# Software Diversity

## A Position statement for the panel *Use of Diversity as a Defense Mechanism*

John M$^{c}$Hugh
Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada
mchugh at cs.dal.ca

For over twenty years, software diversity has been touted in some circles as an effective and cost effective way to improve system reliability. Based on well established practice in the replicated hardware arena, the notion is that, if replica failures are independent, exponential gains in reliability can be obtained for linear increments in cost. Unfortunately in the software case, the "if" is, in general, not true and the gains that can be realized are far from exponential. The kinds of hardware failures against which replication works are distributed independently randomly in the time domain. Software failures are time invariant and distributed in the data domain (with uncertain distributions) so that identical replicas will all fail on the same inputs. Thus, the holy grail of software fault tolerance has been to introduce diversity in such a way that failures are distributed independently in the data domain. These have been largely unsuccessful.
Early researchers assumed that software written by different developers would manifest independent failures. The early experiments by John Knight (later joined by Nancy Leveson) showed that the situation was much more than complex and that "prayer for diversity"(footnote - I coined this term in the early 1980s to cover the case in which the developers were not allowed to communicate and the customer prayed that the results would be sufficiently diverse to provide the necessary gain.) did not work. The Knight and Leveson work demonstrated something more insidious, though it should have been obvious, that errors tended to cluster in "hard" parts of the problem space. This means that correlated failures can occur when multiple programmers make different errors dealing with the same hard case. Shortly thereafter Eckhardt and Lee showed analytically that the reliability gains to be expected from even slightly correlated failures were minimal and this should have been the end of the discussion. In the reliability area, there have been a number of attempts to force diversity, including several that deserve the term "voodoo software engineering." A particularly egregious example was reported at FTCS some years ago and was apparently used in the A320. It involved partitioning the instruction set of the processor used into two subsets, either of which was sufficient for writing the software needed. Since the programmers were writing in two different instruction sets, it was claimed that common mode failures were impossible; QED. The problem of achieving software failure independence through diversity can be paraphrased as follows: Each replica should be correct for most input cases, but when it is not, it must be wrong for cases where the other replicas are correct. This is, arguably, a difficult requirement to fulfill.
More recently, the idea of diversity has resurfaced in security where it has been claimed that the Microsoft/Intel mono culture is a major factor in enabling worm and virus attacks. Appeals are made to the biological arena where diversity (or the lack thereof) are said to be factors in determining whether a particular biological system is robust against disease, insects, and even fire.
The prevalence of vulnerable machines is clearly a major factor in the rapid spread of many worms and viruses, but it is not clear that simply introducing some sort of arbitrary diversity solves the problem. Unlike the reliability case where the consequences of a replica failure are bounded by a voting mechanism, the consequences of a replica failure in a security attack may be unbounded. For example, suppose that we are seeking high level diversity in a web server by using IIS under Windows on an Alpha, Apache under Linux on Intel, and WebSTAR under OS X on a PPC G5. Queries are submitted to all three and the results are voted under some reasonable similarity measure.(footnote - Never mind how improbable it is that the results will be suitably similar or that the data can be kept in sync.) Surely, this configuration should be immune to common mode attacks.
Consider the following scenario. Mallet, the canonical bad guy, using the resources of a national laboratory, develops independent exploits for each server / platform combination. These exploits have the common capability that they attack one platform / server each where they plant a back door for future use and then return the expected result. On the non vulnerable platforms, the attack does not function, but the query simply returns the expected result. Any high level voting mechanism succeeds. The back door is designed to be triggered by the same query on all replicas. After using all the variants of the attack, Mallet has a reasonable level of confidence that all the replicas are compromised and that the common back door is open.

NSPW 2005 Lake Arrowhead CA USA

Although an extreme example, the point to be considered is that diversity comes in a wide variety of flavors and an attacker can often find or force a common vulnerability where none might be expected.

The above example should be taken as illustrating why the appeals to statistics commonly made by the reliability and dependability communities may not be appropriate in the area of security. While it is tempting to view vulnerabilities that can be used for security exploits as just another fault waiting to be discovered and subject to being reached in the fullness of time under normal usage or even test conditions, this is probably not appropriate. In many cases these vulnerabilities lie outside the specified (but not checked or enforced) interface for the program. In most cases, to be effective, the program inputs that reach the vulnerable code are not only well outside the space of "normal" inputs for the program, but must lie within a very specific region of the abnormal space and exhibit a narrowly defined structure. For all practical purposes, the probability of a spontaneous exploitation by a user who is unaware of the vulnerability is zero. Similarly, the probability that the vulnerability manifests itself as a bug in response to normal usage is equally small. Knowledge of the vulnerability by a small community does not change this. The vulnerability can only be exploited if someone who knows about it constructs and uses an appropriate input. If the knowledge is closely held by malicious interests, the likelihood that exploitation will eventually occur is probably one, but the timing depends on the motivations and objectives of the holders. If the vulnerability becomes widely known, the probability of exploitation in the immediate future becomes one as soon as an exploit is developed. Neither the zero or one cases provide any leverage from an analysis viewpoint.

I suspect that most large, complex programs developed using commonly practiced software development techniques using unsafe languages such as "C" have exploitable vulnerabilities and that these are distributed in the potential input space in such a way that they are not removed by the normal test and modify cycle that contributes to reliability growth. From a modeling standpoint, the best that we can do is ascribe a (constant over time) probability of 1 to the existence of exploitable vulnerabilities in such programs. The probability that an exploitable vulnerability will be discovered in a given program over an arbitrary time interval is externalized, depending on the motives of the attackers, their level of skill, where in the code the vulnerability lies, how the program is used, what advantages might accrue to the exploiters, etc. This is unknowable, in general, and appeals to statistics are generally not helpful.