyse choice—how much choice is needed—and given some amount of choice, how does a host choose. In this section we study the second half of this question.

Anti-coordination games. These are two player, two strategy games where it is preferred that both players choose different strategies. Consider two hosts h_1, h_2 and two vulnerabilities v_1, v_2, such that both h_1 and h_2 are connected to v_1. Suppose v_1 gets compromised. Each host now has two choices : to stay with v_1 or to switch to v_2. 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_1, 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. The corresponding payoff matrix is shown in Figure 4(a). An interesting case is when $c_w < c_2$, i.e., it is less expensive to switch than to risk sharing a vulnerability. There are two pure strategy Nash equilibria, $(Stay, Switch)$, $(Switch, Stay)$ and one mixed strategy Nash equilibrium where both players choose to stay with a probability $(c_2 + c_w)/2c_2$. In a large population of hosts, the interpretation of a mixed strategy Nash equilibrium implies that even though software community might make it cheaper to switch to an alternative software, some hosts will choose to stay with their existing software. This indicates that over and above providing ample choice for hosts some form of coordination to distribute software is useful.

Dispersion Games. Two player anti-coordination games generalized to many players and actions are called Dispersion games [6]. 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. In our case we assume that any host can choose any vulnerability, i.e., all agents have the same set of actions A. Some assumptions made in Section 2 translate to assumptions about symmetries in Dispersion games, e.g *Vulnerability criticality* translates to *Action symmetry*. (see [6] for definitions).

For diversity, we study a specific game where a maximally dispersed outcome is preferred but it costs to switch to strategies that lead to it. Figure 4(b) 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 $share(x)$ while the risk associated with increased number of hosts choosing the same software leading to higher probability of attacks might be $risk(x)$. We define the scalability cost as a monotonic function $\Pi(x) = risk(x) - share(x)$. Let c_0 be the intrinsic cost of choosing a lone vulnerability, v, i.e., $deg(v) = 1$. Then $c_0\Pi(x)$ is the cost of choosing v such that $deg(v) = x$.

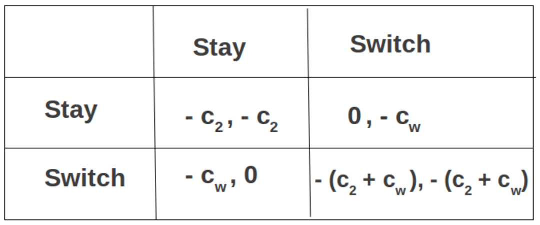
As before, let $c_w =$ cost of switching from one vulnerability to another and $o =< a_1, a_2, ..., a_n >$ be an outcome of this game where $a_i \in V$ is the action taken by host h_i. Further let $n^o_{a_i} =$ number of hosts choosing a_i in o. The utility of a host does not depend on the specific hosts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

$$U_{a_i}(o) = -c_0\Pi(n^o_{a_i}) - c_w(1 - \delta_{a_i a_c}) \qquad (2)$$

where δ_{ij} is the Kronecker delta function. If a host chooses a_j over a_k, then we must have

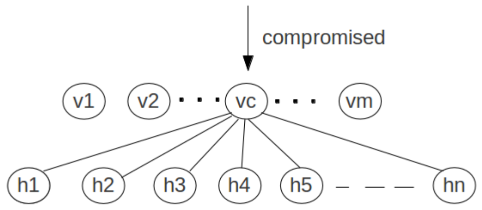
$$c_0(\Pi(n^o_{a_k}) - \Pi(n^o_{a_j})) + c_w(\delta_{a_j a_c} - \delta_{a_k a_c}) \geq 0 \qquad (3)$$

If $j \neq c$ and $k \neq c$, we get $\Pi(n^o_{a_k}) \geq \Pi(n^o_{a_j})$, and symmetrically, a host that chooses a_k over a_j must have



	Stay	Switch
Stay	$-c_2, -c_2$	$0, -c_w$
Switch	$-c_w, 0$	$-(c_2+c_w), -(c_2+c_w)$

(a) Payoff matrix for a two host, two vulnerability game with both hosts connected to v_1 initially. The intrinsic cost of choosing a vulnerability, c_0 (not shown), is common in all cases. c_2 is the additional cost of sharing a vulnerability and c_w is the additional switching cost.



(b) 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 scalability factor defined in Section 4.

Figure 4: Anti coordination games with switching costs

$\Pi(n^o_{a_k}) \leq \Pi(n^o_{a_j})$. 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.

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

$$\Pi(n_s) = \Pi(n - (m-1)n_s) - c_w/c_0 \tag{4}$$

We, of course, do not know what the scalability factor $\Pi(x)$ is, but if we did know that and 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 4. 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.

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 [2]. Typically a team is provided with m, e.g 10, vulnerable services. Game server preiodically 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 for points.

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.

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 gameplay. This last point is the essence of this paper: choice matters. 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 Conclusion

In this paper we define software diversity in terms of the Renyi entropy based on a bipartite graph of hosts and vulnerabilities. We show how this measure can be used to quantify diversity in actual software ecosystems, and how it can be used to define games, both formal and playable, that capture attacker/defender dynamics in the context of diversity.

We believe that the framework we present for formalizing diversity can serve as a guide for developing future defenses. In particular, our models suggest that we need to significantly reduce the overlap of vulnerabilities between hosts if diversity is to be an effective defense strategy. We postulate that to achieve such dispersion of vulnerabilities, we would need the software on each computer system to be unique in behavior, if not in composition. Achieving this level of software diversity in a way that is practical from both a usability and cost perspective is, we believe, an important goal for future research.

It is possible that currently existing security technologies, such as anomaly-based intrusion detection, may already be diversity-type defenses as characterized by our definition. Our hope is this work will inspire others to analyze proposed and deployed defenses in terms of our diversity framework.

Acknowledgements

The authors would like to thank Jeremy Clark, Philip Fong, and Payman Mohassal for comments and feedback on aspects of this work. This work was supported by Canada's Natural Sciences and Engineering Council (NSERC), through their Discovery program (Somayaji, Locasto) and the ISSNet Strategic Network (Somayaji, Neti).

References

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

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

[3] 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.

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

[5] D. E. Geer. Monopoly Considered Harmful. *IEEE Security & Privacy*, 1(6):14 & 17, November/December 2003.

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

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

[8] 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.

[9] NetApplications. Net market share. http://netmarketshare.com [Last accessed: 26 April 2012].

[10] NIST. National vulnerability database. http://nvd.nist.gov/ [Last accessed: 26 April 2012].

[11] 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.

[12] 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.

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

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

[15] 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/`.

[16] 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.

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

[18] 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.