Defence Research and Development Canada
Recherche et développement pour la défense Canada

Towards a reasoning framework using diversity for security

R. Khoury
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2013-220
April 2012

Towards a reasoning framework using diversity for security

R. Khoury
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2013-220
April 2012

Abstract

N-version programming has been shown to be an effective way to increase the reliability of systems. In this technical report, we examine the possibility of extending this approach to address security, rather than reliability concerns. We focus specifically on how to evaluate the efficiency of the use of diversity for security. We show that while several key elements must be taken into account when N-version programming is used for security rather than reliability, it is nonetheless possible to devise a reasoning framework to evaluate the efficiency of this development paradigm in a security context. Furthermore, we present preliminary empirical results indicating that an effective diversity-based intrusion detection scheme is feasible.

Résumé

Des études ont démontré que la programmation en n versions est une méthode efficace pour assurer la fiabilité des systèmes. Dans ce rapport technique, nous examinons la possibilité d'étendre cette approche pour assurer la sécurité des systèmes, plutôt que leur fiabilité. Nous nous concentrons particulièrement sur l'évaluation de l'usage de la diversité à des fins de sécurité. Nous concluons que plusieurs éléments doivent être pris en compte quand la diversité est utilisée dans l'optique de la sécurisation des systèmes, plutôt que dans celle d'en assurer la fiabilité. Néanmoins, il demeure possible de développer un cadre de raisonnement permettant d'évaluer l'efficacité de ce paradigme dans un contexte de sécurisation des logiciels. De plus, les résultats initiaux de nos études empiriques indiquent qu'il est possible d'utiliser la diversité à des fins de sécurité d'une manière efficace.

This page intentionally left blank.

Executive summary

Towards a reasoning framework using diversity for security

R. Khoury; DRDC Valcartier TM 2013-220; Defence Research and Development Canada – Valcartier; April 2012.

Background: N-version programming is a software development paradigm that draws upon the concept of diversity to increase the reliability of software. In this technical report, we investigate the possibility of extending this approach to address security, rather than reliability concerns. We focus specifically on the way to evaluate the efficiency of using diversity for security. Furthermore, we conduct an empirical experiment to determine if diversity is an effective tool for intrusion detection.

Principal results: We find that while several key differences must be taken into account when N-version programming is used for security rather than reliability, it is nonetheless possible to devise a reasoning framework to evaluate the efficiency of this development paradigm in a security context. Furthermore, we present preliminary empirical results indicating that an effective diversity-based intrusion detection scheme is feasible.

Significance of results: Our empirical results argue strongly in favour of the use of diversity towards the goal of securing software against intrusions. Furthermore, our proposed reasoning framework enables system developers to evaluate the effectiveness of different system architectures containing diversity, and select the most effective solution for their specific situation.

Future work: The results of the experiment presented in section 4 should be contrasted with those of a similar experiment that uses invalid or simulated attack traces rather than normal traces. This will allow us to use the reasoning framework exposed in the previous sections in order to evaluate objectively the efficiency of the approach. Performing other experiments will allow us to answer more specific questions about the most effective manner to introduce diversity in software. For instance, by replicating the experiment in a system in which diversity is introduced at the operating system layer or at the hardware layer, we can gain knowledge about the relative benefits of introducing diversity in different components of a system. Likewise, while we experimented with execution traces abstracted into system call traces, experimenting with other abstractions will allow us to determine which abstraction provides the highest intrusion detection rate while maintaining a low false positive rate.

Sommaire

Towards a reasoning framework using diversity for security

R. Khoury ; DRDC Valcartier TM 2013-220 ; Recherche et développement pour la défense Canada – Valcartier; avril 2012.

Introduction ou contexte : La programmation en N-versions (N-version programming) est un paradigme de programmation qui s'appuie sur le concept de diversité pour assurer la fiabilité des logiciels. Dans ce rapport technique, nous examinons la possibilité d'étendre cette approche pour adresser le problème connexe de la sécurisation des logiciels. Nous nous concentrons particulièrement sur la question d'évaluer l'efficacité de l'utilisation de la diversité à des fins de sécurité. De plus, nous effectuons une étude empirique afin de déterminer la faisabilité de l'usage de la diversité comme outil de sécurisation des logiciels.

Résultats : Nous concluons que bien que plusieurs éléments doivent être pris en compte quand la diversité est utilisée dans un contexte de sécurité plutôt que dans un contexte de fiabilité, il demeure possible d'élaborer un cadre de raisonnement permettant d'évaluer l'efficacité d'une telle approche. De plus, les résultats de notre étude empirique démontrent la faisabilité d'une telle approche.

Importance : Nos résultats empiriques militent en faveur de l'usage de la diversité comme outil pour assurer la sécurisation des logiciels. De plus, le cadre de raisonnement que nous proposons permet à des développeurs d'évaluer l'efficacité de différentes architectures qui incorporent la diversité et de sélectionner la plus appropriée à chaque situation.

Perspectives : Les résultats de l'expérience rapportée à la section 4 devraient être contrastés avec ceux d'une expérience similaire qui utilise des traces invalides ou des traces d'attaque. Ceci nous permettra d'utiliser le cadre de raisonnement proposé dans les sections précédentes afin d'évaluer l'efficacité de l'approche. Nous pourrons ensuite répondre à des questions plus spécifiques sur l'usage de la diversité pour la sécurisation des logiciels par l'entremise d'autres expériences. Par exemple, en répliquant l'expérience dans un contexte où la diversité est introduite à un autre niveau du système d'exploitation ou au niveau des composantes physiques, il serait possible de déterminer à quel niveau il est plus bénéfique d'introduire la diversité. De la même manière, nous avons abstrait les traces d'exécutions en traces d'appels systèmes, mais des expériences subséquentes pourraient nous indiquer si d'autres abstractions permettent une détection plus fine des intrusions, avec un plus faible taux de faux positifs.

Table of contents

List of figures

Acknowledgement

The author wishes to extend thanks to Mario Couture, Dr. Abdelwahab Hamou-Lhadj, Robert Charpentier and Dr. Richard Khoury.

This page intentionally left blank.

1 Introduction

The fields of software reliability and security are closely related and several methods simultaneously address both concerns without distinguishing between a malicious and an inadvertent failure. It is thus normal to ask if the N-version programming paradigm, which was developed to address reliability concerns, can likewise be deployed in a security context. We believe this is possible, but that several key elements must be taken into account when diversity is introduced in an architecture for security purposes rather than to increase reliability.

N-version programming [1] is a software development paradigm that draws upon the concept of diversity to increase the reliability of software. The guiding principle of this approach is to produce several distinct versions of a given software, and execute them in parallel with the same inputs. A discrepancy between the outputs of the various instances is an indication that at least one instance has malfunctioned. In that case, a single output value is chosen from the outputs of each program instance by majority voting. The intuition behind this programming paradigm is that while it may be impossible to produce a single flawless instance of any complex system, multiple instances of this system would normally exhibit different faults.

A key concept in the design of N-version architectures is *failure independence.* Informally, this property describes the behavior of a system for which the occurrence of a failure in one instance for a given input value does not provide any information in regard to the probability of failure of another instance for the same input value. It is from the assumption of failure independence that we derive the hypothesis that the probability of *coincident failure* (i.e. two instances failing on the same input) is very small and that gains in reliability can be obtained through the use of N-version programming.

Researchers in security have also shown a great deal of interest in diversity, though not in the context of an N-version architecture. Instead, research in computer security proceeds from the assumption that if the program instance of each user differed from that of every other user, an attack cannot easily be carried over from one system to the next. The attacker will thus be forced to tailor each attack to the system he wishes to compromise. Furthermore, the added uncertainty about the target system increases the cost of the attack [2].

In this study, we propose to join together those two strands of research, and elaborate a N-version architecture for security. The main intuition underlying such an architecture is that, since attacks must be tailored to each program instance, if several program instances are executed in parallel as part of an N-version architecture and an input contains an attack vector, it is likely that the attack will succeed only on some of the several program instances. This in turn will cause the executions of the affected and unaffected instances to diverge observably from one another. Such a divergence can then serve as the basis for intrusion detection and reaction.

In this technical report, we show that such an approach can be used effectively to increase the security of systems and develop a general framework to study its effectiveness. Furthermore, we contrast the use of diversity for security purposes to that of diversity for reliability and highlight a number of key differences that must be taken into consideration in the former case.

The remainder of this report is organized as follows. In Section 2 we contrast the proposed use of N-version programming for security with its more common use for reliability and highlight the relevant differences between the two approaches. Section 3 reviews the literature on both topics while section 4 presents the empirical tests conducted with diversity and approach the results. Concluding remarks and perspectives for future work are presented in Section 5.

2 Diversity for security vs Diversity for Reliability

2.1 General Framework

We propose the following framework to study and reflect about the use of diversity for security. We start with a population $\mathcal{P}$ of programs, that represents a hypothetical set of all possible programs able to solve a given problem. We let π range over possible programs.

$\mathcal{P} = \{\pi_1, \pi_2, \pi_3 ...\}$

These programs take their input from a set of possible input values $\mathcal{X}$. Each input represents the entire interaction a user has with a given program during a session. We let x range over the possible input values.

$\mathcal{X} = \{x_1, x_2, x_3 ...\}$

Some inputs may be *malicious*, meaning that they hide exploits that bring the system in a state that violates the security policy. For instance, an input field may contain data triggering a buffer overflow and allowing code injection to occur. Such input values are said to be *invalid*. Normal input values that do not contain an attack, are said to be *valid*. We write $\mathcal{X}_v$ for the set of valid inputs and $\mathcal{X}_i$ for that of invalid inputs. Note that $\mathcal{X} = \mathcal{X}_v \cup \mathcal{X}_i$ and that $\mathcal{X}_v$ and $\mathcal{X}_i$ are disjoint.

Let $Q()$ stand for the usage distribution of the values of $\mathcal{X}$. This distribution naturally affects only valid executions, since an attacker can alter the distribution of inputs by repeatedly inputting the values he or she needs in order to alter the system.

In the context of their study of diversity for security, Littlewood et al.[3] propose a score function $\upsilon : (\mathcal{P} \times \mathcal{X}) \rightarrow \{1, 0\}$, that indicates whether or not the execution of a given program for a given value fails. The occurrence of such a failure could be immediately observed by inspecting the program's output (for example if the program failed to produce a return value), or discovered by contrasting this output with that of another instance. The score function υ thus serves as basis for evaluating how reliable a given program is and a given diverse architecture containing this program.

However, when the focus is on security, the program's output alone does not provide sufficient information for a meaningful evaluation of the validity of the input. It is entirely possible for an intruder to alter the execution in such a way as to violate the security policy while keeping the output unchanged. Indeed, an attacker who wishes to remain undetected would favor such a course of action. It follows that a diversity-based framework whose focus is security rather than reliability, will necessarily rely on a complete trace of the program's execution, rather than simply on the output, as the basis for its evaluation of the input. This is the first main difference between using diversity for reliability and using it for security.

Difference 1: *Diversity for reliability is implemented by comparing the outputs of multiple instances. When diversity is used for security purposes however, it is necessary to examine execution traces.*

An execution trace is a sequence of atomic actions, performed by the target program during its execution and recorded by a reference monitor. It can hypothetically contain every instruction performed by the target program, or be focused on a subset of security-relevant actions tailored to the security policy of interest or to a specific resource whose security we seek to optimize. Let Σ stand for the set of all possible execution traces and let σ range over traces.

The trace function $\nu : (\mathcal{P} \times \mathcal{X}) \rightarrow \Sigma$ thus replaces the score function of Littlewood et al. This function is given as :

$$\nu(\pi, x) = \sigma \text{ where } \sigma \text{ represents the trace of program } \pi \text{ on input } x. \quad (1)$$

Once the system is executed on multiple instances, it is necessary to contrast the execution traces in order to detect a possible intrusion. Let $\sigma_1 = \nu(\pi_1, x)$ and $\sigma_2 = \nu(\pi_2, x)$ be the execution traces of programs π_1 and π_2 respectively over the same input value x. Let the correlation function $corr(\sigma_1, \sigma_2)$ stand for the degree of similarity observed between these two executions. This correlation is expressed by way of a value between 0 and 1, where 1 indicates identical sequences, and 0 identifies sequences that seem completely unrelated. Several metrics could be used to compute this value. A natural choice is the Levenshtein distance [4], a measure of the number of insertions, deletions and replacements needed to turn one sequence into the other. However, other metrics specifically designed for the problem of detecting divergence between program executions could also be considered.

$$\mathrm{corr}(\sigma_1, \sigma_2) = \text{ The degree of similarity between executions } \sigma_1 \text{ and } \sigma_2. \tag{2}$$

The central idea that underlies the use of diversity as a defence mechanism is that a given attack may succeed on one system but fail on another. This in turn will lead to an observable divergence in the execution traces, allowing the attack to be detected. As discussed above, diversity can be introduced at various levels, such as memory layout [5], instruction set [6] or by using two distinct implementations of the same software or operating system. The success of the approach rests on the capacity to develop systems that are sufficiently different so that most attacks cannot succeed on multiple instances. It follows that while developers building a N-version architecture with the goal of increasing reliability must focus on minimizing the occurrence of coincident failure, those building such architectures for security purposes should seek to maximize the dissimilarity of internal behavior.

Difference 2: *In the context of diversity for reliability, the design of an N-version architecture must minimize the occurrence of coincident failure. However, if the object is security, the design should promote dissimilarity of behavior.*

This second difference raises several interesting questions related to the way to maximize the divergence between systems while maintaining common functionalities between instances, as well as the way to simultaneously update both instances so as to preserve their behavioral equivalence.

Pairs of systems naturally differ as to how much similarity they exhibit while executing normally (i.e. over valid inputs). However, a baseline can be established by examining a sufficiently large and representative sample of executions. This yields a distribution θ as follows:

$$\theta(\pi_1, \pi_2) = \sum_i \mathrm{corr}(\nu(\pi_1, x_i), \nu(\pi_2, x_i))Q(x_i) \tag{3}$$

In effect, this distribution expresses the likelihood that when executing a given input, two executions will differ by a given amount. Note that this distribution is only computed for valid executions. We expect that an invalid execution for which the attack succeeds on one instance only, will contrast with a corresponding valid execution by a higher than average amount, but at the present time this can only be a conjecture.

Let $corr(\sigma_1, \sigma_2)$ be the level of similarity existing between two trace executions σ_1 and σ_2. Our goal is to contrast the observed $corr(\sigma_1, \sigma_2)$ with the known value of $\theta(\pi_1, \pi_2)$ of the program that produced σ_1 and σ_2, as to attempt to determine if the input value x hides an attack. Were we in possession of statistical data about the relative distribution of valid and invalid inputs, as well as of data relating to the expected level of correlation between executions of the target programs over invalid inputs, a statistical analysis could be performed. Such an analysis could return a probability ϕ indicating that x is malicious with a certain degree of confidence. However, as discussed above, it is not meaningful to compute a distribution of invalid inputs when reasoning about the possible behavior of a malicious attacker capable of altering the systems's inputs. Furthermore, while statistical data could be gathered about systems behavior on malicious inputs by simulating their execution using test cases of known attacks, such data may not necessarily be generalized to new or unknown attacks. Of particular interest are zero-day exploits that use unknown vulnerabilities of software.

We propose instead a possibilistic approach. Possibility theory [7], is an alternative to probability theory to reason about uncertainty. Informally, a possibility is a value between 0 and 1 that describes the ease with which an event will occur, or will belong to a set, as opposed to the likelihood that it will occur, which is expressed as a probability.

A possibilistic analysis would thus indicate how “normal" the level of observed correlation is and how unusual it would be for a valid input to result in pair of executions exhibiting this level of correlation. This information is captured by a possibility function *pos* whose domain is the range of possible correlation values and whose image is a possibility value in the interval $[0, 1]$. Dubois et al. [8] show how a possibility function reflecting this value can be computed directly from the probability density function. The technique they propose can thus be used to derive a possibility distribution for correlation values. Formally, the possibility of a correlation u occurring is

$$pos(u) = \int_0^u \theta(y)dy + \int_{f(u)}^1 \theta(y)dy \tag{4}$$

where $f : [a, u_0] \rightarrow [u_0, b]$ is a function defined s.t. $f(u) = max\{y|\theta(y) \leq \theta(u)\}$ with the interval $[a, b]$ being the support of θ (possibility $[0, 1]$) and u_0 being its modal value[1].

Once a possibilistic value has been assigned to the correlation between two sequences, action can be taken based on the environment-specific tradeoff between security and functionality.

One way to control this tradeoff is to state a threshold α, a level of divergence below which any pair of executions is deemed suspect.

[1] This solution is thus only applicable to the cases where θ is continuous and monomodal, but it is reasonable to think that this will usually be the case of any correlation probability density function.

$$\vartheta = \begin{cases} 1, & \text{if } \phi \geq \alpha; \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The choice of the threshold value α will determine the sensitivity of the detection. However, in the absence of data about the level of divergence observable in invalid execution it is not possible to give a numerical value to our confidence in this judgement.

Once the attack is detected, the system administrator can react, usually by terminating the execution. In this distinction lies another consequential difference between using diversity for reliability and using diversity for security: in the first case, the goal is to maximize the number of input values for which a correct service is provided, whereas in the latter, the goal is to weed out and deny service if the input value betrays a malicious intent on the part of its originator. While information from the unaffected instance may be used to recover from the attack, it is undesirable to provide service to a malicious user since doing so betrays information about the system.

Difference 3: *When diversity is implemented with the goal of increasing reliability, the object is to maximize the number of inputs for which a service is provided. In the case of diversity for security, we seek to weed out invalid inputs.*

This is more than a simple difference in the reaction to the discovery of a vulnerability and translates into profound changes as to how an architecture must be evaluated. In particular, it poses a new risk which does not exist when diversity is implemented for reliability: that of the occurrence of false positives when two valid executions diverge to the point that the monitor wrongly marks one of them as being invalid.

Difference 4: *The fact that some inputs will not be answered, coupled with the imperfection present in any security architecture, leads to the occurrence of false positives.*

The likelihood that a pair of executions will be mistakenly marked as malicious is a factor of the decision threshold α used to rule out executions and of the distribution θ. We write $fp(\alpha, \theta)$ for the probability of false positives occurring if the threshold is a set of α and the similarity distribution between valid executions is θ.

This risk is expressed by the following equation:

$$fp(\alpha, \theta) = \int_0^y \theta(u)du \tag{6}$$

where y is the maximal value for which $pos(y) \leq \alpha$.

The occurrence of false positives can at first sight be thought of as analogous to the occurrence of coincident failures in the diversity for reliability context and could thus presumably be studied using the same analytical tools that have already been developed for the latter case. This analogy does however obscure two critical differences. First, coincident failures are the result of *unwanted* commonalties between different instances, and various strategies are employed to minimize and eliminate these commonalities (see for e.g. [9, 10]). False-positives, however, result from the *desired* divergence between instances and we believe that, in general, the approach grows stronger as these differences increase. Future research should determine if there exists an optimal amount of divergence that provides the best ratio of attack detection to false-positives for a given threshold. In the meantime, abstraction and correlation algorithms should be developed to discern the divergence between executions associated with successful intrusion from those which occur naturally because of differences in the underlying programs.

Secondly, the modeling of false positives also differs from that of coincident failures in that the latter relies upon a distribution over all inputs to assess the rate of coincident failures and contrasts it against that of faults which are successfully tolerated by the N-version architecture. However, while a distribution of *valid* inputs can be constructed and the rate of false positives estimated from it, we can never hope to compute a distribution that includes invalid inputs, since the occurrence of invalid inputs implies the presence of an attacker capable of querying the system with inputs of his choice calibrated for the purpose of his attack. The best we can do in this case is thus to estimate the rate of occurrence of false positives of the system when it is not under attack.

It does seem intuitive that as instances grow more different, so does the benefit of using diversity for security. Indeed, it is more likely that an attack will succeed on only one of two instances if they are very different from one another than if they are alike. However, as executions grow more divergent it will also become more difficult to identify similarities and correlations between valid executions. This in turn could lead to an increase in the rate of false positives. It follows that unless the increase in divergence between instances is matched by a corresponding increase in the sophistication of the correlation algorithm, benefits by increasing the divergence between instances will be offset by an increase in the number of false positives

2.2 Multiple Instances

Research on diversity for reliability shows that the overall reliability of the system can be improved (up to a point) by increasing the number of instances present in a N-version architecture (i.e. by increasing the value of N) [11]. This is a strategy designed to cope with the unavoidable presence of coincident failures: even if two or more instances fail on the same input, other instances running concurrently may succeed. Such multi-versions architectures take two forms: one-out-of-N systems, which succeeds if at least one instance succeeds, and majority voting systems (or m-out-of-N systems, with $m = \lceil N/2 \rceil$) that succeeds if a majority of the composing instances succeed.

The previous idea can also be carried over to the context of diversity for security. Running multiple instances will increase the odds that a malice will be detected by making it harder on the attacker to devise an input that can compromise every instance simultaneously. A final key difference between diversity for reliability and diversity for security is that, in the latter case, rather than a rigid choice between one-out-of-N or majority voting a diverse architecture can be devised for any $m < N$, indicating that this attack is deemed to have occurred if at least m instances diverge from the others. As the number of instances deviating from the others increases, the input can be treated as malicious with an increasing level of confidence.

Difference 5: *One-out-of-N and majority voting are no longer the only possible voting paradigms. Instead, a threshold dictating how many instances can deviate from the others before the input is treated as malicious must be chosen in such a way as to balance security concerns with the need to minimize false-positives.*

However, comparing several instances raises a number of difficulties. In particular, the system becomes subject to a variation of the consistent comparison problem raised in [12]. Consider a system with three instances, π_1, π_2 and π_3 which, for a given input value x, produce three traces σ_1, σ_2 and σ_3 respectively. Three pairs of comparisons are possible between these three sequences, namely σ_1 with σ_2, σ_1 with σ_3 and σ_2 with σ_3. Let ρ_1, ρ_2 and ρ_3 be the outputs of these three comparisons and let ϑ be the threshold by which we consider that an instance has diverged from the other (and thus that the input is malicious). It may become possible that $|\rho_1 - \rho_2| < \vartheta$, $|\rho_2 - \rho_3| < \vartheta$ but that $|\rho_1 - \rho_3| > \vartheta$.

2.3 System Health and Self Monitoring Execution

In addition to contrasting the various executions between themselves to detect a violation, each execution can be contrasted with a model of the system's desired behavior to detect if the ongoing execution violates the system's security property. This would have several benefits: first, it would reduce the number of false positives by giving an indication as to whether or not a deviation observed between two execution rallies does correspond to a violation of the security policy. Secondly, it would indicate which of two diverging instances is the one for which the attack has succeeded, thus allowing us to isolate the compromised system and use the healthy one for recovery.

The execution monitoring, like the correlation analysis, would return a probability that the execution under consideration is malicious. This is given by the function $eval : \Sigma \rightarrow [0, 1]$.

$$eval(\sigma) = p \text{ the probability that } \sigma \text{ is malicious.} \tag{7}$$

The evaluation given by *eval* is then contrasted with that of the correlation analysis. We can state with greater confidence that two diverging executions hide an attack if they not only diverge, but if they also have a high probability of maliciousness according to *eval*. Once again, the decision as to whether or not a specific execution is to be treated as malicious will be based on a tradeoff between security concerns and the desire to minimize the rate of false positives.

In the presence of a self-diagnostic for each execution sequence, the equations for detecting an attack and to compute the rate of false positives can be stated in a variety of ways, depending on how much weight is given to each type of judgement. An elegant solution is to merge the two judgements of the *eval* functions into a single value that gives the probability that at least one of the two instances is malicious (according to the self-diagnostic) and then adjust the threshold according to this value. The detection should be more sensitive to divergence between the two instances if the self-diagnostic raises alarms, and conversely more tolerant if no abnormal behavior is detected in either instance.

3 Related Works

The N-programming development paradigm [1], also called design diversity, grew from the longstanding practice of using multiple redundant components to increase the reliability of safety-critical hardware. This practice has a rich literature dating back to the late 70s, that includes various experiments conducted in academic settings to evaluate the feasibility and efficiency of the approach as well as theoretical enquiries aimed at modeling and reasoning about the behavior of N-version systems.

The latter studies begin with Eckhardt et al. [13], who proposed an initial model to study the impact of coincident failures on the effectiveness of using diversity for security. The authors modeled the occurrence of such failures by an intensity function that represents the propensity of programmers to introduce faults in such a way that failure does not occur independently on some inputs. Building upon their work, Littlewood et al. [3] proposed an alternative model, that includes an important dimension of methodology, to model the impact of different development strategies on the distribution of coincident failures in software. Both models are contrasted and discussed in [14]. A final approach is proposed by Partridge et al. [15] to model the distinction between the different ways several instances may fail, even if they fail over the same inputs.

The question of using N-version programming for security, rather than for reliability, was raised in a number of studies. Littlewood et al. examined the question in [16]. In [17], Bessani, et al. argued in favor of using diversity for security on the basis of the recorded distribution of vulnerabilities in several operating systems. In [18], the various layers where diversity can be inserted are examined from the perspective of maximizing security. The use of design diversity to protect against computer viruses were examined in [19]. The use of diversity for security, termed automated diversity, was first suggested by Forrest et al. in [2], where it was observed that the homogeneity of computer systems constitutes an important vulnerability. Drawing on an analogy to biological systems, Forrest et al. argued that the robustness of systems could be improved if the program instance used by each user differed slightly from that of every other user. As a case study, they proposed a method for stack layout randomization and showed that it is effective at disrupting a buffer overflow attack.

Following on this line of enquiry, several studies have proposed introducing diversity at different layers of software systems. These include the instruction set [6, 20, 21], address space [5, 22], data space [23], and system calls [24]. A survey of these techniques is provided in [25].

4 Experimental Results

In the preceding sections, we have shown how diversity could be used to ensure intrusion detection. In these experiments this section, we present the results of initial experiments to implement these ideas. In two servers running the same OS but different COTS html servers execute the same requests in parallel. The output of both servers is then contrasted to detect a higher than usual divergence between their behaviors. The intuition behind this approach is that it is unlikely that an attacker will compromise both systems simultaneously. Furthermore, even if that were the case, the attack would reveal itself differently in the two systems. Thus, if an attack is present in one of the two executions, this fact will be revealed by a higher than usual divergence between the observed behaviors of the two systems.

The first objective of the experiment is to observe and quantify such divergences, and then use it as basis to detect intrusions. A second objective is to be able to "translate" between systems, in much the same way that translation can be performed automatically between natural languages. An intrusion or abnormal behavior would then be seen as a translation error.

The experiment was conducted using two COTS HTML servers that were executed in parallel on two identical machines, running the same OS. This allows that diversity be introduced only at the software layer. Both systems were simultaneously fed the same set of inputs, which consisted in 208 requests for locally hosted html pages. The execution of both systems was monitored using the LTTNg monitoring framework.

LTTNg is able to capture a wide variety of information about the ongoing execution. In order to make our initial investigation more tractable, we chose to abstract the execution traces into traces of system calls. This abstraction is in line with other similar studies on intrusion detection, that also focused on system call traces [26]. In these studies execution trace was on average 40 system call long.

In what follows, we will refer to the two systems (hardware, OS and server) as A and B. Since each request is executed simultaneously on both systems, the data can naturally be aggregated into pairs (a, b) of sequences, where a is a sequence from A, b is a sequence from B and both a and b process the same input. It can thus be stated that a is the corresponding trace to b, and likewise that b is the corresponding trace to a. The data was further partitioned into a training set of 150 pairs and a test set of 58 pairs.

For each pair (a, b) in the test set, we identify the sequence b in the training set that exhibits the lowest edit distance to b, and compare the corresponding sequence a with a. A high value for the difference between the edit distance of a and a and that of the distance between b and b', can be taken as an indication of an attack. We say that the pair (a, b) is a match to the pair (a', b').

Initial results are shown in figure 1. These results are very encouraging. As can be seen in the figure, for 53 out of the 58 test sequences the difference between the edit distance of the observed executions and those of their matches in the training set was 0. Three other pairs of sequences exhibited a minimal value for the difference between their edit distance and that of their match i.e., a distance of 1 or 2. Only two sequences were outliers, exhibiting a distance of 9 and 11.

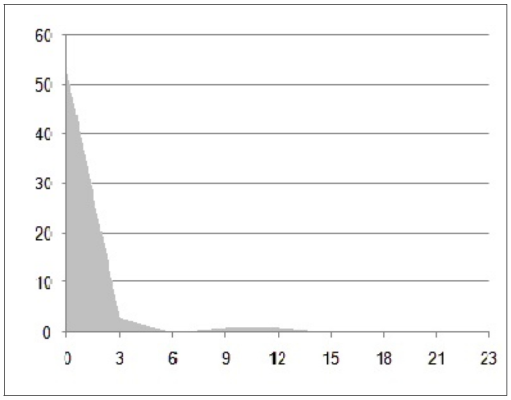

Figure 1: *Experimental results*

These results indicate that whenever the execution is valid, the proposed intrusion detection method can successfully detect an attack (i.e., the risk of false negatives is not prohibitively high).

To be sure, our entire data set is made up of sequences that repeat the same operation, namely accessing an html page upon request. These sequences thus exhibit an high degree of similarity and the promising results reported earlier may have simply reflected this similarity. Put differently, it may be thought that the high correlation between the edit distances of matched pairs of sequences reported above does not result from the successful matching of executions but rather from the fact that all executions were alike. To address the issue, we calculated for each pair of sequence in the test data set, we calculated the average difference in edit distance between the sequences forming this pair and each other pair of sequences in the training set. The results are shown in figure 2.

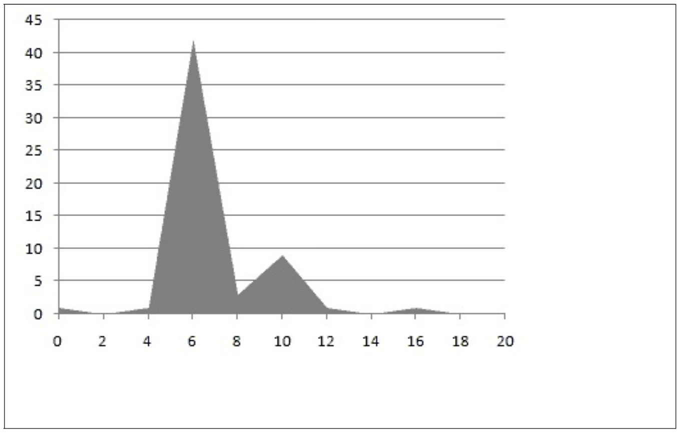

Figure 2: *Experimental results*

This figure shows that the average distance is usually between 6 and 10, much higher than the results given in figure 1. Indeed, in all cases, the difference value computed after selecting a match for a test pair was lower than the average for that pair.

The experiment presented in this section must be extended in several directions. The next step is to test the approach with attack traces, and determine the efficiency of the approach using the reasoning framework proposed in the previous sections. For lack of time, we were unable to complete this phase of this experiment. Nonetheless, a survey of the relevant literature indicates that in most cases, an intrusion would incur a greater amount of disturbance in the sequence in system calls than the level (often nil) of the difference we observed between the sequences of each matched pair. It follows that the approach proposed in this section will likely be able to detect intrusions effectively with a minimal false positive rate.

In addition, the execution sequences we have used are of limited length, averaging 40 system calls each, and every sequence captures the behavior of the system while it is executing the same high-level operation namely answering an html request. Generalizing the approach to the behavior of a real system that is operating continuously and executing multiple threads simultaneously, poses several interesting challenges particularly in regards to the way to untangled the multiple concurrent higher-level behaviors. Alternatively, it would also be interesting to use a finer abstraction than a sequence of system calls. If the intrusion detection mechanism has access to more information about the ongoing execution, it may be able to catch more intrusion, but at the risk of a higher rate of false positives.

5 Conclusions and Perspectives for Future Work

This report proposes a new approach to computer security and host-based intrusion detection, namely N-variant programming for security. In this regard, we combine the complementary fields of design diversity, which is traditionally used for the purposes of increasing the reliability of systems, and automated diversity, which is focused on security.

Several questions must still be answered before N-variant programming can be effectively used for security purposes.

- As discussed above, the central assumption that underlies the use of diversity for security is that if two systems are sufficiently different, attacks targeted at one system cannot be carried over to the other by using the same input values. One of the most pressing topics for further investigation is to test the validity of this hypothesis. In this context, it is important to recall that researchers in the field of diversity for security had assumed that independent development was a sufficient condition to achieve failure independence, until research by [27] showed this was not necessarily the case.
- A second assumption often made in this context is that the more different two systems are, the more likely it is that an attack will succeed on one instance but fail on the other. If this is the case, it becomes necessary to make a difficult compromise between risking a higher rate of false positives by increasing the amount of diversity between the program instances, or risking a lower detection rate by using more similar systems. Answering this question may lead to the development of new metrics to measure the level of divergence between two systems.
- A related problem lies in determining if there are areas of the system for which a greater benefit is derived by introducing diversity. It may happen that introducing diversity in a few key aspects of a system is sufficient to produce two instances for which few attacks can simulta-

neously succeed. Since the cost of building diverse instances may be prohibitive, targeting the introduction of diversity to a few key subsystems can allow a more cost-effective enforcement.

- Another possible avenue of future research is to use multiple correlations between instances. Each correlation would be based on its own trace abstraction, and focussed on preventing a specific class of attacks or on protecting a specific key resource. This opens up the possibility of feedback oriented analysis, where the detection of a possible malice in one of the sequences triggers more scrutiny for all other traces before a final decision can be made about the validity of the trace under observation.
- A final ongoing challenge for future research lies in developing trace analysis and trace abstraction tools that are sufficiently refined to distinguish an attack occurring in a given execution but not in another, from the normal divergence present in a pair of homologous execution traces that are simultaneously executing on diverse systems.

References

[1] Avizienis, A. (1985), The N-Version Approach to Fault-Tolerant Software, *IEEE Trans. Softw. Eng.*, 11, 1491–1501.

[2] Forrest, S., Somayaji, A., and Ackley, D. H. (1997), Building Diverse Computer Systems, In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, pp. 67–72, IEEE Computer Society Press.

[3] Littlewood, B. and Miller, D. R. (1989), Conceptual Modeling of Coincident Failures in Multiversion Software, *IEEE Trans. Software Eng.*, 15(12), 1596–1614.

[4] Levenshtein, V. I. (1965), Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Doklady Akademii Nauk SSSR*, 163(4), 845–848. Original in Russian – translation in Soviet Physics Doklady 10(8):707-710, 1966.

[5] Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., and Boneh, D. (2004), On the Effectiveness of Address-Space Randomization, In Pfitzmann, Birgit and Liu, Peng, (Eds.), *Proceedings of CCS 2004*, pp. 298–307, ACM Press.

[6] Kc, G. S. (2003), Countering Code-Injection Attacks With Instruction-Set Randomization, In *In Proceedings of the ACM Computer and Communications Security (CCS) Conference*, pp. 272–280, ACM Press.

[7] Zadeh, L. A. (1999), Fuzzy sets as a basis for a theory of possibility, *Fuzzy Sets Syst.*, 100, 9–34.

[8] Dubois, Didier, Prade, Henri, and Sandri, Sandra (1993), On Possibility/Probability Transformations, In *Proceedings of Fourth IFSA Conference*, pp. 103–112, Kluwer Academic Publ.

[9] Avizienis, A., Lyu, M. R., and Schutz, W. (1988), In search of effective diversity: A six-language study of fault-tolerant flight control software, In *Proceedings the IEEE Eighteenth Annual International Symposium on Fault-Tolerant Computing (FTCS-18)*, pp. 15–22.

[10] Lyu, M. R. and Avizienis, A. (1991), Assuring Design Diversity in N-Version Software: A Design Paradigm for N-Version Programming, *Dependable Computing and Fault-Tolerant Systems*, 6, 197–218.

[11] Eckhardt, D. E. and Lee, L. D. (1985), A Theoretical Basis for the Analysis of Redundant Software Subject to Coincident Errors, (Technical Report NASA-TM-8636919850015006) National Aeronautics and Space Administration (NASA).

[12] Brilliant, S. S., Knight, J. C., and Leveson, N. G. (1989), The consistent comparison problem in N-Version software, *IEEE Transactions on Software Engineering*, 15, 1481–1485.

[13] Eckhardt, D.E. and Lee, L.D. (1985), A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors, *IEEE Transactions on Software Engineering*, 11, 1511–1517.

[14] Littlewood, B., L., P. Popov, and Strigini (2001), Modeling software design diversity: a review, *ACM Comput. Surv.*, 33, 177–208.

[15] Partridge, D. and Krzanowski, W. (1997), Distinct Failure Diversity in Multiversion Software, Technical Report University of Exeter, U.K.

[16] Littlewood, B. and Strigini, L. (2004), Redundancy and Diversity in Security, In *Computer Security Ű ESORICS 2004, 9th European Symposium on Research Computer Security, LNCS 3193*, pp. 423–438, Springer.

[17] Bessani, A. N., Obelheiro, R. R., Sousa, P., and Gashi, I. (2008), On the Effects of Diversity on Intrusion Tolerance, (DI/FCUL TR 08–30) Department of Informatics, University of Lisbon.

[18] Deswarte, Y., Kanoun, K., and Laprie, J.-C. (1998), Diversity against Accidental and Deliberate Faults, In *Computer Security, Dependability, and Assurance: From Needs to Solutions*, pp. 171–181, IEEE Press.

[19] Joseph, M. K. and Avižienis, A. (1988), A fault tolerance approach to computer viruses, In *Proceedings of the 1988 IEEE conference*

on Security and privacy, SP'88, pp. 52–58, Washington, DC, USA: IEEE Computer Society.

[20] Barrantes, E.G. and Forrest, S. (2006), Increasing Communications Security through Protocol Parameter Diversity, In *In Proceedings of the XXXII Latin-American Conference on Informatics (CLEI 2006).*

[21] Keromytis, A. D. (2009), Randomized Instruction Sets and Runtime Environments Past Research and Future Directions, *IEEE Security and Privacy*, 7, 18–25.

[22] Xu, J., Kalbarczyk, Z., and Iyer, R. K. (2003), Transparent Runtime Randomization for Security, *Reliable Distributed Systems, IEEE Symposium on*, 0, 260.

[23] Bhatkar, S. and Sekar, R. (2008), Data Space Randomization, In *Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings*, Vol. 5137 of *Lecture Notes in Computer Science*, pp. 1–22, Springer.

[24] Chew, M. and Song, D. (2002), Mitigating buffer overflows by operating system randomization, (Technical Report CMU-CS-02-197) Carnegie Mellon University.

[25] Gherbi, A., Charpentier, R., and Couture, M. (2010), Redundancy with Diversity Based Software Architectures for the Detection and Tolerance of Cyber-Attacks, Technical Report DRDC Valcartier.

[26] Gao, D., Reiter, M. K., and Song, D. Xiaodong (2006), Behavioral Distance Measurement Using Hidden Markov Models, In Zamboni, Diego and Krügel, Christopher, (Eds.), *RAID*, Vol. 4219 of *Lecture Notes in Computer Science*, pp. 19–40, Springer.

[27] Knight, J. C. and Leveson, N. G. (1986), An Experimental Evaluation Of The Assumption Of Independence In Multi-Version Programming, *IEEE Transactions on Software Engineering*, 12, 96–109.

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC JUNE 2010

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Towards a reasoning framework using diversity for security

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Khoury, R.

5. DATE OF PUBLICATION (Month and year of publication of document.)

April 2012

6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)

42

6b. NO. OF REFS (Total cited in document.)

27

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Valcartier TM 2013-220

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

N-version programming has been shown to be an effective way to increase the reliability of systems. In this technical report, we examine the possibility of extending this approach to address security, rather than reliability concerns. We focus specifically on how to evaluate the efficiency of the use of diversity for security. We show that while several key elements must be taken into account when N-version programming is used for security rather than reliability, it is nonetheless possible to devise a reasoning framework to evaluate the efficiency of this development paradigm in a security context. Furthermore, we present preliminary empirical results indicating that an effective diversity-based intrusion detection scheme is feasible.

Des études ont démontré que la programmation en n versions est une méthode efficace pour assurer la fiabilité des systèmes. Dans ce rapport technique, nous examinons la possibilité d'étendre cette approche pour as-surer la sécurité des systèmes, plutôt que leur fiabilité. Nous nous concen-trons particulièrement sur l'évaluation de l'usage de la diversité à des fins de sécurité. Nous concluons que plusieurs éléments doivent être pris en compte quand la diversité est utilisée dans l'optique de la sécurisation des systèmes, plutôt que dans celle d'en assurer la fiabilité. Néanmoins, il demeure possible de développer un cadre de raisonnement permettant d'évaluer l'efficacité de ce paradigme dans un contexte de sécurisation des logiciels. De plus, les résultats initiaux de nos études empiriques in-diquent qu'il est possible d'utiliser la diversité à des fins de sécurité d'une manière efficace.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Computer security

Defence R&D Canada

Canada's Leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE R&D DÉFENSE

www.drdc-rddc.gc.ca