Redundancy and Diversity in Security

Bev Littlewood and Lorenzo Strigini

Centre for Software Reliability, City University,
Northampton Square, London EC1V OHB, U.K.
{B.Littlewood,L.Strigini}@csr.city.ac.uk

Abstract. Redundancy and diversity are commonly applied principles for fault tolerance against accidental faults. Their use in security, which is attracting increasing interest, is less general and less of an accepted principle. In particular, redundancy without diversity is often argued to be useless against systematic attack, and diversity to be of dubious value. This paper discusses their roles and limits, and to what extent lessons from research on their use for reliability can be applied to security, in areas such as intrusion detection. We take a probabilistic approach to the problem, and argue its validity for security. We then discuss the various roles of redundancy and diversity for security, and show that some basic insights from probabilistic modelling in reliability and safety indeed apply to examples of design for security. We discuss the factors affecting the efficacy of redundancy and diversity, the role of "independence" between layers of defense, and some of the trade-offs facing designers.

1 Introduction

This paper is about the need for better understanding of how redundancy and diversity are applicable in security. This involves different issues:

- the extent of their useful applicability;
- the ways of evaluating their advantages so as to drive design.

Redundancy as a general approach is clearly understood to be a valid defense against physical faults. There is a rich set of understood design "tricks" that use redundancy against various forms of faults and failures, and knowledge about how to optimize them for different purposes, e.g. in terms of tradeoffs between cost of the redundancy during normal operation and the effectiveness and performance degradation in case of failure. When it comes to design faults, it is commonly accepted that some form of diversity may be necessary to tolerate them. However, whether fault tolerance itself is an appropriate approach is more controversial: many claim that for "systematic" failures the most effective

This work was supported in part by the U.K. Engineering and Physical Sciences Research Council under projects DOTS (Diversity with Off-The-Shelf Component) and DIRC (Interdisciplinary Research Collaboration on Dependability of computer-based systems).

and cost-effective procedure is simply to avoid them in the first place. We have argued elsewhere that these claims are the result of misunderstandings about terms like "systematic". When it comes to *intentional* faults, the value of fault tolerance in general is even less commonly accepted, although some authors have long argued its benefits [1, 2] and despite a recent increase of interest.[1]

In part, the disagreements are an effect of the historical separation between the various technical sub-communities dealing with reliability/safety and with security, leading to different attitudes (not least to the use of probabilistic arguments), and some degree of reciprocal misunderstanding. A risk in cross-community discussion is over-generalization. "Redundancy" or "diversity" are just useful common names for generic design approaches, which have to be adapted to the needs of a specific design. All considerations of applicability and efficacy need to be referred to a specific situation, with its threats and dependability requirements. When we move from general discussions of applicability and intuitive desirability of these methods to deciding in which form (if any) they should be applied in a particular context, more formal, rigorous reasoning is needed. This has been provided, in reliability and safety, by probability models of diversity, giving some new, and sometimes surprising, insights (surveys of these results are in [3, 4]). We believe that these models are directly applicable to the use of diversity in security. This is the main topic of this paper. The recent resurgence of interest in diversity for security seems to include only limited interest in probabilistic study. In fact, some misunderstandings that arose early on about software fault tolerance seem to be occurring again, especially concerning notions of "independence". We will describe some aspects of the existing models that seem directly applicable to security. An area of security research in which there is interest in diversity and redundancy is intrusion tolerance, and we use this example in our discussion.

In the rest of this paper, we first deal with the preliminary issues. We argue the need for probabilistic reasoning about security (section 2). Then we discuss the concepts of redundancy and diversity and the ways of pursuing them (Sect. 3); and, in Sect. 4, we consider examples of these in recent work on security. We then proceed (section 5) to show how previous results about diversity can help in directing its application to improve security attributes of systems.

2 Probability, Redundancy and Security

Discussion of redundancy and diversity, whether it be for reliability, safety or security, must start from an understanding of uncertainty, and a recognition that probability is the appropriate formalism for dealing with uncertainty. This is accepted in the reliability area, because no redundant architecture deterministically prevents all system failures. One instead encounters frequent skepticism in the security community about the use of probabilities. This is reminiscent of

[1] Represented for instance by the U.S. DARPA sponsored OASIS project (http://www.tolerantsystems.org/) and the European MAFTIA project (e.g. http://www.newcastle.research.ec.org/maftia/). More references are in Sect. 4.

similar debates twenty years ago about the use of probabilities for software reliability. Then, it was said software failures were "systematic" and thus not open to probabilistic modelling. Now, it is said that security failures are deliberate and thus not open to probabilistic modelling. Both statements are, we believe, based upon mistaken views of the nature of the uncertainty in the two areas.

The term "systematic failure" arose to distinguish software failures (and other design fault-induced failures) from "random" hardware failures. The words "random" and "systematic" here are misleading: they seem to suggest that in the one case a probabilistic approach is inevitable, but that in the other we might be able to get away with completely deterministic reasoning. This is not so, and stochastic arguments seem inevitable in both cases.

When we use the word *systematic* here it refers to the fault mechanism, the mechanism whereby a fault reveals itself as a failure, and not to the failure *process*. In the case of software, the most widely recognized source of systematic failures, it is correct to say that if a program failed once on a particular input (and internal state), it would fail every time the same conditions occur again, until the offending program fault had been successfully removed. This contrasts with physical hardware failures, for which there is no such certainty. In this very limited sense, software failures are deterministic, and it is from this determinism that we obtain the terminology.

However, our interest really centers upon the failure *process*: what we see when the system under study is used in its operational environment. In a real-time system, for example, we would have a well-defined time variable (not necessarily real clock time) and our interest would center upon the process of failures embedded in time. We might wish to assure ourselves that the rate of occurrence of failures was sufficiently small, or that there was a sufficiently high probability of surviving some pre-assigned mission time. We would *not* instead be able to predict with certainty whether the next mission will end in failure: this depends on unknown details of the mission (the exact sequence of inputs to the system) and of the system (the possibility of unknown bugs). The important point is that this failure process is not deterministic for either "systematic" faults or for random faults.

Similar reasoning applies to security. One frequently hears objections to probabilistic measures of security because of the essentially unrepeatable nature of the key events. People are happy with estimates of the probability of failure of a hardware device because they can envisage testing a sufficient number of copies for long enough, and computing the estimate from the resulting data. For security - as for non-intentional software failures - the uncertainty often concerns one-off events. This requires a subjective, Bayesian interpretation of probability as "strength of belief".

Another objection to probabilistic reasoning about security stems from the deliberate nature of attacks. The attacker *knows* what he is doing, so where is the uncertainty? From the attacker viewpoint it lies, of course, in his uncertain knowledge about the *system*. The system owner, on the other hand, may have greater knowledge about the system, but is uncertain about the attackers' behavior. He knows that he is dealing with deliberately malign attackers, rather than merely randomly perverse nature, but this does not take away the intrinsic uncertainty about what he will see, e.g. when the next successful attack will occur.

So, like the notion of "systematic" failure, the notion of "deliberate" failure concerns the failure mechanism, not the failure process. To a system owner, observing the process of novel intrusions into his system, this will appear as a stochastic process. The nature of this random process characterizes the security of the system.

Of course, there are some ways in which the deliberate nature of (some) security events is important [5]. For example, in the previous paragraph we referred to *novel* intrusions: this is because once a security hole has been discovered, the subsequent process of intrusions will be different - the intruders may try to exploit the hole intensively - until the hole is fixed. Primary interest may be in the first process, e.g. in the random variable "time to first successful intrusion" - thus a simple example of a snapshot measure of security might be the mean of this random variable (*cf* mean time to failure). If holes are fixed as they are found, we may also be interested in the stochastic process of successive events: this is similar to the reliability growth processes for software *reliability.*

The important point here is that for security, as for reliability and safety, we must take account of *inherent* uncertainty. Eliminating the uncertainty completely is almost never an option: no system is completely secure, just as no system is completely safe, or completely reliable.

It follows that our discussion of the use of redundancy and diversity to increase security must be carried out in probabilistic terms. This need not mean that the probabilistic models will be the same as those used for reliability and safety, because the nature of the uncertainty may be different for security (although we believe that some models may carry across more-or-less unchanged).

By arguing for the *necessity* of a probabilistic approach, we do not intend to suggest that it presents no practical difficulties. Measurement and estimation are likely to be hard, and to depend upon a large element of expert judgement. But it is worth asking how judgements about system security are made *without* such a probabilistic formalism. What goes into the assertion "this system is sufficiently secure"? Indeed, what does "sufficiently" mean here?

The most-cited difficulty is that of estimating parameters, e.g. the probability of a specific kind of attack occurring over a certain period of time. From this viewpoint, security-relevant phenomena cover a range from recreational vandalism by large crowds, which allows extensive data collection and presumably some ability to extrapolate to the future (forecasts of mass human behavior are frequently successful in various areas, e.g. elections or buying preferences), to one-off attacks by hostile governments, for which statistical extrapolation will be close to impossible.

It is not necessary to have complete faith in the accuracy of numerical estimates of security for a probabilistic approach to have value. Even for hardware reliability, the numerical predictions obtained from "reliability models" are often just educated guesses: useful, but far from infallible. The strength of probabilistic methods is in allowing statements like: "with these components, design A will give better reliability than design B over short mission times", or "the crucial parameter for this system is the probability of errors being correctly detected in component C", which are true for wide ranges of the values of the model parameters. In more complex uses like safety cases, the main value of the use of probabilities is often in the way that formal reasoning allows an expert's argument to be laid out for scrutiny and critique. It enables third parties to question the contributions to the expert's decision of his various information, beliefs, assumptions and reasoning.

3 Redundancy, Diversity and Dependence

Redundancy and diversity are widely used to protect against mistakes and failures. Applications range from quite informal usage - e.g. having someone else check your arithmetic - to engineered fault tolerant systems.

If we look at the simple case of parallel redundant systems, the term "redundancy" usually indicates simple replication of a component in identical copies, as adopted against "random" hardware failures. The term "diversity" has come to be used especially for "multiple version software", in which redundant software "versions" are deliberately made to be different. This is because multiple copies of a program, with exactly the same fault, may provide little protection against software failures. With diverse versions, one hopes that any faults they contain will be sufficiently different for the versions to show different failure behaviour.

The distinction between redundancy and diversity is not a hard and fast one, but the notion of *deliberate difference* is the key to their use in computer systems to protect against design faults. This difference can be achieved in several ways, based on enforcing differences in the ways the versions are built. Thus different design teams may be used, different software engineering practices, etc.

It is here that the word "independent" has been over-used. In the early literature on software fault tolerance, it is possible to find people writing about "independent" teams building "independent" versions, in the hope that these would fail "independently". It is really only the last of these uses of the word that is formally defined - and this statistical independence of version failures would indeed be a worthy goal. It would, for example, allow us to claim a probability of failure on demand (*pfd*) of 10^{-6} for a 1-out-of-2 system built from two versions each having *pfd* 10^{-3}. Claims for independence of failure are, unfortunately, hard to justify and rarely (if ever) correct. Indeed, experiments have shown that real software versions had failure processes that are quite strongly correlated, so that systems built from such versions would be much less reliable than an independence assumption would suggest. Nevertheless, there was benefit in the use of diversity: the multiple version systems were a lot more reliable on average than individual versions. The lesson here is that the gain from the use of diversity will depend on the degree of dependence between the failure processes of the versions, not only on their individual reliabilities.

We would stress that much of what we say applies more widely than to multi-version software. For example, work on diverse software fault-finding procedures [6] has shown (admittedly in an experimental context) that "better than independence" can actually be attained. In terms of system structure, there is no need for the complete similarity of functionality, implicit in parallel-redundant systems. One "version" may have a simple "get you home in an emergency" function, or a monitor/watchdog function, next to a "main" version of much greater complexity and functionality. But in all applications of diversity the key lies in *dependence* - we need to make this as low as we can to *achieve* dependability; we need to evaluate it in order to *assess* dependability.

Failure independence itself is clearly not the optimum result. The best effect of diversity would be a situation in which all the circumstances in which one version fails are ones where another succeeds, and vice-versa, so that the probability of common failure is 0. In fact, looking at designers' attitudes to seeking diversity between redundant subsystems, we can identify different categories, with different ideal best results (the names used below are tentative, just for use in our discussion):

1. "separation": designers simply seek to isolate redundant subsystems from as many as possible common potential causes of failure. To tolerate physical faults, this implies physical separation, separate power supplies, etc. In multiple-version software, one usually attempts to isolate the development teams so that biases and misunderstandings do not propagate between them. It is natural to see failure independence as the optimum that this approach can aim for, though one would not expect to achieve it. The models of Eckhardt and Lee and Hughes (see [3, 4]) show that any remaining common influence on the failure processes of two subsystems, *including the very fact of receiving the same input sequences*, will lead to positive correlation of failures;
2. "forced diversity": trying to diversify the way the unavoidable common influences affect the redundant subsystems. Using different, functionally equivalent components in redundant subsystems eliminates the certainty that any design faults will be identical among the subsystems. Forcing developers of "diverse" software versions to use different algorithms for the same function should reduce the risk that the common difficult areas in the requirements will lead to common mistakes in the implementations [7]. There is no clear guide to how much advantage this approach should produce, but the model of Littlewood and Miller (see [3, 4]) shows that, at least in some scenarios, everything else being equal, it can only be an improvement (i.e., reduce correlation between subsystem failures) and makes the goal of negative correlation between failures and even zero failure rate at least theoretically achievable;
3. "tailored diversity": if the designers know in some detail how their precautions affect susceptibility to causes of failure, this allows them to focus "forced diversity" for lower correlation between the effects of common influences on the redundant subsystems. For instance, redundant hardware subsystems could intentionally be selected so that they are most reliable in

different temperature ranges, within the range in which the redundant system is to operate. For multiple-version software, instead, there is rarely any attempt to "tailor" the versions to the particular kinds of faults that might be anticipated [7]. In any form of "defense in depth" (including computer security), this form of diversity demands that each successive layer of defense be especially strong against those threats most likely to penetrate the other layers. Again, the ideal system-level goal is 0 failure rate, though it may be unattainable due to the inevitable "leakage" of each level.

In the next sections, we look at various possible applications of diversity for security, and discuss them in light of these different possible approaches.

4 Redundancy and Diversity for Security

"Security" encompasses multiple attributes (confidentiality, availability, . . .) and defending against multiple threats. Just as in other areas of dependability, different security attributes may create conflicting demands on designers. Redundancy and diversity, in their turn, come in many forms.

Voted redundancy, often seen as the stereotypical form of redundancy against accidental faults, is comparatively rare in security. It may be used for decisions on trust, when there is low enough probability that a majority of the parties involved in voting has been compromised. A designer can adjust the degree of majority required for the desired trade-off between the needs for a low probability of the "bad guys" gaining a majority and determining the decision's outcome, and for a high enough probability of a decision being reached. The methods are the same as in design for reliability.

More common is the case of redundancy of resources, in an (ideally) 1-out-of-N configuration (i.e., in which the service can be provided, possibly in a degraded fashion, if at least 1 of the N redundant resources remains available). For instance, if a server (for any kind of service) can be disabled by an attack, having multiple servers is a defense. In general, whenever the goal of an attack is to cause effects similar to those of a physical fault, e.g. unavailability of a resource, it is natural for designers to consider the same defense. So, security benefits, in terms of ability to guarantee a service even after an attacker has inflicted some damage, are obtained from replication of communication lines, of messages, of stored data; and from data redundancy, watchdog and audit programs for detecting damage. Some security-oriented designs, like "secret sharing" [8, 9], combine resilience against partial damage with constraints on the loss of confidentiality that a successful attack on a subset of the servers can achieve.

It is in the forms of error propagation and common-mode failures that the analysis of a redundant scheme from a security viewpoint will differ from the analysis from a reliability viewpoint. The main objection to trusting redundancy for security is that if an attacker can defeat a certain system or defense, the same attacker will have no trouble defeating two copies of the same. This kind of objection cannot be discussed without reference to the multiplicity of security attributes, threats and design possibility mentioned before.

For instance, suppose we have multiple servers in an 1-out-of-N configuration. An example would be the Internet's root Domain Name Servers, which were repeatedly attacked recently [10]. They are meant to be a 1-out-of-13 parallel-redundant system. Against certain threats, this set-up is clearly effective:

– accidental local faults (hardware faults, flood, fire) at different sites, without a common cause, will be close to being independent events, giving vanishingly small probabilities of common failure of all 13: these events can reasonably be neglected in estimating the failure probability of the whole parallel system;
– physical attack to the premises of all the servers will require close to N times the effort required to attack one of them. How much this reduces the probability of system failure would be complex to assess, requiring assumptions about the attackers' resources and cost-benefit trade-offs, but still redundancy can be seen to be clearly effective.

Indeed [11] wrote before the attacks: "...root servers are extremely secure ... The protocols that govern the 13 root servers call for complete diversity. The servers are geographically distributed, ... Those in the United States are almost evenly divided between the East and West coasts ....".

However, a common argument goes, attackers can easily steal resources around the Internet to use in massive attacks, making the cost of attacking all servers much lower. This makes it more likely that an attack that disabled one server could disable them all. Failures of the redundant channels will have high positive correlation, and the parallel-redundant nature of the system would no longer be a very useful defense. To continue the example, consider a different kind of attacker, one who would try to penetrate, undetected, the host computers to produce some subtler corruption of their data and software. How effective would redundant defenses then be?

– suppose that the attacker needs to guess a key or password for each server (and these have been chosen well to make the attack expensive). Then, the attacker's required effort is effectively multiplied by N;
– but if the attacker instead discovers a defect that creates a back door in all servers, the extra effort due to the multiple servers will be minimal. To avoid this risk, one would probably want some diversity of software among the servers (which in this example probably exists, although it did not seem important to the author of [11]), in addition to geographical separation;
– or the attacker could aim to penetrate just one server but to create local damage there that will subtly subvert the whole redundant set. The multiple servers become effectively a series system: compromising one will cause the whole system to fail.

These complexities are often quoted to caution against expecting too much help from redundancy. However, they are not specific to security. Designers in other areas of dependable design are familiar with similar complexities; one would not expect memory error-correcting codes to be a defense against application software bugs, or replicated CPUs to defend against power failures; or arrangements for safety to automatically benefit availability.

What can be generalized from these examples is perhaps obvious:

- different threats call for different kinds of redundancy, even in the same system;
- whether a certain kind of redundancy is a cost-effective defense depends on the detailed circumstances of the design problem;
- trade-offs may be required between various requirements, including security requirements. These inevitably require quantitative (albeit approximate) reasoning. Researchers have recently started publishing studies of security scenarios using tools originally developed for studying design compromises in fault-tolerant systems [12, 13].

A subtler point concerns “diversity”. In which sense is the dispersion of servers over the world “diversity”? What does it add to simple “redundancy” (replication)?

This leads back to the various forms of “diversity” introduced in Sec. 3. Creating two copies of a server is, against accidental faults, simple “separation”, which can be increased by further physical isolation between the two. Making the two servers use different software can be seen as further “separation”, against accidental software faults. The different software for the two machines may be intentionally developed in different ways to reduce the chance of common faults: “forced diversity”, through “separation” of the fault-producing processes; and the different ways may intentionally be chosen to have different known strengths and weaknesses [7]. Geographical distance, as a defense against physical causes of failure, is simply added “separation”. Against software faults, it is not even that. Against physical attacks, it can be more than that: for an enemy with limited resources, it would make it unlikely that both servers can be attacked at once, and thus push towards negative correlation of attacks, and thus of the failures they may cause. Against distributed attacks with stolen resources, again it may not even confer “separation”.[2]

This discussion shows again that the meanings of words like “redundancy” and “diversity” are somewhat ambiguous and ill-delimited. We do not advocate more stringent definitions, which would contrast with common usage; rather, in analyzing design solutions one needs to refer to the specific threats and mechanisms employed and how they affect the correlation among failures.

There is currently renewed interest in using diversity for security. Many recent papers invoke “diversity” as an aid for security. Without citing them all, we can identify a few basic categories. Economic factors, with the increasing common dependence on off-the-shelf products, naturally push towards greater application of fault tolerance for all aspects of dependability [14, 15]. A category of proposed designs for intrusion tolerance thus is meant to allow for diverse off-the-shelf applications or platforms to coexist in a system. Another category of proposals stems from the often repeated observation that lack of diversity in the

[2] Similar reasoning applies to non-replication redundancy, e.g. error-detecting/correcting codes, redundant data structures, watchdog, monitor or audit processes.

computers on a network creates the potential for broad propagation of attacks, as demonstrated by several Internet worm attacks. Authors have proposed e.g.: random diversification of compilations to defeat buffer overflow attacks [16]; generating "variants of many OS modules, so some of the variants will be resistant to new, previously unknown attacks" [17]; "randomizing" at installation time the choice of COTS components (among different, equivalent implementations) for forming a system [18]. HACQIT (Hierarchical Adaptive Control of Quality of service for Intrusion Tolerance) [19] uses diverse off-the-shelf applications. The Cactus architecture [20] and the SITAR architecture [21] are meant to support diversity among application modules to enhance survivability; similar ideas are proposed in [22]. Proposals for agent-based distributed intrusion detection [23] cite support of diversity as one of their goals. Last, discussion papers argue the general desirability of diversity; e.g., [24] proposes diversity among network elements, in along all possible dimensions of a design; [25] lists forms of diversity available at different system levels.

Most of these examples can be seen as degrees of the "separation" or "forced diversity" approaches: the hope is that "diversified" subsystems, though all having unknown vulnerabilities, will not succumb to the same attacks. There is no "tailoring" of diversity to threats. But, since general categories of attacks can be identified and the efficacy of defenses varies between them, there is also a role for "tailored diversity". We will return to this after looking at some weaknesses of the current debate about "diversity for security".

5 Applying results from diversity research

The papers we cited demonstrate awareness, in the security community, of diversity as a potentially valuable tool. But, interestingly, none of these papers discusses how to choose among different diverse designs, that use e.g. different architectures or different selections of diverse components for the same architecture, or how to evaluate the effectiveness of the design once selected. The rare statements about these issues are somewhat simplistic, and limited to design factors that a designer can control directly, without consideration of how to evaluate their actual effect. For example:

- "The deployment environment is not susceptible to common-mode failures since ITDOS supports implementation diversity in both language and platform" [26]. This clearly refers to the intentions of the fault-tolerant design rather than its effectiveness;
- "An important factor ... is the *independence* of the methods used, where two methods A and B are independent if compromising A provides no information that makes it easier to compromise B, and vice versa. A simple example of non-independence is when two encryption methods use the same key ... While the independence of encryption methods is difficult to argue rigorously, the risk of methods not being independent is likely to be minimized if the methods are substantially different or if they encrypt data in

different size blocks" [27]. This emphasizes the need for *"separation"* against faults affecting both components or propagating between them, but does not address the other factors that may make common failures too likely.

Choosing among (diverse or non-diverse) solutions presents various difficulties. Practitioners are familiar with some of these, e.g. the difficulty of evaluating even a simple, non-redundant security system. We will discuss here some aspects specific to diversity, hoping that this will contribute to insight for practitioners and to selecting research directions.

An example of application of diversity is that of intrusion detection. Any criterion, and any implemented system, for recognizing hostile activity has incomplete coverage (less than 100% probability of recognizing such activity when it happens): it appears natural to combine multiple criteria, and one design solution is to deploy multiple intrusion detection systems (e.g., advertisements for intrusion detection products claim as self-evident that combining "anomaly-based" with "signature-based" intrusion detectors is desirable). The simplest model for their use would be a 1-out-of-N system: a threat is assessed to be present provided that at least one of the intrusion detection systems (IDSs) detects it. The problem would arise of deciding how effective the combined IDS would be, so as to choose them appropriately. In this simple form of combination, increasing the number of diverse IDSs in the system can only increase their combined coverage, but this may not be a feasible option: each added IDS increases cost (false alarms, ownership costs, run-time overhead).

To choose a single IDS from those available, one would try to rank them by performance, and choose the best one. Using the limited data known about commercial or research systems [28–30], plus one's opinions and anecdotal evidence, a designer will be able to choose the "best" system in view of his constraints.

Suppose now that one is to choose two IDSs to deploy together. Should one simply choose the two "best" IDSs from the previous ranking, supposing that they satisfy one's cost constraints? Not necessarily, since their combined effectiveness will depend on *both* their individual effectiveness *and* the correlation among their failures; any choice we make affects both.

Some help may come from a "tailored diversity" approach. Different IDSs will be effective against different attacks. It would then seem that the correct criterion for choosing is a deterministic coverage criterion: with the help of a comparison of the various IDSs' characteristics as in [31] and a knowledge of which characteristics help to detect which attacks, one would enumerate the attacks "covered" by each tool in isolation, and then by each pair of them in combination. Ranking the sets of attacks covered by each pair, one would then choose the "best" pair of IDSs, as proposed e.g. in [32]. This method, however, neglects the uncertainty on the actual detection of each specific attack. Again, we need to use probabilities. One clearly needs to combine the information about the classes of attacks covered and about how well they are covered.

It is here that some results from previous research on diversity may help (see [3, 4]). A first-cut description of the problem runs as follows. If we choose a *specific* attack x, a certain IDS, A, in the given operational conditions has a probability $\theta_A(x)$ of failing to detect it. Another IDS, B, will have a probability $\theta_B(x)$ of failing to detect it. As the attacks arrive unexpectedly, according to some probability distribution, the probabilities of A and B each missing the next, unknown attack will be weighted averages (over attacks) of the functions $\theta_A(x)$ and $\theta_B(x)$, say, Q_A and Q_B. A designer will try to protect the two IDSs from causes of common failure, e.g. if they monitor a network will try to run them on separate hosts, so that the activity of A will not affect the performance of B, and vice versa. There will be still some common influences on both: e.g., the amount of network traffic to be monitored. To take account of these common environmental conditions, we would include them in the description of the individual attacks. An idealized model, then, will assume that, for a *specific attack* x, the failures of the two IDSs are independent. But they will not be independent in general, for the next, random attack X. A well-known equation from the literature on diversity gives:

$$P\left(\text{A and B both fail to detect} X\right) = \sum_{x \in D} P(x)\theta_A(x)\theta_B(x) = \\ Q_A Q_B + cov_x\left(\theta_A(x), \theta_B(x)\right) \quad (1)$$

where $P(x)$ indicates the probability of the attack x in the environment of use, D is the set of all possible attacks, and *"cov"* designates the covariance of the two functions, roughly an indication of how similar the two "θ" functions are. When high, this indicates that attacks which are likely to be missed by A are also likely to be missed by B. Zero covariance indicates independence. Negative covariance is most desirable. Ideally, any attacks missed by A would be detected by B, and vice versa. While this is unlikely to be achieved, nothing in principle prevents the values of the covariance from being very low, especially if A and B were "tailored" for different methods of attack. The modelling confirms, though, that to evaluate the pair it is not sufficient to evaluate A and B separately. Somehow, one must evaluate either the left hand term of the equation (the effectiveness of the pair as a whole) or the separate terms of the right-hand side, which describe each IDS separately plus, through the covariance, the effect of combining them. In some experiments on software diversity against design faults, the covariance part dwarfed the product that precedes it. A difference when applying equation 1 to IDSs' failures to detect attacks is that "failure" encompasses not only the effects of software bugs, but also the natural limits of the intrusion detection algorithms. Thus, we will expect the terms Q_A, Q_B to be greater than those observed in software diversity experiments; but also, on the other hand, a better chance for the designer of being able to rank the possible pairs of IDSs in terms of covariance, by looking at which cues individual IDSs use and thus which attacks they seem least able to detect. This possibility requires some more discussion.

The functions $\theta_A(x)$ and $\theta_B(x)$ are in practice unknowable: they describe the effectiveness of an IDS with respect to every possible attack episode, specified in minute detail. One can usually estimate (more or less accurately, by combining statistical measures and educated guesses) probabilities of missing attacks by *type* of attack. Given the attack types, C_1, C_2, ..., we would thus have variables $Q_{A|1}$, $Q_{A|2}$, .. indicating the probability of e.g. IDS A failing to detect an attack of category 1, 2, etc. As shown in [33], equation 1 can be rewritten as:

$$\begin{aligned} P\left(\text{A and B both fail to detect} X\right) = \\ \sum_i P\left(X \in C_i\right) P\left(\text{A and B both fail to detect} X | X \in C_i\right) = \\ \sum_i P\left(X \in C_i\right) \left(Q_{A|i} Q_{B|i} + cov_{x \in C_i}\left(\theta_A(x), \theta_B(x)\right)\right) \end{aligned} \tag{2}$$

This gives some practical improvements compared to (1). If we can choose our classification of attacks so that within each type at least one between A and B has practically constant value of its θ function for all the attacks of that type (a simple extreme case being that of deterministically detecting all, or none, of the attacks of one type), the covariance terms will be zero. If we can trust this *conditional independence* within each attack type, the problem is reduced to evaluating A and B separately, albeit in somewhat greater detail (i.e., for each attack type separately) than required if the covariance term were zero in (1). A designer will try to make the sum $\sum_i P(C_i) Q_{A|i} Q_{B|i}$ as small as possible, by choosing A and B so that their respective strengths and weaknesses are complementary (low covariance between the *average* probabilities of missing attacks of each type). One can also write this more modest inequality:

$$\begin{aligned} P\left(\text{A and B both fail to detect} X\right) = \\ \sum_i P\left(X \in C_i\right) P\left(\text{A and B both fail to detect} X | X \in C_i\right) \leq \\ \sum_i P\left(X \in C_i\right) min\left(Q_{A|i}, Q_{B|i}\right) \end{aligned} \tag{3}$$

which, given the designer's preference for IDSs with "complementary" weaknesses, may give low enough upper bounds to demonstrate the advantage of the diverse system over either IDS alone. Of course, upper bounds are not sufficient for an *optimal* choice of a pair of IDSs: for this, it is inevitable to refer to some variation of equation 2, allowing for the large uncertainties on all its parameters.

Environmental conditions (e.g., network load) may influence the θ functions. It will then be advisable to use, in the categorization of attacks, not just qualities like the kind of security weakness they exploit, but also these environmental circumstances, e.g. the type "overflow exploit type X" would be divided into subtypes "..with heavy load" and "..with light load". In mathematical terms, this attempts to reduce the covariance term within each attack type simply by narrowing the range of variation of either θ function.

The important point here is that it is necessary to evaluate IDSs by types of attacks and of conditions of use. In practice, however, in the rare meritorious efforts to report the detection efficacy of IDSs [29, 28], only *average* measures of efficacy are often obtained. However, [30] classified test results by coarse categories of attack (8 categories); finer classifications may be needed. This greater attention in data collection to the variations in IDS efficacy would be useful in any case, irrespective of whether diversity is used, for system designers to gauge the range of effectiveness they can expect depending on variations in the attack population, which of course is under the control of attackers.

6 Conclusions

This is a very initial attempt to highlight important aspects of diversity for security and the need for more of a formal mathematical approach to estimating its effectiveness. Some immediate conclusions from applying earlier models are:

- the idea of "independence" must be treated with care, making sure not to confuse its various meanings;
- in choosing diverse subsystems from a given set, attention is needed to the trade-off between the goodness of the individual subsystems and their "diversity";
- it is important to measure the performance of IDSs by category of attack, rather than for some average mixture of attacks; to some extent, the equations help to combine one's assessments of specific aspects of IDSs into guidance for design.

We would be the first to admit that these results are rough and imprecise; for some readers, they will undoubtedly just confirm common sense. But we have seen in previous research that common sense about diversity is actually very different for different people, and often leads to conclusions that turn out to be demonstrably false. For example, the formal statement and proof of the elusiveness of failure independence has proven counterintuitive (and thus important) in reliability and safety, and will apply to many security scenarios as well.

Important directions for developing these ideas include:

- clarifying the roles of the various forms of uncertainties affecting any prediction, e.g. differentiating the effects of the unknown attack profiles from that of the unknown defects of the defender's system;
- analyzing the complex real-life systems, where redundancy/diversity are deployed in different guises, concurrently, against various threats.

We do not propose these approaches as a way for obtaining precise, demonstrably correct predictions of e.g. how many attacks will go undetected on a particular system. This is beyond the reach of probabilistic methods in most field of dependability, and the more so the fewer statistical data are available.

Much of the chance for security system designers to make well-guided choices depends on better empirical measurements of the actual effectiveness of their various defense methods and components. However, these will never be very accurate. Yet, design decisions should at least be consistent with the information that one does have. Explicit probabilistic modelling seems the only way for ensuring this consistency.

References

1. Randell, B., Dobson, J.E.: Reliability and Security Issues in Distributed Computing Systems. In Proc. 5th IEEE International Symposium Reliability in Distributed Software and Database Systems, Los Angeles (1986) 113-118.
2. Joseph, M.K., Avizienis, A.: A Fault-Tolerant Approach to Computer Viruses. In Proc. 1988 Symposium on Security and Privacy, Oakland, CA (1988) .
3. Littlewood, B., Popov, P., Strigini, L.: Modelling software design diversity - a review. ACM Computing Surveys 33 (2001) 177-208.
4. Littlewood, B.: The impact of diversity upon common mode failures. Reliability Engineering and System Safety 51 (1996) 101-113.
5. Littlewood, B., Brocklehurst, S., Fenton, N.E., Mellor, P., Page, S., Wright, D., Dobson, J.E., McDermid, J.E., Gollmann, D.: Towards operational measures of computer security. Journal of Computer Security 2 (1994) 211-229.
6. Littlewood, B., Popov, P., Strigini, L., Shryane, N.: Modelling the effects of combining diverse software fault removal techniques. IEEE Transactions on Software Engineering SE-26 (2000) 1157-1167.
7. Popov, P., Strigini, L., Romanovsky, A.: Choosing effective methods for design diversity - how to progress from intuition to science. In Proc. SAFECOMP '99, 18th International Conference on Computer Safety, Reliability and Security, Toulouse, France (1999) 272-285.
8. Shamir, A.: How to share a secret. Comm. of the ACM 22 (1979) 612-613.
9. Deswarte, Y., Blain, L., Fabre, J.-C.: Intrusion tolerance in distributed systems. In Proc. IEEE Symp. on Research in Security and Privacy, Oakland, USA (1991) 110-121.
10. Cherry, S.M.: Took a Licking, Kept on Ticking. IEEE Spectrum December (2002).
11. Cherry, S.M.: Striking at the Internet's Heart. IEEE Spectrum December (2001).
12. Madan, B.B., Goseva-Popstojanova, et al : Modeling and Quantification of Security Attributes of Software Systems. In Proc. DSN 2002, International Conference on Dependable Systems and Networks - International Performance and Dependability Symposium, Washington, D.C., USA (2002).
13. Singh, S., Cukier, M., Sanders, W.H.: Probabilistic Validation of an Intrusion-Tolerant Replication System. In Proc. DSN 2003, International Conference on Dependable Systems and Networks - Dependable Computing and Communications Symposium, San Francisco, U.S.A. (2003) 615-624.
14. Popov, P., Strigini, L., Romanovsky, A.: Diversity for off-the-Shelf Components. In Proc. DSN 2000, International Conference on Dependable Systems and Networks - Fast Abstracts supplement, New York, NY, USA (2000) B60-B61.
15. Cowan, C., Pu, C.: Survivability From a Sow's Ear: The Retrofit Security Requirement. In Proc. Information Survivability Workshop - ISW '98, Orlando, USA (1998).
16. Forrest, S., Somayaji, et al: Building Diverse Computer Systems. In Proc. 6th Workshop on Hot Topics in Operating Systems (HotOS-VI), (1997) 67 -72.
17. Cowan, C., Pu, C.: Immunix: Survivability Through Specialization. In Proc. SEI Information Survivability Workshop, San Diego (1997).
18. Casassa Mont, M., Baldwin, A., Beres, Y., Harrison, K., Sadler, M., Shiu, S.: Towards Diversity of COTS Software Applications: Reducing Risks of Widespread Faults and Attacks. Trusted E-Services Laboratory, HP Laboratories Bristol, document HPL-2002-178, June 26 (2002).

19. Reynolds, J., Just, J., Lawson, E., Clough, L., Maglich, R., Levitt, K.: The Design and Implementation of an Intrusion Tolerant System. In Proc. DSN 2002, International Conference on Dependable Systems and Networks, Washington, D.C., USA (2002) 285-292.
20. Hiltunen, M.A., Schlichting, R.D., Ugarte, C.A., Wong, G.T.: Survivability through Customization and Adaptability: The Cactus Approach. In Proc. DARPA Information Survivability Conference and Exposition, (2000).
21. Wang, F., Gong, F., Sargor, C., Goseva-Popstojanova, K., Trivedi, K., Jou, F.: SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services. In Proc. 2001 IEEE Workshop on Information Assurance and Security, West Point, New York, U.S.A (2001).
22. Ellison, R., Fisher, D., Linger, R., Lipson, H., Longstaff, T., Mead, N.: Survivability: Protecting your critical systems. IEEE Internet Computing 3 (1999) 55-63.
23. Dasgupta, D.: Immunity-Based Intrusion Detection System: A General Framework. In Proc. 22nd National Information Systems Security Conference, NISS, Arlington, USA (1999).
24. Zhang, Y., Vin, H., Alvisi, L., Lee, W., Dao, S.K.: Heterogeneous Networking: A New Survivability Paradigm. In Proc. NSPW'01 , 2001 Workshop on new security paradigms, Cloudcroft, New Mexico, USA. (2001) 33-39.
25. Deswarte, Y., Kanoun, K., Laprie, J.-C.: Diversity against Accidental and Deliberate Faults. In Proc. Computer Security, Dependability and Assurance: From Needs to Solutions, York, England and Washington, D.C., USA (1998).
26. Sames, D., Matt et al: Developing a Heterogeneous Intrusion Tolerant CORBA System. In Proc. DSN 2002, International Conference on Dependable Systems and Networks, Washington, D.C., USA (2002).
27. Hiltunen, M.A., Schlichting, R.D., Ugarte, C.A.: Using Redundancy to Increase Survivability. In Proc. Third Information Survivability Workshop (ISW-2000), Boston, Massachusetts, USA (2000).
28. Durst, R., Champion, et al: Testing and Evaluating Computer Intrusion Detection Systems. Comm. of the ACM 42 (1999) 53-61.
29. Maxion, R.A., Tan, K.M.C.: Benchmarking Anomaly-Based Detection Systems. In Proc. DSN 2000, International Conference on Dependable Systems and Networks, New York, New York, USA (2000) 623-630.
30. Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., David McClung, Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., Zissman, M.A.: Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation. In Proc. DARPA Information Survivability Conference and Exposition (DISCEX '00), Hilton Head, South Carolina, U.S.A. (1999) 12-26.
31. Jackson, K.A.: Intrusion detection system (IDS) product survey. Los Alamos National Laboratory, document LA-UR-99-3883, June (1999).
32. Alessandri, D.: Using Rule-Based Activity Descriptions to Evaluate Intrusion-Detection Systems. In Proc. 3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000), Toulouse, France (2000) 183-196.
33. Popov, P., Strigini, L. et al: Estimating Bounds on the Reliability of Diverse Systems. IEEE Transactions on Software Engineering SE-29 (2003) 345-359.
34. Kennedy, C.M., Sloman, A.: Closed Reflective Networks: a Conceptual Framework for Intrusion-Resistant Autonomous Systems. University of Birmingham, School of Computer Science, Technical Report CSR-02-3, February (2002).