Redundancy with diversity-based software architectures for the detection and tolerance of cyber-attacks

State-of-the-Art

Abdelouahed Gherbi
Robert Charpentier
Mario Couture
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2010-287
February 2012

Redundancy with diversity-based software architectures for the detection and tolerance of cyber-attacks

State-of-the-Art

Abdelouahed Gherbi
Robert Charpentier
Mario Couture
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2010-287
February 2012

IMPORTANT INFORMATIVE STATEMENTS

Abstract

Software systems security remains a critical issue. This is evidenced by the ever increasing number and sophistication of cyber-attacks. This situation is the result of the combination of several factors. The software-based functionality of these systems is increasingly complex. The systems are often connected through open networks such as the Internet, which is increasingly accessible to potentially malicious users. Finally, these systems run software which is substantially similar. This is called *IT monoculture*. The mitigation against this issue requires implementation of the principle of diversity. The principle of diversity aims to reduce the common vulnerability in software and, in turn, increase the difficulty of violating the security of the systems that use diversity. The objective of this document is to present the state of the art in terms of approaches which use diversity for security purposes. Three different approaches can be distinguished: automated diversity, diversity-based behavior monitoring and diversity-based intrusion tolerance.

Résumé

La sécurité des systèmes informatique demeure une problématique ardue. Ceci est confirmé avec la croissance continue de cyber-attaques en nombre et en sophistication. Cette situation est le résultat de la combinaison de plusieurs facteurs. La fonctionnalité de ces systèmes basée sur les logiciels est de plus en plus complexe. Les systèmes sont souvent reliés par des réseaux ouverts comme Internet qui est de plus en plus accessibles aux utilisateurs potentiellement malveillants. Enfin, ces systèmes exécutent des logiciels qui sont substantiellement similaires. Ceci est communément appelé IT monoculture. La solution de ce problème est basée sur le principe de la diversité. Cette dernière vise à réduire les vulnérabilités communes dans un ensemble redondant et divers de logiciels. Ceci augmente la difficulté de violer la sécurité des systèmes qui utilisent la diversité. L’objectif de ce document est de présenter l’état de l’art en termes d’approches qui utilisent la diversité à des fins de sécurité. Trois approches différentes peuvent être distinguées : la diversité automatisée, la surveillance de comportement basée sur la diversité et la tolérance d’intrusion basée sur la diversité.

This page intentionally left blank.

Executive summary

Redundancy with Diversity Based Software Architectures for the Detection and Tolerance of Cyber-Attacks

Abdelouahed Gherbi, Robert Charpentier, Mario Couture; DRDC Valcartier
TM 2010-287; Defence R&D Canada – Valcartier; February 2012.

Background: Mission-critical software systems used to support military operations are increasingly complex. Such complexity makes it very difficult to design, deploy, configure and manage these systems without faults. Despite the quality controls used at different steps of the software development process several faults remain hidden and represent latent vulnerabilities. Moreover, the systems are often interconnected through open networks such as the Internet. It is highly likely that, in such hostile environments, determined attacker will end up discovering the system vulnerabilities and be able to mount sophisticated cyber-attacks.

Redundancy has long been used with efficacy as a means to achieve system reliability because hardware failures are typically independent. Therefore, the replication of components provides added assurance. When it comes to software, however, failures are due to design, implementation and/or configuration faults. These faults are embedded within the software and the manifestation of these faults (i.e errors) is systematic. That is, every copy of a faulty software will have an identical behavior when provided with the same (malicious) input. Therefore, redundancy alone is not effective against software faults.

Design diversity is therefore used to achieve software failure independence. This approach is implemented by the N-version programming where different implementations of the same functional specification are developed by independent teams. The rational behind this technique is that the diversification of the software development would yield functionally equivalent programs which do not share the same faults.

From the security standpoint, a fault embedded in a software is a vulnerability. This may end up being successfully exploited by an external interactive malicious fault (i.e. attack) and ultimately enable the violation of the system security property (i.e. a security failure). Consequently diversity has caught the interest of the software security research community. The main idea is that diversity reduces significantly common vulnerabilities which makes it more difficult for an attacker to break into a software system with the same attack.

The main objective of this document is to report on the state of the art in terms of concepts, techniques and technologies based on the deployment of redundancy with diversity as a mechanisms to achieve the security of software systems. This state of the art study revealed that the principle of diversity can complement the principle of redundancy to build software systems that are able to resist, detect and tolerate cyber-attacks. More specifically, this study showed that diversity can be deployed in three different ways:

- Automated Diversity: These techniques consist in applying automated procedures to randomize either the code (Instruction Set Randomization), the address space layout (Address Space Randomization and Data Space Randomization) or both to provide a probabilistic defense against unknown threats.
- Diversity-based behavior monitoring and intrusion detection: Several techniques have been devised to use diversity enable efficient monitoring of the system behavior in order to detect suspicious activities or intrusions.
- Diversity based architectures for intrusion tolerance

Sommaire

Redundancy with Diversity Based Software Architectures for the Detection and Tolerance of Cyber-Attacks

Abdelouahed Gherbi, Robert Charpentier, Mario Couture ; DRDC Valcartier
TM 2010-287 ; R & D pour la défense Canada – Valcartier ; février 2012.

Les systèmes informatiques critiques qui sont utilisés pour supporter les opérations militaires sont de plus en plus complexes. Cette complexité rend très difficile la conception, déploiement, la configuration et la gestion de ces systèmes sans l'introduction de fautes. Malgré les contrles de qualité qui sont intégrés aux différentes étapes du processus de développement de logiciels, plusieurs fautes sont introduites et demeurent cachées et représentent ainsi des vulnérabilités latentes. En outre, les systèmes informatiques ne sont plus isolés. Ils sont interconnectés par des réseaux ouverts tels que l'Internet. Il est trs probable que, dans de tels environnements hostiles, des individus or organisations malicieux et dterminés vont finir par dcouvrir les vulnérabilités du systme et être en mesure de monter cyber-attaques sophistiqués.

La redondance a été efficacement utilisée comme moyen pour réaliser la fiabilité des systémes parce que les pannes matérielles sont généralement indépendantes. Par conséquent, la réplication des composants fournit une assurance supplémentaire. Ceci n'est cependant pas le cas des systèmes logiciels car les défaillances sont dues à des défauts de conception, de mise en uvre et / ou de configuration. Ces défauts sont par conséquent embarqués avec les logiciels et leurs manifestations (erreurs) sont systématiques. Ceci signifie que toutes les copies d'un logiciel présentant un défaut auront un comportement identique vis a vis les mêmes données d'entrée (malveillantes). Par conséquent, la redondance seule n'est pas efficace contre les défauts logiciels.

La diversité de conception a été donc utilisée pour réaliser l'indépendance de pannes logicielles. Cette approche est mise en œuvre par la *N-version programmation* où différentes implémentations de la même spécification fonctionnelle sont développées par des équipes indépendantes. L'idée de base sous-jacente de cette technique est que la diversification du développement du logiciel produirait des programmes fonctionnellement équivalents mais qui ne partagent pas les mêmes défauts.

Du point de vue de scurité, un défaut dans un logiciel est une vulnérabilité. Celle ci peut finir par être exploité avec succès via une faute externe malveillante interactive (une attaque). Une attaque peut causer la violation d'une propriété de sécurité du système. Par

conséquent, la diversité a suscité l'intérêt de la communauté de la recherche en sécurité des logiciels. L'idée principale de l'approche de sécurité basée sur la diversité est que la réduction des vulnérabilités communes a pour conséquence d'accroitre la difficulté pour qu'un individu ou une organisation malicieux réussisse à violer la scurité d'un système logiciel dont l'architecture est fondée sur la diversité avec la même attaque.

L'objectif principal de ce rapport est de documenter l'état de l'art en termes de concepts, des techniques et des technologies basées sur le déploiement de la redondance avec la diversité pour supporter la sécurité des systémes logiciels. Cette étude de l'état de l'art a révélé que le principe de la diversité est necessaire pour complémenter la redondance pour être à la base de systèmes logiciels qui sont capables de résister, détecter et tolérer des cyber-attaques. En particulier, cette étude a montré que la diversité peut être déployée en trois manières différentes :

- Diversité automatisée : Ces techniques consistent à appliquer des procédures automatiques de randomisation soit du code (Instruction Set randomisation), de l'organization de l'espace d'adressage (Address Space Randomization and Data Space Randomization) ou des deux. Ces techniques fournissent un moyen de é probabiliste contre les menaces inconnues.
- Surveillance du comportement et de détection d'intrusion fondées sur la diversité : Plusieurs techniques utilisant la diversité ont été mis de l'avant pour permettre un surveillance éfficace du comportement des systèmes afin de détecter les activités suspectes ou les intrusions.
- Architectures tolerantes aux intrusions basées sur la diversité.

Table of contents

List of figures

1 Introduction

The security of software systems remains an extremely critical issue. This is evidenced by the continuous growth of cyber threats. For instance, Figure 1 which is taken from (Symantec 2009), shows the tremendous increase in terms of new malicious code. In addition, cyber-attacks are not only increasing in number but also in sophistication and scale. Indeed, some attacks are now nation/state class such as the one which targeted Estonia in 2007 (emerging risks team 2009). This observation can be explained by a combination of a multitude of contributing factors which we consider in this section of the document.

The system functionality, which is implemented by software, is increasingly complex. This makes it very difficult to produce fault free software in spite of the quality control that should be part of the software development process. These *residual faults* constitute often dormant vulnerabilities. These would eventually end up being discovered by determined malicious opponents and exploited to carry out cyber-attacks. In (PITAC 2005) the problem of software security was summed up as follows:"*Software development is not yet a science or a rigorous discipline, and the development process by and large is not controlled to minimize the vulnerabilities that attackers exploit. Today, as with cancer, vulnerable soft-ware can be invaded and modified to cause damage to previously healthy software, and infected software can replicate itself and be carried across networks to cause damage in other systems. Like cancer, these damaging processes may be invisible to the lay person even though experts recognize that their threat is growing.*"

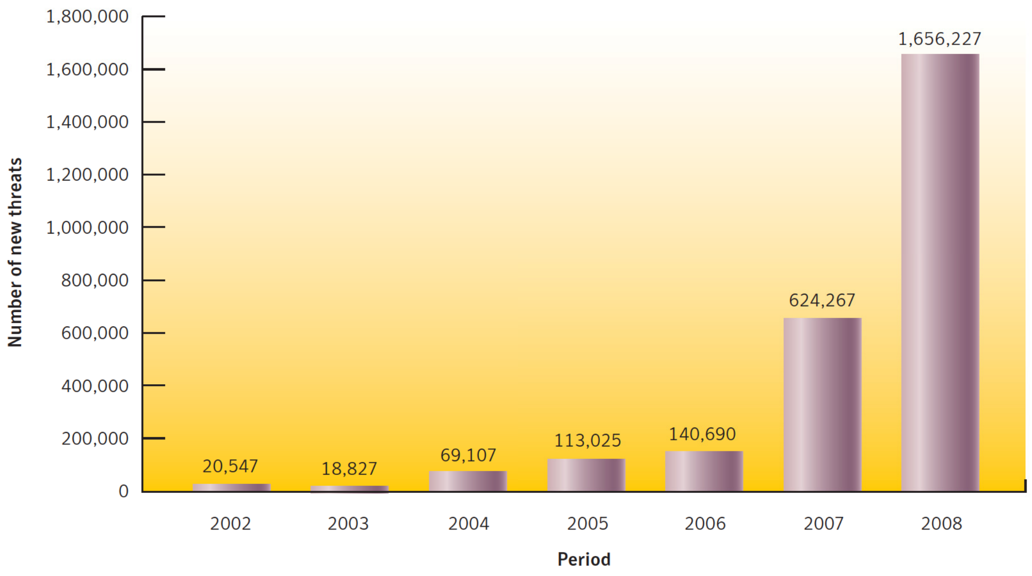


Figure 1: *Growth of Malicious Code (Source: Symantec)*

Software systems are distributed and interconnected through open networks such as the Internet in order to communicate and to transport data. The Internet user population is growing at a dazzling pace (as is shown in Figure 2). This in turn increases tremendously the risk of attacks as a fraction of this population is likely to have malicious motivations.

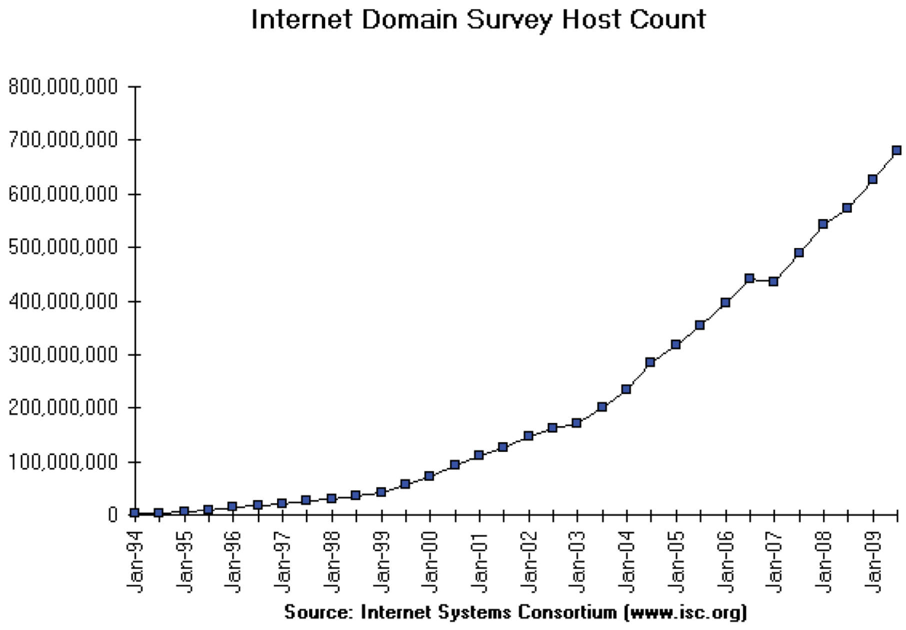


Figure 2: *Internet Growth*

In general, information systems are are running significant similar software. This is called IT monoculture (Lala and Schneider 2009), (Lala 2003). On one hand, IT monoculture present several advantages, including easier management, fewer configuration errors and support for interoperability. On the other hand, IT monoculture has serious disadvan-tages. Indeed, systems share common vulnerabilities and consequently facilitate the spread of viruses and malware. In (Symantec 2009), one can read the following: ”*In order to compromise the largest possible number of web sites with a single mechanism, attackers will attempt to compromise an entire class of vulnerability by searching for commonalities within them and generically automating their discovery and exploitation. This allows attackers to compromise web sites with the efficiency commonly found in network worms.*” This means that the success of attacks depends, to a large extent, on the presence of common vulnerabilities in the IT infrastructure (Operating systems, middleware, Web servers, etc).

One emerging approach which aims at mitigating the effects of IT monoculture in order to achieve software security is based on the principle of diversity. The latter has already been used to complement redundancy in order to achieve software systems reliability and fault tolerance. In the context of software security, the approach based on diversity seeks specifically to reduce the common vulnerabilities between redundant components composing a system. In consequence, it becomes very difficult for a malicious opponent to design one attack that is able to exploit different vulnerabilities in the system components simultaneously. Therefore the resistance of the system to cyber attacks is increased. Moreover, the ability to build a system out of redundant and diverse components provides an opportunity to monitor the system by comparing the behavior of the diverse components when presented with the same input. This enables the system to be endowed with efficient intrusion detection capability. Finally, diversity is not only limited to space by deploying a set of redundant and diverse but functionally equivalent components. It can also be time based which means that the system is designed through diversity with the ability to adapt its behavior or structure over time. This adaptation can be reactive after the detection of an intrusion or proactive.

1.1 The objectives

In this report, we document a state of the art study of the different approaches to software systems security using the diversity principle. In this study we have put the focus on the concepts, techniques and technologies based on the deployment of redundancy with diversity as a mechanism to achieve the security of software systems. Therefore, this document consists mainly of: (i) a comprehensive review of the most important previous and current research initiatives and the available literature devoted to the usage of the redundancy with the diversity principle for the design and deployment of secure, attack tolerant and survivable software systems, and (ii) identification of important research issues that require further effort in a future project. Ultimately, this state of the art study aims to set the stage for an extensive investigation of a framework which enables the construction of attack-tolerant and survivable command and control software system.

1.2 Organization of the Document

The present document is organized as follows: In order to make this report as self contained as possible, we introduce in Section 2, the main concepts and definitions from the domain of dependability and fault tolerance. The idea of using diversity as a defense mechanism and the original research work devoted to it are introduced in Section 3. Sections 4, 5 and 6 are respectively devoted to presenting the three main approaches of using diversity for security purposes, namely, the automated diversity, the diversity-based behavior monitoring and the intrusion tolerance approach. We devote Section 7 to the main research work which focuses on the analysis and evaluation of the extent of security provided by diversity. Other research work related to diversity is presented in Section 8. We conclude this report and present some issues which provide opportunity for future research work in Section 9.

2 Background

In order to make this document as self-contained as possible, we introduce the fundamental concepts and definitions from the domain of dependable computing and fault tolerance.

2.1 System Dependability and Security

Dependability and security are two important system properties. The concept of dependability is defined in (Avizienis et al. 2004) as the ability to deliver service that can be justifiably trusted. Dependability is an integrating concept which encompasses several attributes: availability, reliability, safety, integrity and maintainability. Security is also an integrating concept, which encompasses the attributes of confidentiality, integrity and avail-ability. Therefore, dependability and Security are closely related and their relationship is defined in terms of their main attributes as shown in Figure 3.

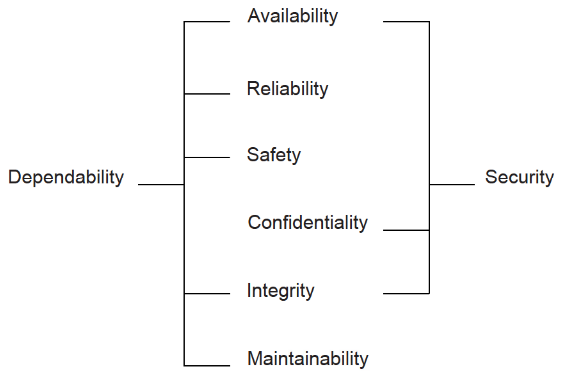


Figure 3: *Dependability and Security Attributes*

A conceptual framework has been established in (Avizienis et al. 2004) to present the different aspects related to the concept of dependability. This framework considers the concept of dependability from three different perspectives: Attributes, Threats and Means as it in Figure 4.

In the following sections, we focus on the threats to and means to achieve dependability.

2.2 Threats to Dependability: Fault, Error and Failure

In this section, we introduce the definitions of the concepts of Fault, Error and Failure. These definitions are the most widely accepted in the system dependability and fault-tolerance community considering a variety of publications (Avizienis et al. 2004), (Kienzle 2003), (Laprie 1995).

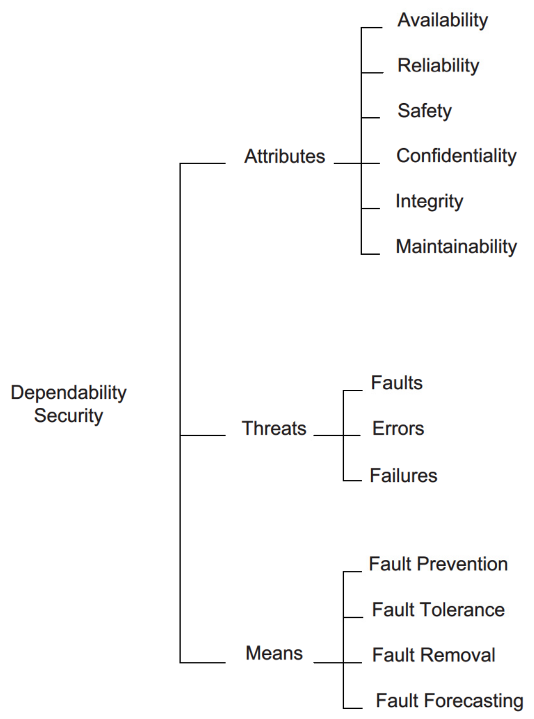


Figure 4: *Dependability and Security Attributes, Threats and Means*

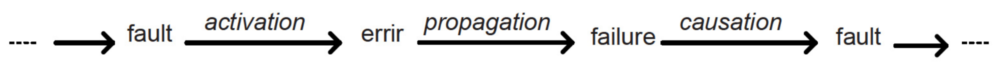


Figure 5: *Faults, Error, Failure Chain*

The service delivered by a system is its behavior as perceived by the user at the system *user interface*. The user is another system (physical, human) which interacts with the system. It is then a sequence of the system external states.

The service is said to be *correct* when it implements the function of the system as it is specified in the functional specification. Therefore, a *service failure* is an event which occurs when the delivered service deviates from the correct service. The deviation is called an *error* and is the part of the total state of the system that may lead to its subsequent service failure. The error has an adjudged or hypnotized cause of an error is called a fault. There is a recurrent causation relationship between fault, error and failures as it is shown in Figure 5.

A comprehensive classification of faults according to six major criteria has been introduced in (Avizienis et al. 2004) and it is shown in Figure 6.

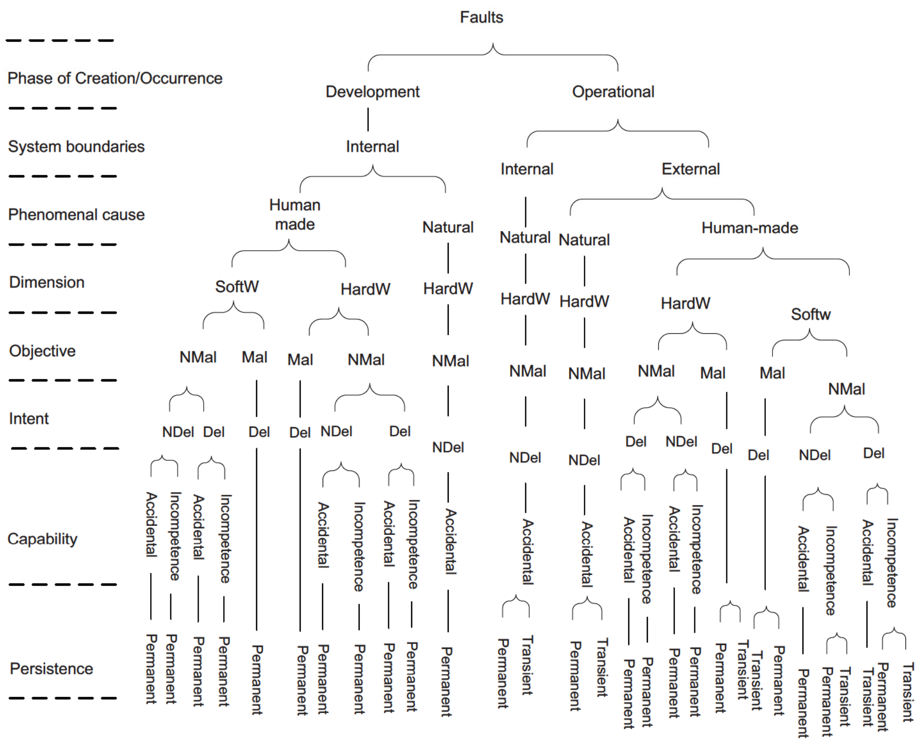


***Figure 6:** Faults Classification*

2.3 Threats to Security: Vulnerability, Attack, Intrusion and Security Failure

According to the fault classification presented in the previous section, it emerges that vulnerability, attack and intrusion are special classes of faults. These are defined as follows:

- **Vulnerability:** It is a fault created during the development or operation of of the system (e.g. design, configuration faults) that could be exploited to create an intrusion. For example, during a multi-stage attack, an attacker might try to introduce vulnerabilities in the system in the form of malware.
- **Attack:** A malicious interaction fault that attempts to exploit existing vulnerabilities in the system, through which an attacker aims to deliberately violate one or more security properties.
- **Intrusion:** It is a malicious, externally-induced fault resulting from an attack that has been successful in exploiting a vulnerability. An intrusion is characterized by an erroneous system state. For example, a system file with an unjustifiable access privileges for the attacker.
- **Security Failure:** There is a security failure whenever any of the security properties or the security policy goals in place is violated. Security failures caused by the errors, which are in turn caused by intrusions, are not properly handled.

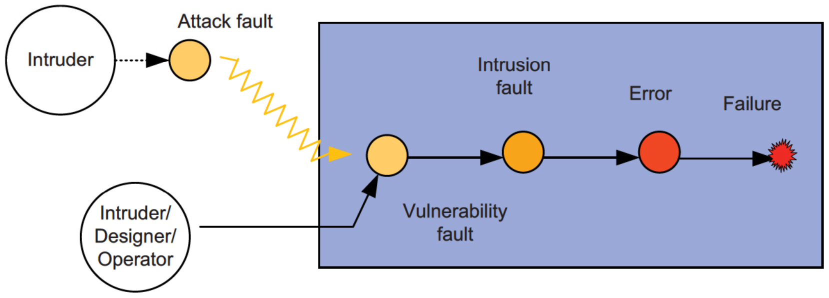


Figure 7: *AVI Model*

Figure 7 depicts the Attack Vulnerability Intrusion (AVI) fault model 7. This model is one of the results of the MAFTIA project (see 6.2.2) on intrusion tolerant systems and it captures the relationship between the threats to security mentioned above.

2.4 How to Achieve Dependability?

The followings are the major methods that are often used in combination to achieve the dependability requirements:

- **Fault prevention** means to prevent the occurrence or introduction of faults.
- **Fault tolerance** means to avoid service failures in the presence of faults.
- **Fault removal** means to reduce the number and severity of faults.
- **Fault forecasting** means to estimate the present number, the future incidence, and the likely consequences of faults

The objective of fault tolerance is to avoid system failures. To this end, errors should be prevented from causing failures and therefore they should be detected and properly handled. In the case where errors have caused failures, the system service should be recovered. In both cases the faults which originated the errors should be handled also. The different techniques involved in fault tolerance are shown in Figure 8.

2.5 Intrusion Tolerance and Survivability

A system is said to be survivable when it is designed with the capability of providing its services in a timely manner even if significant portions are incapacitated by *attacks* or *accidents* (Barbacci 1996), (Hiltunen et al. 2003). System survivability is a more general concept than intrusion tolerance. Indeed, an intrusion tolerant system has the capability of providing a secure service in spite of intrusions in some of its components (Verissimo et al. 2003). This definition implies that the functionality may be degraded but its secu-rity (i.e. availability, integrity and confidentiality) is preserved.

The underlying idea of intrusion tolerance is to apply the fault tolerance paradigm to address the issue of software system security. Intrusion tolerance is not a new concept nor

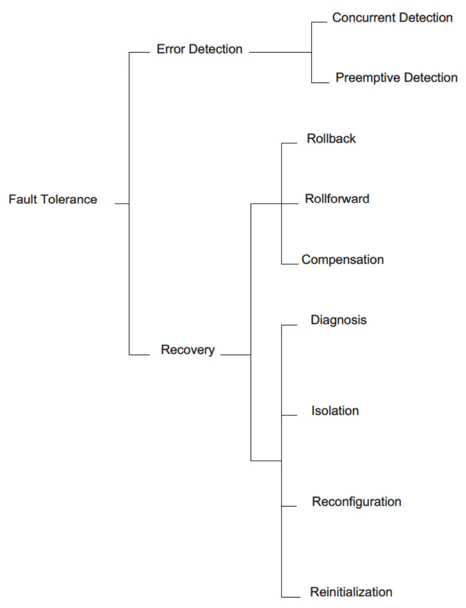


Figure 8: *Faults Tolerance Techniques*

is the application of a fault tolerance approach to security a new idea. As a matter of fact, the first proposal of intrusion tolerant systems goes back the seminal work of (Fray et al. 1986) and (Joseph and Avizienis 1988). The intrusion tolerance approach is based on an important assumption, which is failure independence. In order to achieve failure independence, redundancy with diversity is used. In the following section, we focus particularly on the principle of diversity for security. Later, we consider software architectures which used diversity to achieve security. Figure 9 shows a conceptual model, which is put forward in the MAFTIA project, of the intrusion tolerance approach and how it complements the traditional approach to dealing with the different security threats (vulnerability, attack).

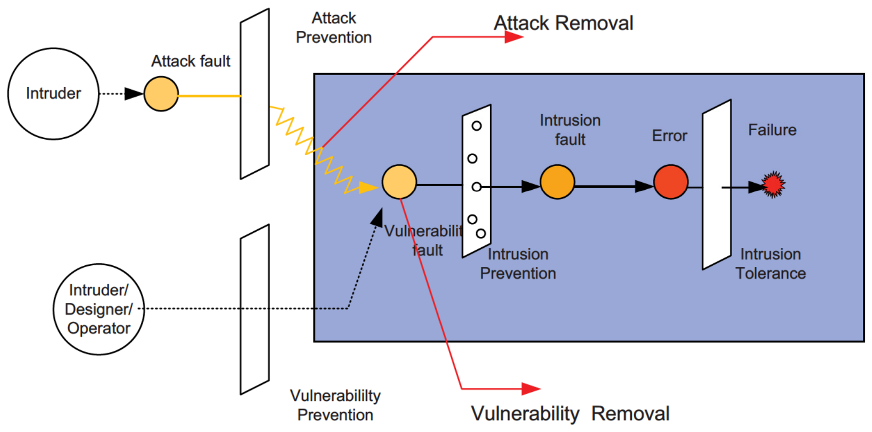


Figure 9: *Intrusion Tolerance Conceptual Model*

3 Diversity as a Defense Mechanism

In this section, we focus on the origin of the diversity-based software security approach. We present the first research initiatives that were devoted to this concept.

3.1 Redundancy Limitation for Software Fault Tolerance

Redundancy has long been used by the fault tolerance community as a fundamental means to achieve higher system reliability. This has proven to be valid mainly for hardware because of the failure independence assumption as hardware failures are typically due to random faults. Therefore, the replication of components provides added assurance. When it comes to software, however, failures are due to design and/or implementation faults which escape the quality controls in place during the development processes. As a result, such faults are embedded within the software and the manifestation of these faults (i.e errors) is systematic. That is, every copy of a faulty software will have an identical behavior when provided with the same input. Therefore, redundancy alone is not effective against software faults.

3.2 Design Diversity

Design diversity is therefore used to achieve software failure independence. This approach is implemented by the N-version programming where different versions of the same program (i.e. different implementation of the same functional specification) are developed by independent teams. The rational behind this technique is that the diversification of the software development would yield functionally equivalent programs which do not share the same faults.

3.3 Diversity for Security Purposes

From the security standpoint, a fault embedded in a software is a vulnerability. This may end up being successfully exploited by an external interactive malicious fault (i.e. attack) and ultimately enable the violation of the system security property. It is therefore a logical consequence that diversity has caught the interest of security research community. The main idea is that by the means of diversity common vulnerabilities are decreased if not eliminated. As a result, it is very difficult for a malicious opponent to be able to break into a system composed of a set of diverse yet functionally equivalent components with the very same attack.

In the following, we succinctly review the main research initiatives into the use of diversity as a defense mechanism:

- The seminal work presented by Forrest et al. (Forrest et al. 1997) promotes the general philosophy of system security using diversity. The authors argue uniformity contains, as a side effect, a potential weakness, because any flaw or vulnerability in an application is replicated throughout many machines. Therefore, the security and the robustness of a system can be enhanced through the deliberate introduction of diversity. The paper outlines how to introduce diversity using randomized compilation. In particular, this work discusses a specific extension to the GNU GCC Compiler which, basically, pads each stack frame by a random amount to defeat stack-based buffer overflow attacks.

- Deswarte et al. review (Deswarte et al. 1998) how the different levels of diversity of software and hardware systems (user and operator level, human-computer interface, application software level, execution level, and hardware or operating system level) have made those systems more reliable and secure. More recently, the work presented in (Obelheiro et al. 2006) distinguishes different dimensions and different degrees of diversity.

- Bain et al. (Bain et al. 2001) presented a study of a set of widespread computer attacks. This set of attacks includes the Morris worm, which spread by exploiting vulnerabilities in TCP/IP capabilities; the Melissa virus, which used the macro capabilities of a Microsoft Word attachment to e-mail; and the LoveLetter worm, which used a Visual Basic script attached to e-mail. The objective of this study was to understand the effect of diversity on the survivability of attacked systems.

- Last but not least, panels of renowned researchers have held intense discussions to address the use of diversity as a strategy for computer security and to identify the main remaining related open issues requiring further research (Taylor and Alves-Foss 2005). The main outcome of such discussions is that diversity is a different security paradigm which is in need of further definition. In particular, the discussions highlighted the lack of quantitative information on the cost associated with diversity based solutions and the lack of knowledge about the extent of protection provided by diversity.

We can distinguish three categories of approaches which use diversity for security purposes:

- Automated Diversity Techniques

- Diversity-based Intrusion Detection

- Diversity-based for Intrusion Tolerance

The research work pertaining to these different approaches is reviewed and analyzed in the following chapters.

4 Automated Diversity Techniques

The Diversity-based approach to addressing software system security, which consists in developing and deploying multiple versions of the program, is in general associated with higher cost. In order to cope with this issue, automated (also called artificial) diversity techniques (Just and Cornwell 2004) are used. These techniques consist in applying automated procedures to create diverse versions of the software. However, it is not straightforward to create diversity automatically at the level of functional behavior of programs such as design and algorithms. Therefore, the diversity-based security approach is weak in dealing with vulnerabilities that involve design or logical faults including, for example, input validation faults. Automated diversity is introduced by applying automatic program transformations that preserve functional behavior and the programming language semantics. This consists essentially in a randomization of either the code, the address space layout or both to provide a probabilistic defense against unknown threats. On the other hand, automatic introduction of diversity presents some shortcomings. In particular, it is not straightforward to create diversity automatically at the level of functional behavior of programs such as design and algorithms. Therefore, diversity-based security approach is weak in dealing with vulnerabilities that involve design or logical faults including, for example, input validation faults.

4.1 Instruction Set Randomization

The Instruction Set Randomization (ISR) technique (Kc et al. 2003) (Barrantes et al. 2003)(Keromytis 2009) changes the instruction set of the processor so that unauthorized code will not run successfully. The main idea underlying ISR is to decrease the knowledge of the attackers about the language used by the runtime environment on which an application runs. These techniques aim at defending against code injection attacks, which consist in introducing executable code within the address space of a target process, and then passing the control to the injected code. Code injection attacks can succeed when the injected code is compatible with the execution environment. For example, the injection of x86 machine code to a process running on a SPARC system would crash the process rather than causing a security failure.

The usage of ISR ggenerates a diversification of the runtime environment such that a successful attack against one process or host will not succeed against another. This is particularly useful in the context of self-propagating malware such as virus and worms, which exploit the common vulnerability across different systems, and therefore are able to compromise several systems.

4.2 Address Space Randomization

Address Space Randomization (ASR) (Shacham et al. 2004) is a technique used to increase software resistance to memory corruption attacks. These are designed to exploit memory manipulation vulnerabilities such as stack and heap overflows and underflows, format string vulnerabilities, array index overflows, and uninitialized variables. The ASR technique consist basically in randomizing the different regions of the process address space such as the stack and the heap. The ASR technique has been integrated into the default configuration of Windows Vista operating system (Whitehouse 2007). Two subcategories of ASR can be distinguished. The first one is Absolute Address Randomization AAR include [][]Transparent Runtime Randomization (TRR) (Xu et al. 2003), which randomizes the absolute memory address of various code and data objects but not the relative distances between objects. These techniques are effective against pointer corruption attacks because the objects referenced by a corrupted pointer value is no longer predictable. As an example, the stack-smashing attack which overwrites functions return address with a value pointing to a buffer variable holding injected code or data, is defeated using the AAR technique. This is because the location of the buffer becomes unpredictable. The second category of ASR technique is called Relative Address Randomization (RAR), which also randomizes the inter-object distances and therefore is able to defeat non-pointer attacks.

4.3 Data Space Randomization

Data Space Randomization (DSR) is a different randomization-based approach which also aims at defending against memory error exploits (Bhatkar and Sekar 2008). In particular, DSR consists randomizing the representation of data objects. This is often implemented by applying a modification of the data representation such as applying an XOR operation to each data object in the memory using a random value (i.e. a mask). The data is unmasked right before its use. Therefore, this makes the usage of the corrupted data highly unpredictable. This is because even if a malicious opponent succeeds in corrupting a data (e.g overwrites a variable X with a value v), using a buffer overflow for example, this data is interpreted with a random value (the value of X is interpreted as $v \bigotimes m_X$ where m_X is the mask corresponding to X). Consequently, because of this random incorrect value $v \bigotimes m_X$, the software may behave unpredictably (in general it crashes) preventing the security from being compromised. The DSR technique seems to have advantages over ASR. In particular, DSR provides a larger range of randomization such that on 32-bit architectures, integers and pointers are randomized over a range of 2^{32} values. In addition, DSR is able randomize relative distance between two data objects which addresses the weakness of the ASR technique.

4.4 System Call Randomization

One way to achieve a code injection attack is based on making direct system calls in order to interact with the operating system. Chew and Song (Chew and Song 2002) proposed a technique to defend against such attacks based on randomizing the mapping of system calls. This defense forces the attackers to guess the system call numbers and consequently any wrong number used will lead to an incorrect correct system call. This technique requires a recompilation of the kernel with the randomized system call mapping. In addition, it requires rewriting of existing binaries such that the old system call numbers are replaced to reflect the new mapping. Even though this work is widely referenced by the different authors of the other automated diversity techniques, it is not clear whether this work has been evaluated.

	Artificial Diversity Techniques		
	Objective	• Defense against code injection attacks • Defense against memory-related vulnerability exploits	
	Principle	Disadvantage	Reference
Instruction Set Randomization	diversification of the runtime environment (processor language)	• Protection limited to code injection attack • Requires run-time support	• (Kc et al. 2003) • (Barrantes et al. 2003) • (Keromytis 2009)
Address Space Randomization	Randomization of the process address space regions (e.g stack, heap)	• Ineffective against attacks that corrupt non-control data • Low entropy enabling brute force attacks	• (Shacham et al. 2004) • (Whitehouse 2007) • (Xu et al. 2003)
Data Randomization	Randomization of the representation of data objects through the application of XOR operations to each data object using a random value	• runtime overheads due to: • need for masking/unmasking after every memory access • additional memory overheads for accessing mask data	(Bhatkar and Sekar 2008)
System Call Randomization	Randomization of the mapping of system calls in order to decrease the attacker correct system call number knowledge	• Recompilation of the OS kernel • Rewriting existing binaries	(Chew and Song 2002)

5 Diversity-based Intrusion Detection

Intrusion Detection Systems (IDSs) are an established component of the security solution. The objective of IDSs is essentially to generically detect attacks/intrusions. In general, IDSs can be classified according to the approach used which can be knowledge-based and behavior-based. The IDSs which implement the former approach use a signature database that describes known attacks and intrusions, using that information to perform the detection. The IDSs based on the latter approach have or build data about what is normal or expected behavior, and detect deviations from it. Moreover, IDSs can also be host-based (i.e. detect intrusions in the host) or network-based (i.e detect malicious activity in a network). A difficult issue with IDSs is the problem of false negatives and false positives. The former is the omission of the detection of a real intrusion and it is due to the lack of knowledge (signature) about the IDSs database. The latter are alarms which turn out to not to be a real attack/intrusion. This is a problem of reference model accuracy. A typical and conservative strategy used by the IDSs consists in generating large quantities of these alarms in order to minimize missing a real malicious activity. The obvious drawback of such a strategy is the system management is tedious.

In this section, we put the focus on intrusion detection systems for two reasons. First, IDS are a part of an intrusion tolerance architecture as they are the trigger of the recovery processes or the adaptive responses. In addition, intrusion detection can take advantage of the same principle of diversity as systems that are based on output voting. Second, intrusion detection systems can themselves be made intrusion tolerant. In the following section we consider the research work which addresses this aspect.

5.1 Intrusion Detection using Output Voting

Several systems used output voting for the sake of detecting some types of server compromises. As an example, the HACQIT system (Reynolds et al. 2002), which will be described further in this report, uses the status codes of the server replicas responses. If the status codes are different, the system detects a failure. Totel et al. (Totel et al. 2006) extended this work to make a more detailed comparison of the replica responses. They realized that web server responses may be slightly different even when there is no attack, and proposed a detection algorithm to detect intrusions with a higher accuracy. These projects specifically target web servers and analyze only server responses. Consequently, they cannot detect a compromised replica that responds to client requests consistently, while attacking the system in other ways.

5.2 Behavior monitoring in N-Variant Systems

N-variant systems (Cox et al. 2006) is a framework which enables execution of a set of automatically diversified variants using the same inputs. The framework monitors the behavior of the variants in order to detect divergence. The variants are built so that an anticipated type of exploit can succeed on only one variant. Therefore, such an exploit can be rendered detectable. The building of the variants requires a special compiler or a binary rewriter. Moreover, this framework detects only anticipated types of exploits, against which the replicas are diversified.

5.2.1 Multi Variant Execution Environment

The multi variant code execution (Salamat et al. 2009) (Weatherwax et al. 2009) is a run-time monitoring technique, which prevents malicious code execution. This technique uses diversity to protect against malicious code injection attack. This is achieved by running several slightly different variants of the same program in lockstep (Salamat et al. 2009). The behavior of the variants is compared at some synchronization points, which are in general system calls. The divergence in the behavior is suggestive of anomaly and raises an alarm.

The architecture of Multi Variant Execution Environment (MVEE) proposed in (Salamat et al. 2009) is shown in Figure 10. It allows conventional applications to run along with ones which are protected through diversity. MVEE is an unprivileged user-space application which does not need kernel privileges to monitor application. The main features of this architecture are:

- The monitor: This is the main component of the MVEE. The monitor creates child processes for the variants. System call made by the variants are intercepted by the monitor in order to synchronize the variants. These should make the same system call with equivalent arguments within a short time window.
- Reverse Stack Execution: This is a compiler driven technique used to generate program variants. In (Salamat et al. 2009), the two generated variants which write the stack in opposite directions (i.e conventional downward direction and the reverse -upward direction). This enables a variant to be obtained that is resilient against code injection attacks exploiting stack based buffer overflow such as activation record overwrites, return-to-lib(c) and function pointers overwrites. Indeed, the two variants will execute two different sets of instructions causing a divergence that will be detected by the monitor.

5.3 Behavioral Distance

Diversity can be leveraged to support and enhance the effectiveness of intrusion detection systems. Traditional anomaly-based intrusion detection proved to be of limited effectiveness against mimicry attacks (Giffin et al. 2006). These attacks managed to emulate the original system behavior including returning the correct service response. These limitations were strong motivations for using "behavioral distance", which measures the extent to which two processes behave differently when provided the same inputs. Therefore, behavioral distance enables the detection of sophisticated attacks through the comparison of the behaviors of two diverse processes running the same input.

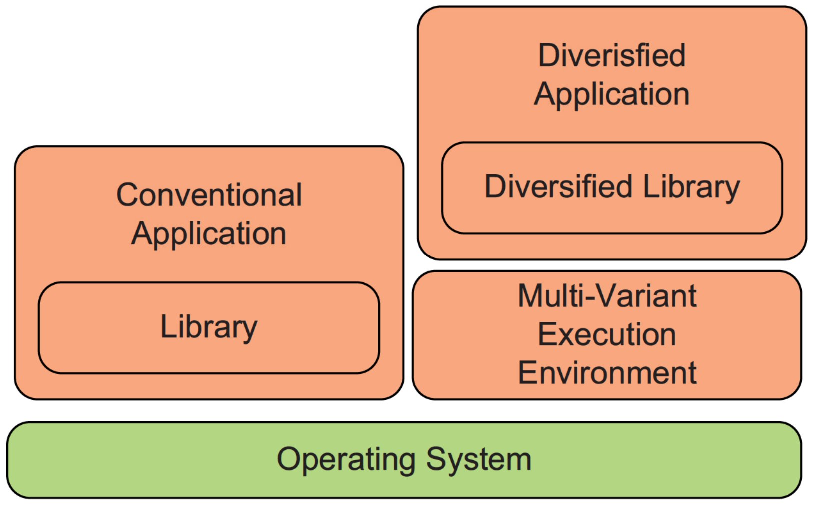


Figure 10: *Multi Variant Execution Environment Architecture*

Gao et al. proposed two approaches (Gao et al. 2006a) (Gao et al. 2006b) for the measurement of behavioral distance. The first approach (Gao et al. 2006a) is based on Evolutionary Distance (ED) (Sellers 1974), which is studied in the molecular biology and evolution field of research. ED-based behavioral distance of two sequences of system calls is a two step process. The first step is a pre-processing, which handles the misalignment between system call sequence due to the diverse implementations or diverse platforms. The second step computes the distance as a sum of the distances of corresponding pairs of system calls using a universal distance table. This table is established in a learning phase in a secure operational context. The ED-based behavioral distance approach suffers from an important limitation, however, which stems from the fact that it does not take into consideration the order of system calls. For instance, reversing the two sequences yields the same distance.

The second approach (Gao et al. 2006b) is based on Hidden Markov Models (HMM) and it addresses the aforementioned limitation of ED-based behavioral distance. Intuitively, HMM models are double stochastic processes. A first non observable stochastic process influences a second which generates a sequence of observable symbols. These symbols represent process behavior, such as the system calls, while the non observable states model the by-product tasks performed by the processes such as opening a network socket or writ-ing a file.

The important observation put forward by the author is that these hidden states should be the same or similar, which corresponds to the same program executed on different platforms or two programs implementing the same functionality such as two different web servers. The observable behaviors of the two processes can then be correlated in the case of no attack otherwise the behavioral distance is increased. Gao et al. use a single HMM to model both processes where a pair of system calls is an observable symbol of the HMM. Each observable symbol is generated by hidden states with some finite probability. With the HMM-based approach, the behavioral distance is the probability with which the HMM generates the pair of system call sequences of interest.

One general drawback of the behavioral distance approach is that it is control flow based; it uses only the system call number and not the system call arguments. Therefore, the attacks which do not change the system call behavior but exploit the system call arguments would not change the distance and therefore would not be detected. Gao et al. presented (Gao et al. 2009) a very interesting proof of concept in order to demonstrate how behavioral distance can be used in practice and to evaluate its performance. This proof of concept consists in a software architecture based on virtualization, which enables monitoring of the system call behavior of two diverse platforms. This architecture is used to implement via two applications namely a web server and a game server. This implementation shows the feasibility of the concept but also reveals the challenges which face the use of behavioral distance. In particular, it is necessary to account for the semantics of the service to be monitored in order to correctly capture the system call sequences. Indeed, the web server application, which uses the traditional request-response-transactional model, was relatively easy to adapt in contrast to the game server application where the authors had to use the player ID (semantic level information) in order to differentiate the system calls instead of the process/thread ID in the case of the web server application.

Finally, the diversity dimension considered in the software architecture presented in (Gao et al. 2009) is limited to two operating systems both running the same application in parallel. This suggests that the scalability of this approach to higher levels of diversity involving complex configurations, including the usage of more operating systems and/or different implementations of the same application needs further investigation.

5.4 Intrusion Tolerance for Intrusion Detection Systems

IDS is basically a software system, which may have security weaknesses and vulnerabilities of its own that result from flawed design assumptions. Therefore, the security of the IDS is also an issue which need to be addressed. In addition, IDS are an important component of the security solution, which makes them targets for attackers, who attempt to crash an IDS through exploiting these vulnerabilities. Attackers tend to first disable the IDS in order to conceal their subsequent behaviors. Therefore, it is necessary to ensure continuity of IDS service. However, as a newly emerging field, IDS dependability has not been addressed adequately. Some research proposals (Shen et al. 2000) (Yu and Frincke 2004)(Siqueira and Abdelouahab 2006) discuss the issue of establishing mechanisms to respond to attacks. However, the mechanisms proposed are isolated from the IDS infrastructure. Furthermore, the effectiveness and performance of the mechanisms are not rigorously evaluated across IDSs.

	Diversity-enabled Behavior Monitoring			
	Principle	Advantage	Limitations	Reference
Output Voting	Comparison of the output of a diversified set of servers given the same input	Efficient for web servers	ineffective against compromised replica that responds to client requests consistently	• (Reynolds et al. 2002) • (Totel et al. 2006)
N-Variant Systems	Framework enabling the execution of artificially diversified variants with the same input	The variants are built so that the exploits can succeed on only one variant	• Requires special compiler or binary rewriter • Detect only anticipated exploits	• (Cox et al. 2006)
Behavioral Distance	Measures the extent to which two (diverse) processes behave differently when provided the same inputs	Effective against mimicry attacks as it does not only focus on the output but monitor the behavior	• control flow based as it uses only the system call number and not the system call arguments (The attacks which exploit only the system call arguments would not be detected)	• (Gao et al. 2006a) • (Gao et al. 2006b) • (Gao et al. 2009)

6 Diversity-based for Intrusion Tolerance

This section is devoted to the presentation of the most important software architecture. We distinguish two categories of such architectures. The first category encompasses those that use and integrate COTS applications, without almost any modification to the applications, within a system to provide intrusion tolerance. The second category provides intrusion tolerance through a middleware. The applications are then aware of the middleware in order to take advantages of its services.

6.1 System Integration Architectures

The software architectures described in this section implement the architectural pattern depicted in Figure 11. This approach is ideal for a system integration of Out-Of-The-Shelf (OTS) components or legacy and closed applications in order to deliver the services. The servers are shielded from the user side through proxies. A monitoring mechanism and a voting mechanism are used to check the health of the system, validate the results and detect abnormal behavior.

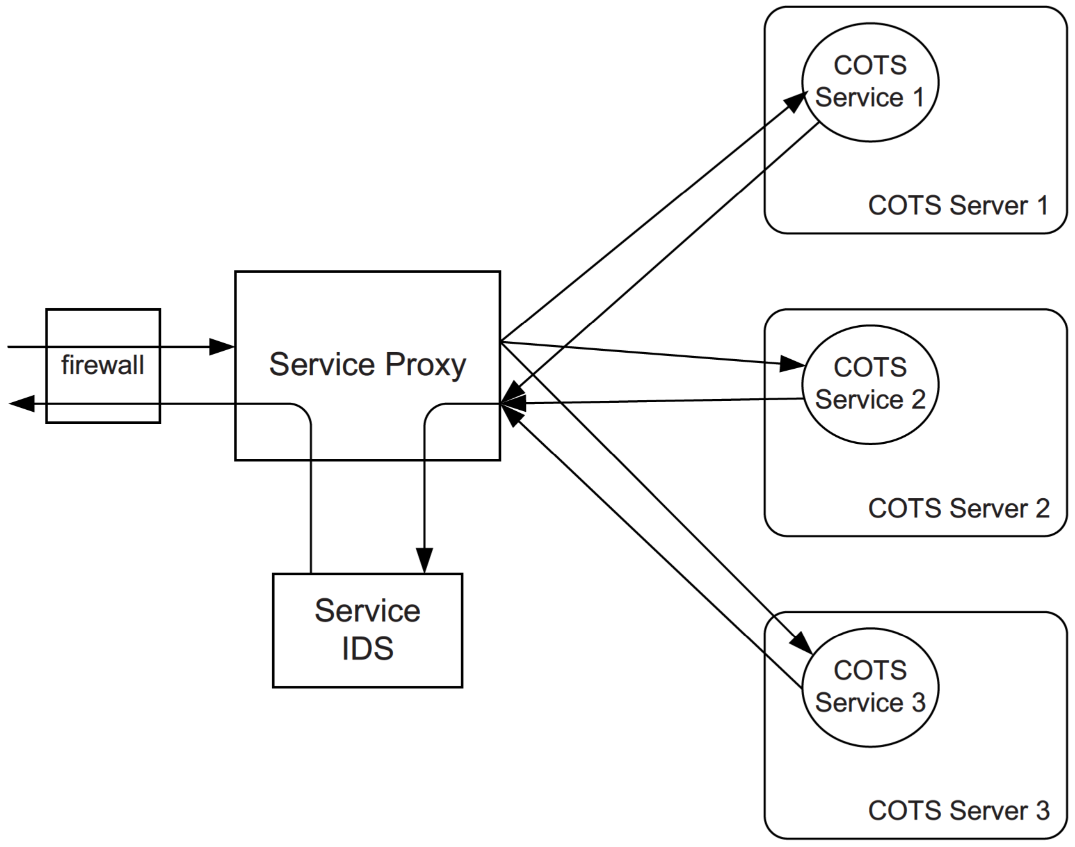


Figure 11: *General Pattern of Intrusion Tolerance Architecture*

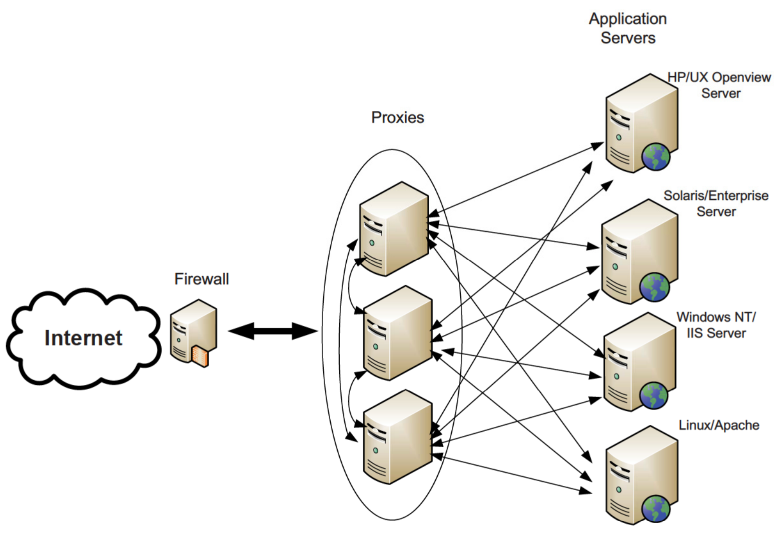


Figure 12: *DIT Architecture*

6.1.1 Dependable Intrusion Tolerance Architecture

The Dependable Intrusion Tolerance (DIT) architecture was developed in cooperation between LAAS-CNRS and SRI International (Deswarte and Powell 2004), (Valdes et al. 2003). This architecture is used to build Web servers that continue to provide correct service in the presence of attacks. The architecture design, which is shown in Figure 12, is based on a diversification approach. Indeed, the architecture used a diversified set of Web servers (Apache, Microsoft IIS, Enterprise Server, Openview Server etc.) running on top of a diversified set of hardware including Sparc, Pentium, PowerPC which in turn run a diversity of operating systems (Solaris, Microsoft Windows, Linux, MacOS etc.). In this architecture, the servers are isolated from the Internet through proxies. The latter are purpose-built software running on a diversified hardware. In addition, the requests coming from the Internet are filtered by a firewall. This architecture shows the complementarity of diversity based intrusion tolerant and prevention security mechanism. The filtered requests are handled by the proxy *leader*, which then distributes them to the Web servers and checks the responses before forwarding them to the client. The remaining set of proxies monitor the behavior of the leader by observing the traffic firewall/proxy and proxy/servers. A new leader is elected in case of the leader failure. In addition, the proxies process alarms issued by intrusion detection sensors placed on the Web servers and the networks.

In this architecture, different redundancy levels (the number of actual different servers used to handle the same request) are used namely simplex mode (one server), duplex mode (two servers), triplex mode (three servers) or all available servers. The servers deliver back to the leader a response and an MD5 checksum of the response. These are compared by the leader in order to decide which response to send back to the user. This comparison is a majority vote in case of triplex or all-available modes.

The alert level is set based on the alarms triggered by the intrusion detection mechanisms, the errors detection mechanisms (result comparison) and external information sources (CERTs). The redundancy level is dynamic and is set to the alert level and it represents the adaptation in this architecture.

The DIT architecture has been enhanced and extended recently in (Saidane et al. 2009) to propose a generic intrusion tolerant architecture for Web servers. Indeed, the proposed architecture, in contrast with the original DIT architecture which supports only servers with static content, has provision to address dynamic content and online updating issues.

6.1.2 Salable Intrusion-tolerant Architecture

Scalable Intrusion Tolerant Architecture (SITAR) for distributed services is an intrusion tolerant architecture (Wang et al. 2003b). The general structure of SITAR architecture is shown in Figure 13. The architecture is therefore composed of the following components:

- Proxy Servers: The proxy servers are the only machines of the architecture that are visible to the end users. Therefore, the COTS servers are not directly accessible. In SITAR, the proxy servers share a pool of virtual IP addresses which support address migration and thus enable the load balancing and dynamic reconfiguration. Since the proxy servers are directly accessible by the users, they are exposed to the risk of attacks. In order to detect such attacks or compromised proxy servers Intrusion Detection System (IDS) are deployed on each proxy server. The IDS software on each proxy server monitor the network traffic as well as the behavior of the other proxy servers. The IDS notify the reconfiguration module upon detection of an attack or a compromised proxy server.
- Acceptance Monitors: The acceptance monitors receive the responses from the COTS servers and perform acceptance verification checks. They forward the responses and the verification results to the Ballot Monitors. In addition, the acceptance monitors are in charge of detecting intrusions in the COTS servers and notifying the reconfiguration modules. The verification checks are highly application dependent and are in general checks on the reasonableness of the results.
- Ballot Monitors: The ballot monitors carry out the decision making process to determine the final response. This is achieved through a majority voting or Byzantine agreement process.
- Audit Control: Each of the above described components of the SITAR architecture maintains a record of its activity. The audit control component verifies the audit records and identifies abnormal behavior in the components by conducting periodic diagnostic tests. It also maintains test suites for the components with the required responses to the tests. It generates requests to the components and detects abnormal behavior by verifying the responses to the requests. The audit control forward the responses to the dynamic reconfiguration module for further actions.
- Dynamic Reconfiguration Module: SITAR is a reconfigurable architecture by supporting various configuration options in support for different security levels. The main aspects of reconfiguration in SITAR are: the COTS services offered via the proxy servers can be distributed among the servers based on the service itself (e.g. if the service is only provided by a subset of the servers), the current load on each proxy or the desired redundancy. The Ballot Monitor might use more robust checksum (e.g. keyed-MD5) to meet

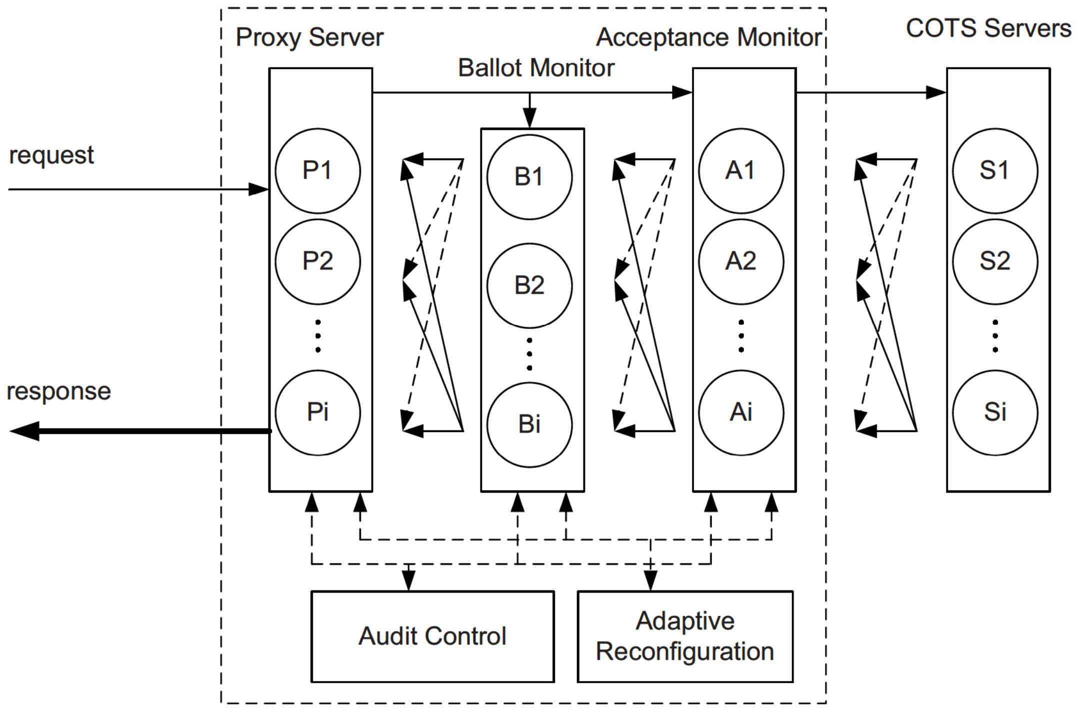


Figure 13: *SITAR Architecture*

increased security requirements. A shift from single ballot voting to distributed voting. The active redundancy configuration varies from zero to full servers. The ARM evaluates the notifications and the security relevant information from other modules and decides on whether a reconfiguration is necessary. Reconfiguration for the Proxy Servers includes changing the level of access control imposed on clients, degrees of redundancy used to fulfill a client request and increased auditing.

6.1.3 Hierarchical Adaptive Control of Quality of service for Intrusion Tolerance

Hierarchical Adaptive Control for QoS Intrusion Tolerance (HACQIT) (Reynolds et al. 2002) is a project that aims at providing intrusion tolerance for web servers. The architecture is made up of two COTS web servers: an IIS server running on Windows and an Apache server running on Linux. One of the servers is declared as the primary and the other one as the backup server. Only the primary server is connected to users. Another computer, the Out-Of-Band (OOB) computer, is in charge of forwarding the request of each client from the primary server to the backup one, and of receiving the responses from each server. Then, they compare the responses given by each server. The comparison is based on the status code of the HTTP response. In addition to this detection mechanism, host monitors, application monitors, a network intrusion detection system (Snort) and an integrity tool (Tripwire) are also used to detect intrusions.

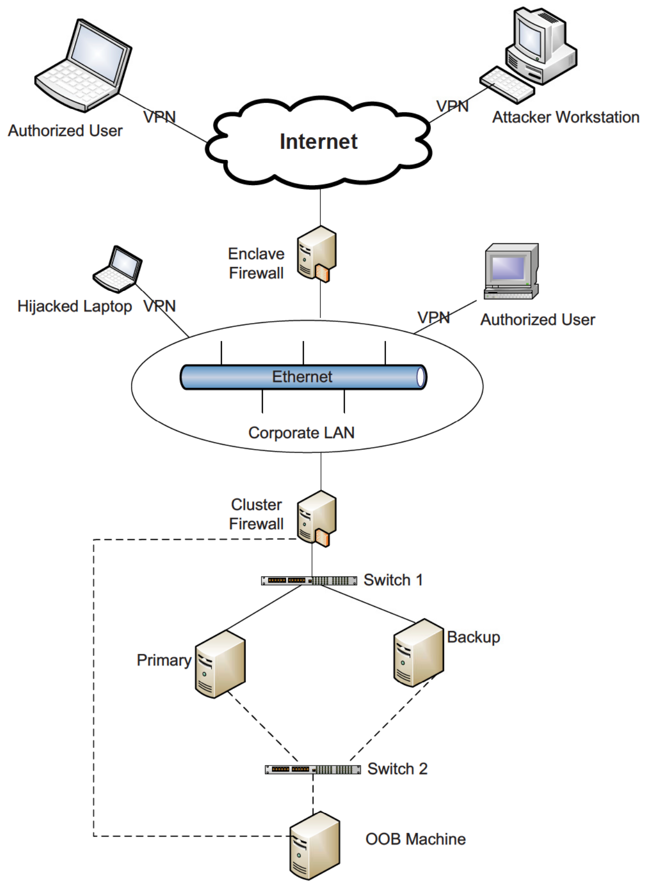


Figure 14: *HACQIT Architecture*

6.2 Middleware-based Software Architectures

The different software architectures reviewed in this section adopt a middleware-based approach to provide intrusion tolerance. Middleware provides an ideal platform for intrusion tolerance extensions because it allows for a variety of applications to be built that can transparently take advantage of the intrusion tolerance properties of the middleware, eliminating the need for custom solutions for each application (Sames et al. 2002).

6.2.1 Intrusion Tolerance by Unpredictable Adaptation

The Intrusion Tolerance by Unpredictable Adaptation (ITUA) architecture is a distributed objects framework which integrates several mechanisms to enable the defense of critical applications (Pal et al. 2006). These mechanisms are based on Byzantine fault tolerance, redundancy, diversity and adaptive responses. The objective of this architecture is to enable the tolerance of sophisticated attacks aiming at corrupting a system.

A typical system made intrusion tolerant using ITUA is composed of a set of nodes which are partitioned into a set of disjointed security domains. A security domain may be composed of a single node or be composed of several nodes and it provides some boundary which is difficult to be circumvented by an attackers. An example of a typical security domain is a LAN protected with a firewall. Diversity is used to ensure that different domains do not share security vulnerabilities through, for example, the usage of different operating systems. The overall system architecture of ITUA is given in Figure 16. The middleware allows the transparent replication of application objects. It controls the number of replicas, their placement within security domains, and maintains the state consistency of replicas. The ITUA architecture supports adaptation, which may involve the reconfiguration of system resources and management of certain infrastructure mechanisms, such as firewalls. In order to manage such adaptation, the architecture uses a decentralized management system.

In ITUA-based systems, a manager process runs on every host in the ITUA system. All the managers in the system form the the manager group. Configuration decisions are made by the managers on the basis of the domain-specific information available to them and information received from other managers about their own configuration changes. Each manager carries out two main functions: security advising and replication management. The behavior of the managers is local as it impacts only the resources associated with the security domain in which the manager runs. However, some decisions made by the man-agers can have an effect on replicas running outside the security domain. Therefore, the managers need to cooperate in order to effect these changes. As a security advisor, the manager uses several local sensor-actuator loops to monitor the system for intrusions and perform quick knee-jerk reactions to some specific events when they are observed, and to provide other managers with information regarding what is being observed within their domain.

A representation of the ITUA architecture is shown in Figure 15. This representation shows also elements (e.g. fuselet, guardian) of a particular application deployed and integrated

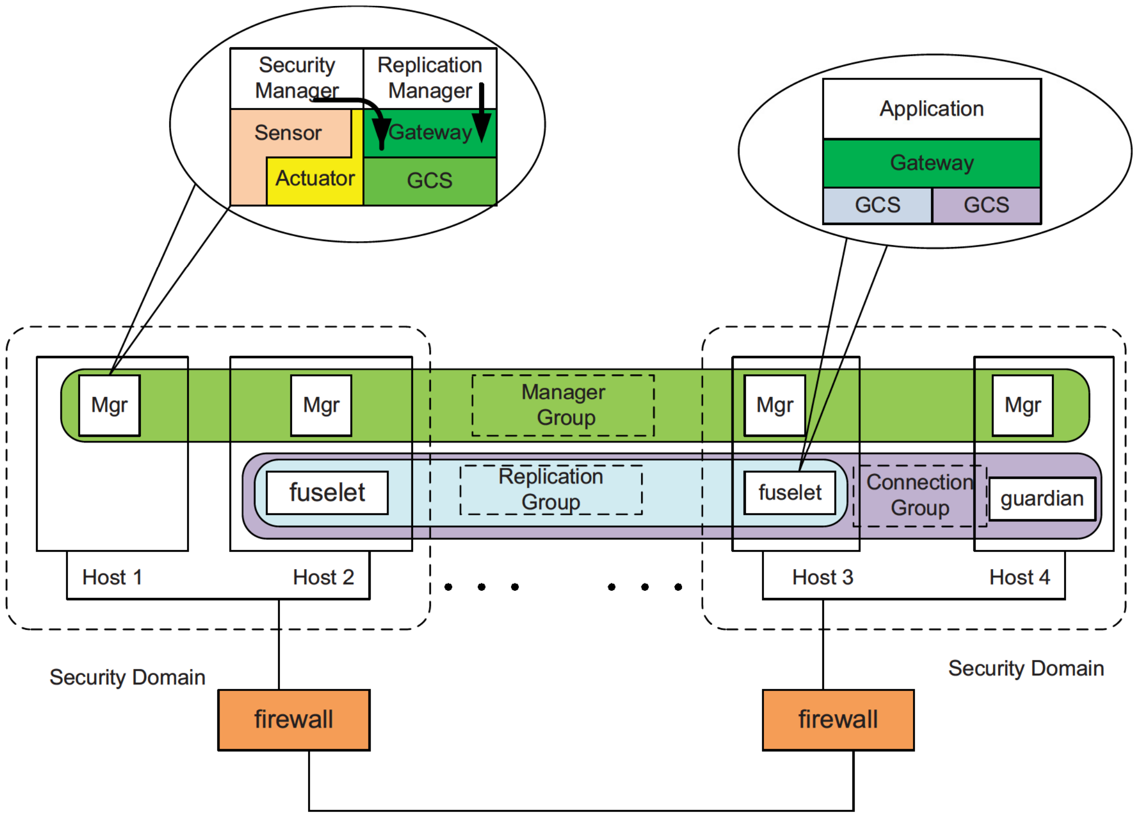


Figure 15: *ITUA Architecture*

with ITUA. This application Insertion Embedded Infosphere Support Technologies (IEIST) (D. Corman and Satterthwaite 2001), which is an advanced avionics systems integrated with the Joint Battlespace Infosphere (JBI) (USAF 1999).

ITUA relies on redundancy and diversity to enable the intrusion tolerance. Indeed, application objects (e.g. fuselets) and the architecture infrastructure elements (e.g. managers) are replicated. In addition, different forms of diversity are employed where different OS platforms and diverse application objects implementation. Moreover, ITUA injects uncertainty by adopting adaptive responses.

6.2.2 Malicious and Accidental Fault Tolerance for Internet Applications

Malicious and Accidental Fault Tolerance for Internet Applications (MAFTIA) (Veríssimo et al. 2006) is a research project supported by EU IST project. This project targeted the objective of systematically investigating the *tolerance paradigm* in order to build large-scale dependable distributed applications. The project had a comprehensive approach that included both accidental and malicious faults. The research work in the MAFTIA project

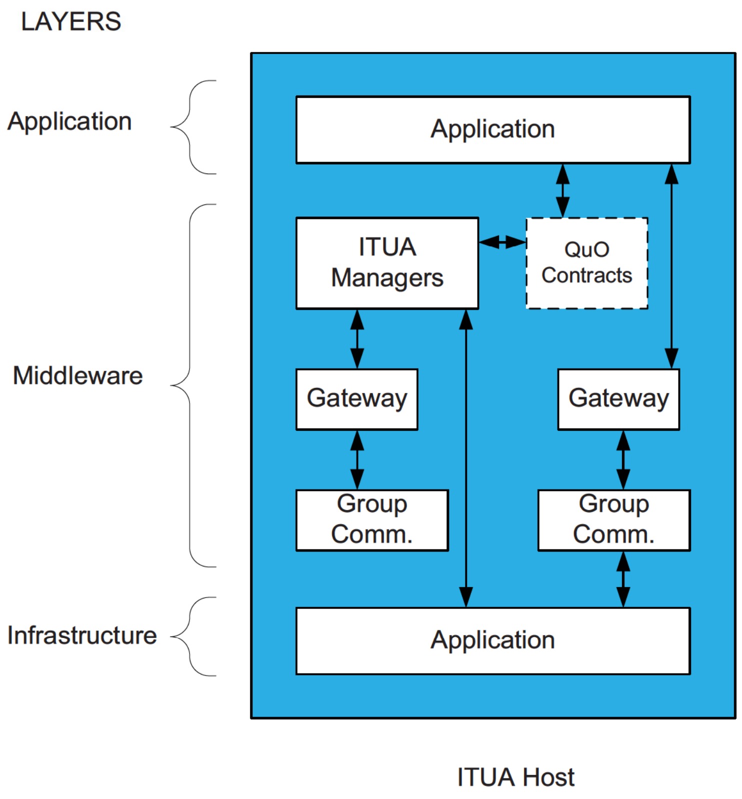


***Figure 16:** ITUA Middleware*

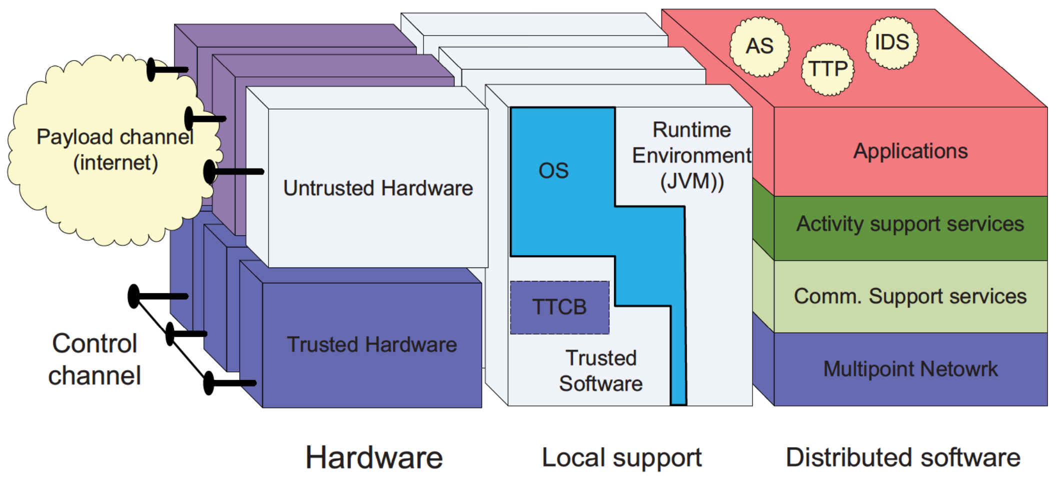


Figure 17: *The MAFTIA Architecture*

has focused in particular on the development of a coherent set of concepts that can be used to build an architecture able to tolerate malicious faults. This has been achieved through a definition and a mapping of the main intrusion tolerance concepts using the classical dependability concepts. This mapping led to the definition of the AVI composite fault model which was presented previously.

The most important contributions of the research work in MAFTIA consists in the design of intrusion-tolerant mechanisms and protocols including the the MAFTIA middleware. In particular, the intrusion detection was considered in MAFTIA as a mechanism enabling the intrusion tolerance as well as a service which needs to be made intrusion-tolerant. Therefore, MAFTIA developed a distributed intrusion-tolerant intrusion detection system and investigated the issue of handling high rates of false alarms and the correlation of the alarms generated by several IDSs. Moreover, MAFTIA generated Trusted Third Parties (TTPs) such as the design of a distributed intrusion tolerant certification authority based on threshold cryptography and intrusion-tolerant protocols as well as a distributed optimistic fair exchange service. Finally, MAFTIA research allowed definition of a distributed authorization service based on fine grained protection (i.e. at the object method call level).

The MAFTIA middleware is group-oriented. It is designed to support the communication between groups of application level software components called participants. Two levels are distinguished in the architecture of the middleware, which is shown. The participant level carry out the communication among participants and the the site level handles inter-host communication. A participant-group is multiplexed into a site-group, which is composed of all hosts where there are participants of the former group. This division in participant and site levels is the implementation of a form of clustering where a site is a cluster of participants and it aims at improving the scalability of the middleware.

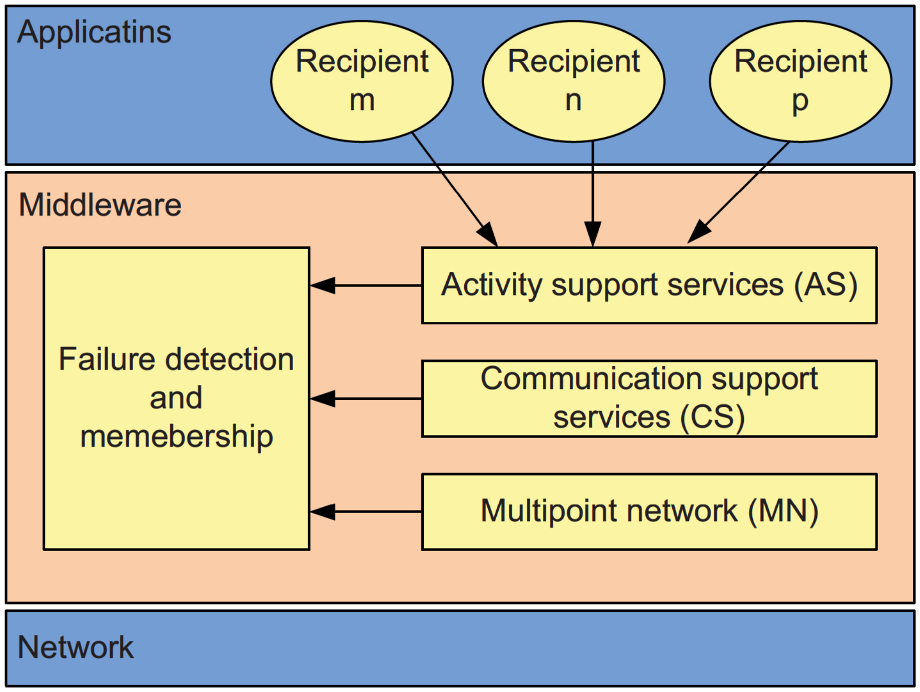

Applicatins
Recipient m
Recipient n
Recipient p
Middleware
Activity support services (AS)
Failure detection and memebership
Communication support services (CS)
Multipoint network (MN)
Network

Figure 18: *The MAFTIA Middleware*

At the site level, the Site Failure Detector module, SF, assesses the connectivity and correctness of sites. The Site Membership module, SM, depends on information given by the SF module. It creates and modifies the membership of site-groups. The Communication Support Services module, CS, implements basic cryptographic primitives, group communication with several reliability and ordering guarantees (reliable multicasts, atomic multicast), and other core services. The CS module depends on information given by the SM module about the composition of the groups, and on the MN module to access the network. At the participant level, the Participant Failure Detector module, PF, assesses the liveness and the correctness of local participants. The Participant Membership module PM function is similar to the SM, but uses the membership of participant groups. The PM uses the information generated by SM and PF modules to monitor all groups with local members.It also cooperates with the corresponding modules in the concerned remote sites. The Activity Support Services module, AS, implements building blocks that assist participant activity, such as replication and transactional management. The lowest layer of the architecture is the Multipoint Network module, MN, created over the physical network infrastructure. The objective is to provide some degree of abstraction of the specific underlying network(s) below. Its main properties are the provision of multipoint addressing and a moderate best-effort error recovery ability, both depending on topology and site liveness information.

6.2.3 Designing Protection and Adaptation into a Survivability Architecture

The Designing Protection and Adaptation into a Survivability Architecture (DPASA) (Atighetchi et al. 2005), (Chong et al. 2005) is a survivability architecture providing a diverse set of defense mechanisms. This architecture relies on a robust network infrastructure which supports redundancy and provides security services such as packet filtering, source authentication, link-level encryption and network anomaly sensors. The detection of violations triggers defensive responses provided by middleware components in the architecture. These responses effect configuration changes as well as the usage of the network fabric. In the following, we present an overview of the main elements of the DPASA architecture, which is outlined in Figure 19.

- DPASA uses redundancy in the form of four core quadrants called Quads. Each quad runs on a dedicated Local Area Network (LAN) implemented as a Virtual Local Area Network (VLAN). The LANs are connected through a layer 3 switch emulating the public IP networking infrastructure. Each LAN is protected by a VPN firewalls. The VPN hide the internal network addresses and payload content.
- The nodes in the four quads use different operating systems (SELinux, Windows and Solaris) and the clients hosts run SELinux or Solaris.
- Each node has an Autonomic Distributed Firewall Network Interface Card (ADFNIC) which performs packet filtering and encryption.
- The quads are organized into the following three zones:
  - The Executive zone which contains the management and control functions of the system. The hosts in this zone are called System Managers (denoted qXSM in Figure 19). The system managers collect system information and control the other components using adaptive algorithms
  - The operations zone which contains the nodes that carries out the main functional operations. In the example of the JBI application, these nodes implement the Publish-Subscribe-Query (PSQ) service and the supporting repository. This zone includes intrusion detection systems (qXNIDS) and a Corelator (qXCOR).
  - The crumple zone is composed of proxies hosts called Access Proxies (qXAP in Figure) which proxies the operations zone function for the clients.
- The zoning has an impact on connectivity and communication.

6.2.4 Fault/intrusiOn REmoVal through Evolution and Recovery

Fault/instrusiOn REmoVal through Evolution and Recovery (FOREVER) (Bessani et al. 2008) (Bessani et al. 2009) is a service which is used to enhance the resilience of intrusion tolerant replicated systems. FOREVER achieves this goal through the combination of recovery and evolution. FOREVER allows a system to recover from experienced malicious attacks or faults using time-triggered periodic recoveries. FOREVER service follows the hybrid system model and architecture where a system is composed of two parts (payload and wormhole). This is shown in Figure 20.

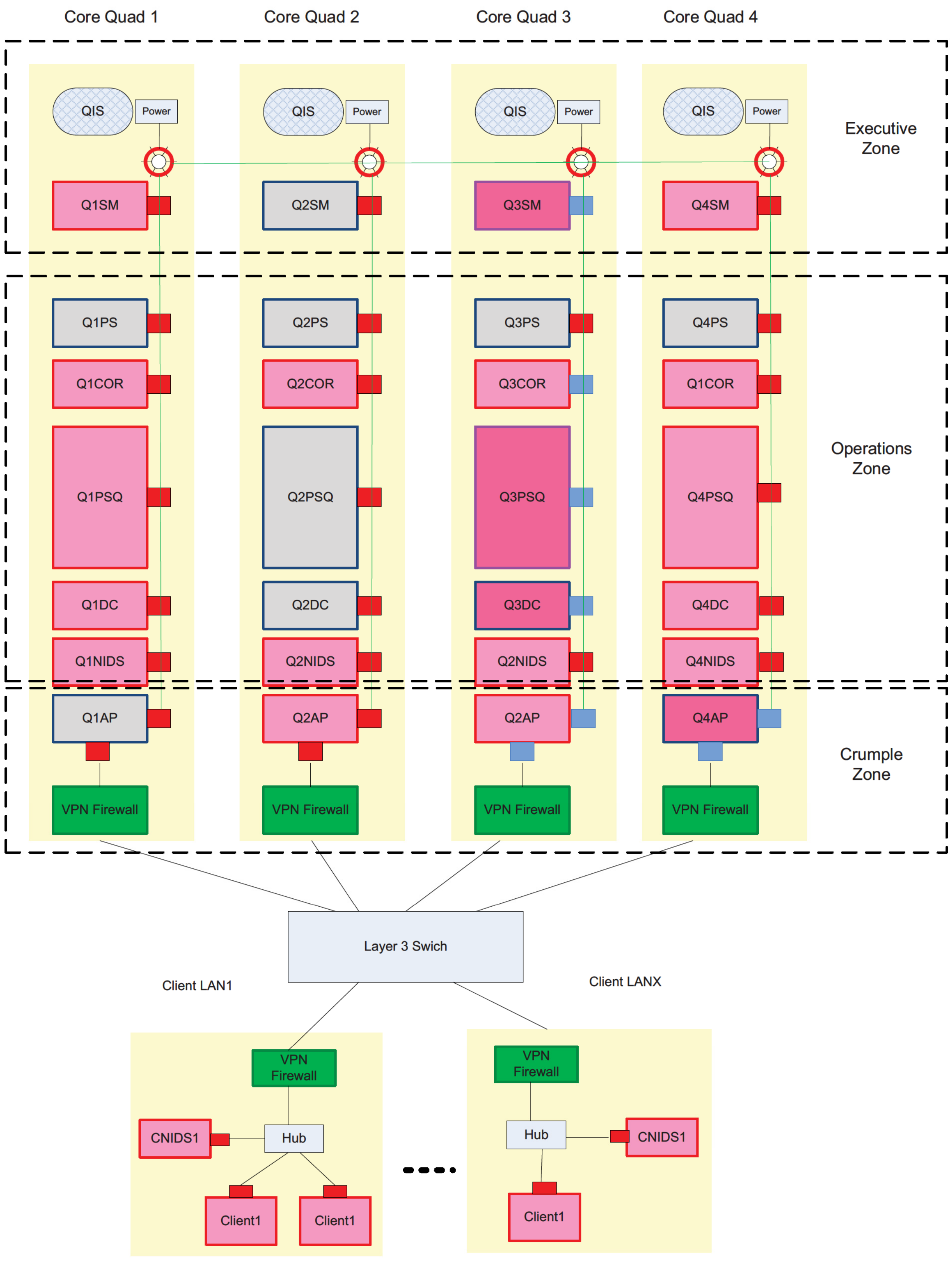


Figure 19: *DPASA Architecture*

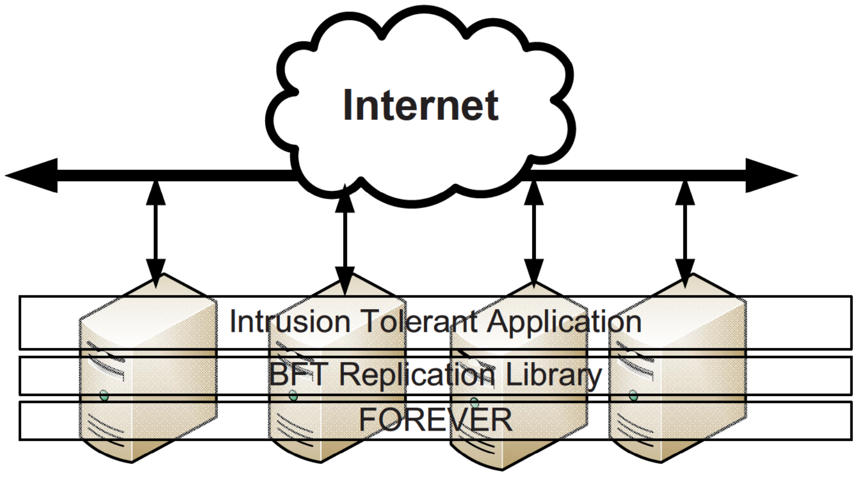


Figure 20: *FOREVER Architecture*

The intrusion-tolerant application along with the replication redundancy runs in the payload model while FOREVER service runs in the wormhole part which is guaranteed to be secure and timely.

6.2.5 Intrusion Tolerant Distributed Object System

The Intrusion Tolerant Distributed Object Systems (ITDOS) provides an architecture for a heterogeneous intrusion tolerant distributed object system (Sames et al. 2002). ITDOS integrates a Byzantine Fault Tolerant multicast protocol into an open-source Common Object Request Broker Architecture (CORBA) Object Request Broker (ORB) to provide Intrusion Tolerant middleware. Figure 21 depicts a client server architecture using ITDOS.This foun-dation allows up to *f* simultaneous Byzantine failures of replicated servers in a system of at least $3f+1$ replicas. Voting on unmarshalled CORBA messages allows heterogeneous ap-plication implementations for a given service, allowing for greater diversity in implementa-tion and greater survivability. Symmetric encryption session keys generated by distributed pseudo-random function techniques provide confidential client-server communications.

6.2.6 Secure System Architecture Based on Dynamic Resource Reallocation

This system architecture is based on a two-level approach (Min et al. 2004) (Min and Choi 2004). At the node level, dynamic resource reallocation within a computing node is used to enable preselected critical services to survive even after the occurrence of an attack. In the case that the node level adaptive actions are no more sufficient to guarantee the required resources for the services within the node, system level mechanisms for survival and intrusion tolerance are used. The system level mechanism is to deliver the intended services transparently to the clients even when a node fails by means of inter-node resource

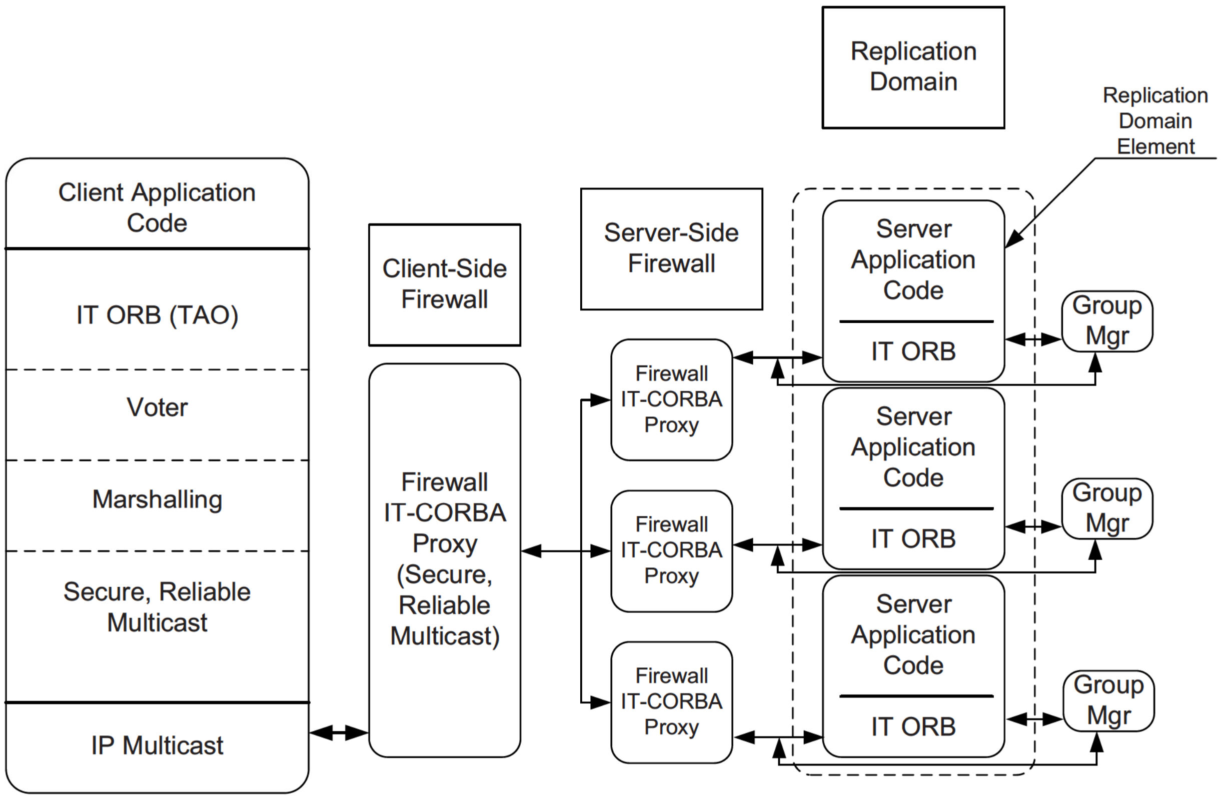


Figure 21: *ITDOS-based Client Server Architecture*

reallocation. At this level, the architecture is based on a set of diverse redundant computing nodes. This architecture is shown in Figure 21.

6.3 Other Architectures and Frameworks

In this section, we present some architectures and frameworks that do not exactly fit into the two categories presented previously. T

6.3.1 Willow Architecture

The Willow architecture provides a comprehensive architectural approach to *survivability* in distributed applications used in nation critical infrastructure such as transportation, power distribution, financial services as well as defense critical systems such as the Defence Department Global Command and Control Systems (GCCS)(Knight et al. 2001). This architecture deals with faults with an approach which includes fault avoidance, fault elimination and fault tolerance. In order to do so, this architecture integrates mechanisms to avoid faults when the system is deployed and enhanced (i.e. upgraded), mechanisms to eliminate faults when detected or suspected to prevent them from causing failure and

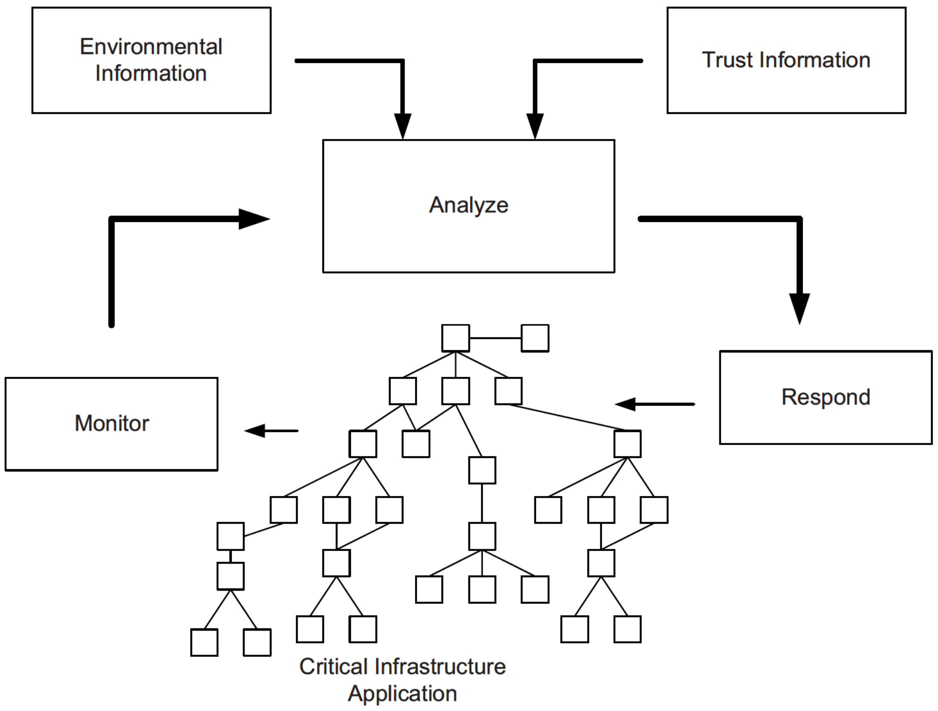

Environmental Information
Trust Information
Analyze
Monitor
Respond
Critical Infrastructure Application

Figure 22: *The Willow Architecture*

mechanisms to tolerate the effects of faults during operation. The core of this architecture is the notion of reconfiguration at the system and application level and a framework which implements a monitor/analyze/respond approach.

The Willow is based on a system architecture called the survivability architecture or infor-mation survivability control system (Sullivan et al. 1999) which is depicted in Figure 22. The main characteristic of this architecture is that it has a basic structure of control loop which is used during system operation to monitor the system, analyze it and trigger appropriate responses when the analysis reveals any problem. The Willow architecture extends the control concept by including multiple inter-operating loops.

The main components of the Willow architecture are shown in Figure 23. These are described briefly as follows:

- **Control loops:** The Willow architecture is based on a structure composed of a set of control loops. These use a set of components to achieve sensing, diagnosis, synthesis, coordination and actuation. The control loops start with a shared sensing capability within the application nodes (example of sensors can include reports from applications, application heartbeat monitors, intrusion detection alarms, etc.) The sensing events are used by the diagnosis and synthesis components to build a model of the application state and to determine the required state changes. In the current Willow architecture, there are two of these components: the Administrators Workbench for proactive reconfiguration and RAPTOR for reactive reconfiguration. Synthesis components issue their intended application changes as workflow requests, which are coordinated by the workflow and

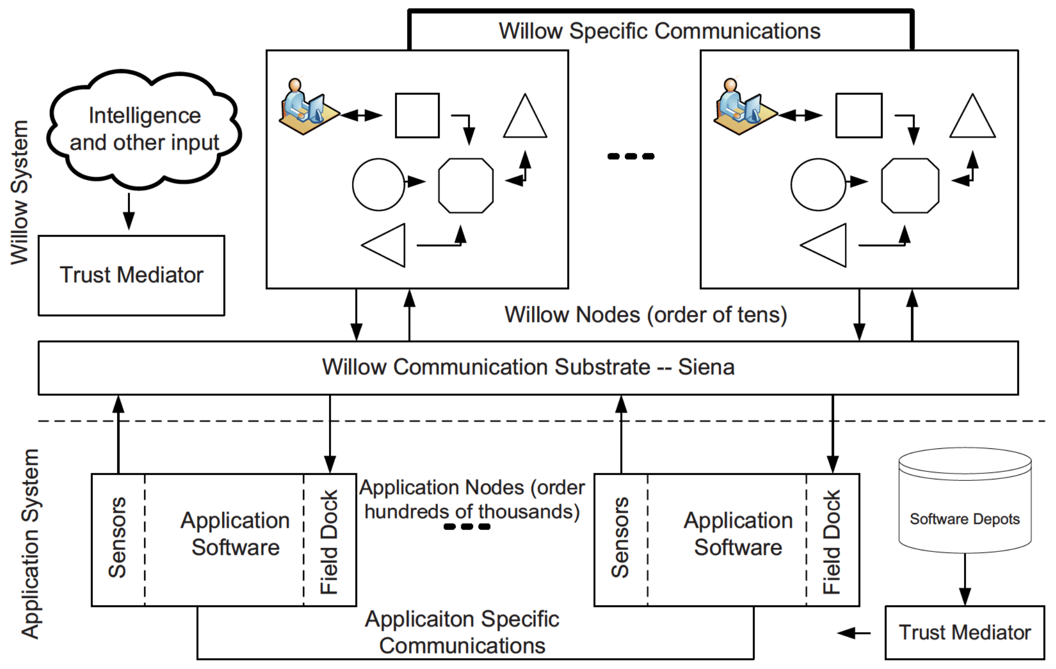

Willow Specific Communications
Willow System
Intelligence and other input
Trust Mediator
Willow Nodes (order of tens)
Willow Communication Substrate -- Siena
Application System
Sensors
Application Software
Field Dock
Application Nodes (order hundreds of thousands)
Software Depots
Trust Mediator
Applicaiton Specific Communications

Figure 23: *The Willow Architecture Components*

the resource managers to ensure their consistency. Workflow events are received by the Field Docks located within the application nodes and result in local system state changes. The Field Dock infrastructure provides a single standard interface for universal actuation at application nodes. Actuation completes the control loop cycle.

- **Proactive Control:** The proactive controller component in the Willow architecture is called the Administrative Workbench. It is an interactive application allowing system administrators to monitor system conditions remotely and initiate the proactive reconfigurations. The Software Depot shown in Figure 23 is an external source of information used by Willow in order to complete its reconfigurations. In general, there will likely be many such depots. This information may include models of new applications, components needed in a new configuration, and components that provide additional actuators allowing to access built-in reconfiguration capabilities, etc.
- **Reactive Control:** The reactive controller in the Willow architecture is called RAPTOR. It is a fully automatic structure composed of a set of finite state machines. These carry out error detection. The faults which cause errors trigger changes in the application states. Such state changes are input events to the state machine which cause a state transition. If errors are detected the new state machine state is designated as erroneous.
- **Communication**: The communication approach used by Willow is based on a highly efficient event notification service. This service constitutes a communication substrate. To this end, Willow relies on Siena (Carzaniga et al. 2001).

Since the Willow architecture is based on a control structure, its protection is therefore very important as an adversary might take control of the Willow infrastructure to breakdown the application based on Willow. There are two security concerns in the Willow architecture: the first is to secure the mechanisms of the architecture and the second is to ensure that the information used by the system is current, accurate and integral. The first is achieved by securing the Willow servers, the sensors and actuators in the applications nodes and the communication mechanisms used by Willow architecture. Several techniques are used to do so including classical security mechanisms (passwords, biometrics etc), trusted components for the sensors and actuators. In addition, temporal diversity and randomization techniques are used. One important note of the author here is the fact that the Willow architecture is not a diversity based architecture but it uses diversity to a certain extent to protect its fundamental architecture.

6.4 Protocols used in Intrusion Tolerance

The different intrusion tolerant architecture reviewed in this report uses as infrastructure some important communication protocols to enable intrusion tolerance. In this section, we summarize the most important ones.

6.4.1 Agreement and Consensus

The agreement or consensus problem is fundamental to several replication protocols (Fischer 1983). The strongest fault model that researchers consider, such as in the case of MAFTIA, is the Byzantine model. In this model some participants behave in an arbitrary manner. If communication is not authenticated and nodes are directly connected, $3f + 1$ participants are required in order to tolerate f Byzantine faults. If authentication is available, the number of participants can be reduced to $f + 2$ [10].

6.4.2 Byzantine Group Communication

An important research area which is fundamental for intrusion tolerance research is group communication system research. There are several systems that based on the concept of process groups, including Totem (Moser et al. 1996) and SecureRing (Kihlstrom et al. 1998), Horus (van Renesse et al. 1996), and Transis (Dolev and Malki 1996). There are also some commercial projects, such as Phoenix at IBM and NT Clusters at Microsoft, that are based on the concept of process groups. Projects like AQuA (Cukier et al. 1998) and Eternal (Moser et al. 1999) at UCSB have used group communication to develop higher-level abstractions to support reliable distributed computing. The ITUA and ITDOS architecture reviewed in this report use replication on top of group communication systems to achieve survivability.

6.4.3 Replication with Byzantine faults

Paxos (Lamport 1998)(Lamport 2001) is a fault-tolerant protocol. It is used to allow a set of distributed servers, which are exchanging asynchronous messages, to establish a total order of the client requests in the benign-fault crash-recovery model. Paxos uses an elected leader to coordinate the agreement protocol. In the case where the leader crashes or becomes unreachable, a new leader is elected by the other servers. Paxos requires at least $2f+1$ servers to tolerate f faulty servers. Since servers are not Byzantine, only a single reply needs to be delivered to the client. Paxos uses two asynchronous communication rounds to globally order client updates. In the first round, the leader assigns a sequence number to a client update and sends a Proposal message containing this assignment to the rest of the servers. In the second round, any server receiving the Proposal sends an Accept message, acknowledging the Proposal, to the rest of the servers. When a server receives a majority of matching Accept messages indicating that a majority of servers have accepted the Proposalit it orders the corresponding update.

Castro and Liskov (Castro and Liskov 1999) (Castro and Liskov 2002) presented a practical Byzantine fault-tolerant replication protocol called BFT. This protocol addresses the problem of replication in the Byzantine model where a number of servers can exhibit arbitrary behavior. BFT also uses an elected leader to coordinate the protocol. BFT extends Paxos into the Byzantine environment by using an additional communication round in ensure consistency both in and across views and by constructing strong majorities in each round of the protocol. In particuler, BFT uses a flat architecture and requires acknowledgments from $2f+1$ out of $3f+1$ servers to mask the behavior of f Byzantine servers. A client must wait for $f+1$ identical responses to be guaranteed that at least one correct server assented to the returned value.

BFT uses three communication rounds. In the first round, the leader assigns a sequence number to a client update and proposes this assignment to the rest of the servers by broadcasting a Pre-Prepare message. In the second round, a server accepts the proposed assignment by broadcasting an acknowledgment, Prepare. When a server collects a Prepare Certificate (i.e., it receives the Pre-Prepare and 2f Prepare messages with the same view number and sequence number as the Pre-Prepare), it begins the third round by broadcasting a Commit message.A server commits the corresponding update when it receives $2f+1$ matching commit messages.

Yin et al. (Yin et al. 2003) propose to separate the agreement component which carry out the requests ordering from the execution component which processes the requests. This separation allows the same agreement component to be used for many different replication tasks and reduces the number of execution replicas to $2f+1$. Martin and Alvisi (Martin and Alvisi 2006) introduced a two-round Byzantine consensus algorithm, which uses $5f+1$ servers in order to overcome f faults. This approach trades lower availability

(4 f + 1 out of $5f + 1$ connected servers are required, instead of $2f + 1$ out of $3f + 1$ as in BFT), for increased performance. The solution is appealing for local area networks with high connectivity. The ShowByz system of Rodrigues et al. (Rodrigues et al. 2007) seeks to support a large-scale deployment consisting of multiple replicated objects. ShowByz modifies BFT quorums to tolerate a larger fraction of faulty replicas, reducing the likelihood of any group being compromised at the expense of protocol liveness. Zyzzyva (Kotla et al. 2007) uses speculative execution to reduce the cost of Byzantine fault-tolerant replication when there are no faulty replicas.

7 Diversity Modeling and Analysis

In this section, we consider research work which has proposed models and analysis techniques to enable the evaluation of the security of software systems built using the principle of redundancy with diversity. In our study, we included not only research related to security but also reliability. This is because models and techniques proposed to evaluate the security have built upon models and techniques devoted initially to reliability. In all the studies, the probabilistic approach dominates, because of the high level of uncertainty. From the reliability perspective, we deal with the uncertainty of the faults hidden in the system and with the uncertainty of the manifestation of those faults as errors and eventually as failures. Similarly, from a security perspective, we deal with uncertainty due to vulnerability and residual design faults which might be exploited and lead to intrusion. Uncertainty is also related to malicious behavior. For these reasons the probabilistic approach dominates as the general approach used to model and analyze software systems.

7.1 Probabilistic Modeling of Diversity-based Security

Littlewood and Strigini discuss in (Littlewood and Strigini 2004) the relevance of redundancy and diversity not only for reliability but also for security. In this paper, the authors shows that probability is the appropriate formalism for modeling diversity based security approaches because of the uncertainty involved. This uncertainty from the attacker standpoint lies in his uncertain knowledge about the system. From the system owner/protector, on the other hand, the uncertainty stems from the attacker behavior. He doesn't know when and how the attack is going to occur. Therefore, the process of novel intrusion into a system is a stochastic process.

7.1.1 Eckhardt and Lee (EL) model

The EL model (Eckhardt and Lee 1985) captures two sources of uncertainty. On one hand, the random selection of demands (input) from the demand space. This selection can be represented by a probability distribution over the demand space. On the other hand, the randomness is associated with the program development (creation). This is captured as a selection from the population of all programs that could be written to solve the problem at hand. This random selection follows a probability distribution.

The key variable in this model is the difficulty function $\theta(x)$, which is the probability that a program selected randomly, via the probability distribution over all the programs, will fail on a particular input x. In other words, among a big number of programs selected independently, $\theta(x)$ is the proportion of those which fail over the input x. This difficulty varies over the input space. Therefore, for a randomly selected input, the difficulty is also a random variable. The unreliability of a randomly selected program is therefore:

$$P(randomlyselectedprogramfailsonrandomlyselectedinput) = E_X(\theta(X)) \tag{1}$$

Considering the independent development of two programs π_1 and π_2. In EL model this is the random selection of π_1 and π_2. It is shown in [] that for a particular input x:

$$P(\pi_1 and \pi_2 bothfail|inputx) = P(\pi_1 fails|inputisx).P(\pi_2 fails|inputisx) = [\theta(x)]^2 \tag{2}$$

Or equivalently:

$$P(\pi_2 failsonx|\pi_1 failsonx) = P(\pi 2 failsonx) = \theta(x) \tag{3}$$

Therefore, there is a conditional independence in the failure behavior of the programs: they are independent for any given (known) input x. The most important result of the EL model is that even if the versions are developed independently to achieve this unconditional independence, the versions will, however, fail dependently in the case where the input is selected randomly (unknown). The probability that a randomly selected pair of programs both fail on a randomly selected input X is:

$$\Sigma_x P(X=x).P(\pi_1, \pi_2 bothfail|inputisx) = E_X([\theta(X)^2) = Var_X(\theta(X)) + [E_X(\theta(X)]^2 \tag{4}$$

This probability represents then the unreliability of a system composed of two randomly selected programs (i.e. an 1-out-of-2 system). This result shows that independently developed programs do not fail independently when executing random inputs. Indeed, the first term of the above equation is based on the incorrect assumption of independence which would underestimate the probability of simultaneous failures by the amount of $Var_X(\theta(X))$

7.1.2 Littlewood and Miller (LM) Model

The EL model implies natural diversity as the independent program development is represented by a random selection of programs from a population. An alternative approach is based on "enforcing" diversity in the development of the different versions through the use of different programming languages, testing techniques, etc.
The LM model (Littlewood and Miller 1989) is a generalization of the EL by taking into consideration forced diversity. This is achieved using different distributions over the population of all the programs. As a consequence of these different distributions over the programs, there are different difficulty functions induced over the input space.
The probability of a program randomly selected using a methodology A to fail on an input x is denoted $\theta_A(x)$ and the probability of a randomly selected A program failing on a randomly selected input is $E_x(\theta_A(X))$

The EL is a special case of the LM model when $\theta_A(x) = \theta_B(x) \forall x$

Difficulty functions may differ according to the methodology (i.e., for x such that $\theta_A(x)$ is large, $\theta_B(x)$ is small). This is the case where the difficulty functions are negatively correlated.

It is shown that for any particular input x, two programs πA and πB, independently developed using respectively a methodology A and B, fail independently:

$$P(\pi_A, \pi_B failonx) = \theta_A(x).\theta_B(x) \tag{5}$$

or equivalently,

$$P(\pi_B failsonx|\pi_A failsonx) = P(\pi_B failsonx) = \theta_B(x) \tag{6}$$

The unconditional probability of a randomly selected pair of programs using the methodologies A and B both failing on a randomly selected input x is:

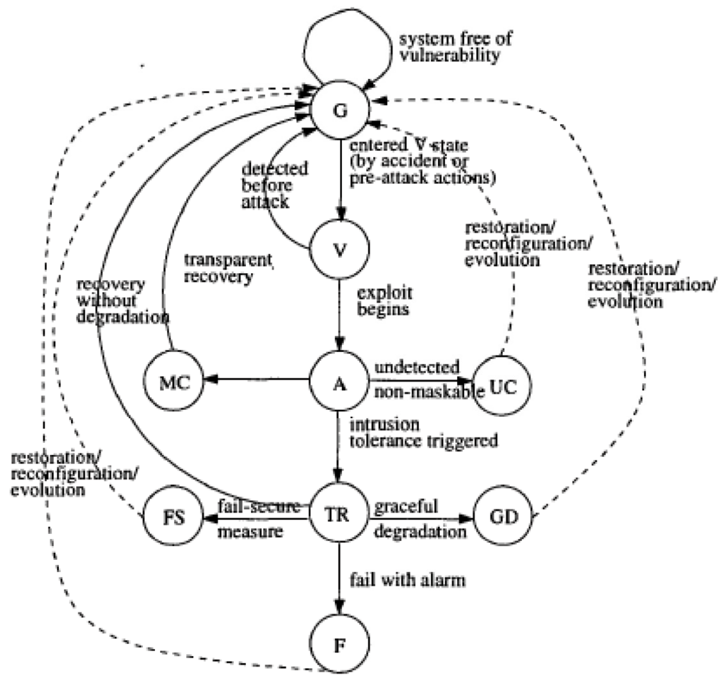
$$E_x(\theta_A(X)\theta_B(X)) = E_x(\theta_A(X)]E_X[\theta_B(X)] + Cov(\theta_A(X)\theta_B(X)) \tag{7}$$

The first term of the right hand side part of the above equation is similar to the EL model and it represents the result one gets when the independence failure is assumed. The second term,$Cov(\theta_A(X)\theta_B(X))$, is the covariance of the random variables A and B. This is where the difference between the EL and LM models lies. Indeed, the covariance term might be negative or positive. Therefore, the probability of failure of both randomly selected versions might even be lower that in the independence case.

The intuitive idea underlying the LM model is that the difficulty functions differ from one development process to another. The correlation between the difficulty functions might then be negative. Therefore a diversified system (1-out-of-2 system) might have a reliability even greater than one that could be reached under the assumption of independence (Littlewood et al. 2001). Although this might be difficult to achieve in practice, the LM model is still in favor of the diversity as a way to achieve reliability.

7.2 Intrusion Tolerance Dynamic Behavior Model

Geoseva et al. present (Goseva-Popstojanova et al. 2001) a State Transition Model (STM) to describe the dynamic behavior of intrusion tolerant systems.This model is shown in Figure 24. The aim of this model is is to provide a framework which can be used to define the set of vulnerabilities and threats that should be dealt with by the system. In this work, several known vulnerabilities are mapped to this model. It is not, however, a description of the behavior of a specific intrusion tolerant model. See State transition diagrams in Figure 24, below . (Wang et al. 2003a)


system free of vulnerability
G
entered V state (by accident or pre-attack actions)
detected before attack
V
transparent recovery
recovery without degradation
exploit begins
restoration/ reconfiguration/ evolution
restoration/ reconfiguration/ evolution
MC
A
undetected non-maskable
UC
intrusion tolerance triggered
restoration/ reconfiguration/ evolution
FS
fail-secure measure
TR
graceful degradation
GD
fail with alarm
F

G	good state	TR	triage state
V	vulnerable state	FS	fail-secure state
A	active attack state	GD	graceful degradation state
MC	masked compromised state	F	failed state
UC	undetected compromised state		

Figure 24: *State Transition Model of an Intrusion Tolerant System*

7.3 Effectiveness of Diversity for Security

There is very limited previous research focused on evaluating the actual effectiveness of implementing diversity in order to achieve software system security. We have identified two recent studies which contributed to address this question. In this section, we describe this work.

7.4 Methodologies and Techniques for Security Evaluation of Diversity-based Software Systems

7.4.1 Qualitative Evaluation Approaches

Stroud et al. present (Stroud et al. 2004) a qualitative analysis technique applied to the MAFTIA architecture. In this work, the author used a case study of an Internet application using MAFTIA. The authors have shown how the services of the MAFTIA middleware make the system survivable under typical attack scenarios. These malicious attacks are systematically dissected using fault-tree analysis.

7.4.2 Quantitative Evaluation Approaches

In (Madan et al. 2004), the authors address the issue of quantifying the security attributes of an intrusion tolerant system. Basically, the approach proposed is to consider a security intrusion and the response of an intrusion tolerant system as a random process. Therefore taking advantage of stochastic modeling techniques to capture the behavior of the attacker and the system response. In this work, a metric called the Mean Time To Security Failure is computed.

The Mean Time To Security Failure (MTTSF) metric was first proposed (Ortalo et al. 1999) for the quantification of software system security. The approach used to compute this value is based on the transformation of a privilege graph into a Markov chain whose states denote the enhanced privileges that an attacker gains, which is also the progressive deterioration towards the security failed states, through a series of atomic attacks on a system. The arcs on the other hand, represent the effort *e* spent by an attacker to cause state transitions in this Markov chain. This effort in injecting an atomic attack is modeled as a random variable with exponential distribution function $P(e) = 1 - exp(-\lambda e)$, where $1/\lambda$ is the mean effort to succeed in a given elementary attack. This model is then used to evaluate the λe proposed measure of operational security mean effort to security failure, analogous to mean time to failure.

Fujimoto et al. recently presented (Fujimoto et al. 2009) a technique to evaluate quantitative security measures using Makov Regenerative Stochastic Petri Nets (MRSPNs).

8 Related Work on Diversity

We devote this section to a brief review of the patents and/or patent applications which touch on the subject of intrusion tolerance and system survivability in general.

8.1 Dependable Diverse SQL Server Architecture

Gashi et al. (Gashi et al. 2007) report the findings of two studies meant to assess the potential dependability gains from using a diverse set of off-the-shelf database servers. The studies are based on bug reports for four popular database products. The first study (Gashi et al. 2004a) was based on four DBMS products: two commercial (Oracle 8.0.5 and Microsoft SQL Server 7, without any service packs applied) and two open source ones (PostgreSQL Version 7.0.0 and Interbase Version 6.0). The diversity was also at the level of the operating system as Interbase, Oracle, and MSSQL were all run on the Windows 2000 Professional operating system, whereas PostgreSQL 7.0.0 (not available for Windows) was run on RedHat Linux 6.0. The second study (Gashi et al. 2004b) considered the later releases of the open source DBMS products.
The authors state that for most bugs, a simple configuration of a system composed of two diverse DBMS products would detect the failures. In addition, the second study, which is based on later releases of the same products confirmed the general conclusions of the first study.

8.2 Experimental Study of Diversity with Off-The-Shelve Anti-Virus Engines

The empirical study reported in (Gashi et al. 2009) provides an analysis of potential gains in terms of detection capability using diverse anti-virus products for the detection of self-propagating malware. This study is based on a real-world dataset of 1599 malware samples gathered by using SGNET, which is a distributed honeypot (Leita and Dacier 2008). Only the signature-based detection engine, which is the common type of component in most anti-virus products, is used and any alarm message issued by this component is assumed to be a successful detection regardless of the correctness of the alarm message. The evolution of the detection capability as well as the impact of diversity on this capability are studied using an engine of 32 different anti-virus products.

The main findings of the analysis can be summarized as follows: the experiments with 1-out-of-2 pairs of engines showed that the detection capability resulting from the usage of diversity is improved significantly. Almost a third of the resulting pairs achieved a better detection rate compared with the best single engine. One particular pair build using free detection engines showed a perfect detection rate for the malware collected in the study and four other pairs of free detection engines had higher detection capability than the best single engine.

According to the authors of the present study, one of the remaining areas in need of further research is the study of malicious executable files only. Other malware vectors also need to be considered, such as media files. Moreover, the present study was limited to an analysis of diversity using 1-out-2 systems of anti-virus engines. This leaves a room for the analysis of other configurations such as 2-out-3 or more complex combinations of diverse AV products.

8.3 CloudAV: Diversity-based Anti-virus Platform

Jon Oberheide et al. (Oberheide et al. 2008) proposed a new model for the detection of mal-ware by providing an in-cloud network antivirus service. This work presents a technique called N-version protection, leveraging the power of diverse detection engines working in parallel to perform the identification of malicious and unwanted software. It is claimed that this approach presents several advantages, including better detection of malicious software, enhanced forensics capabilities, retrospective detection, and improved deployability and management.

The authors implemented this technique in the form of an in-cloud antivirus system called CloudAV. The latter is composed of a lightweight cross-platform host agent and a network service with ten antivirus engines and two behavioral detection engines. Figure 25 shows the architecture of CloudAV approach. The authors evaluated the performance, scalability, and efficacy of the system using data from a real-world deployment lasting more than six months and a database of 7220 malware samples covering a one year period. The findings of this evaluation seem to show that the detection coverage of CloudAV is 35% higher than what one would get with a single antivirus engine and that the overall detection rate over the dataset is 98%.

The authors also present two case studies to demonstrate how the forensics capabilities of CloudAV were used by operators during the deployment.

8.4 TCP Protocol Parameter Diversity for Communication Security

Barrantes and Forrest present (Barrantes and Forrest 2006) a diversity based technique to defeat denial of service attacks. This technique is based on randomizing congestion control parameters of the Transmission Control Protocol (TCP) (Allman et al. 1999) to increase the unpredictability of the timing properties of the protocol. This enables mitigat-ing certain attacks, such as the *shrew* attack (Kuzmanovic and Knightly 2003), which is

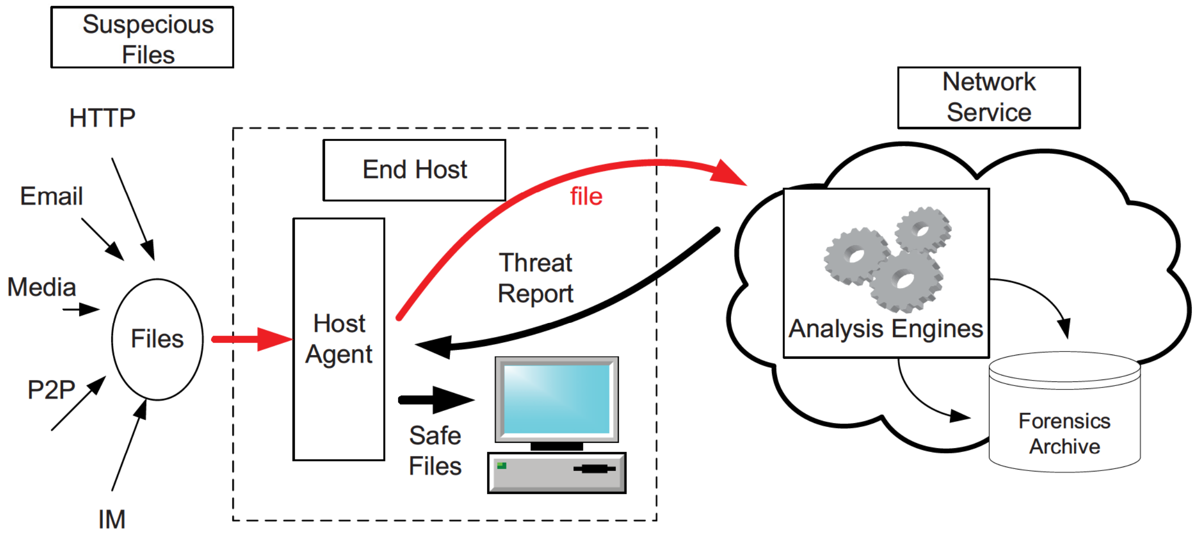


Figure 25: *Architectural Approach of CloudAV*

based on slowing down TCP to achieve a denial of service attack. The technique is to diversify the protocol parameters instead of generating diverse implementations of the protocol.

8.5 Related Patents

In this section, we describe succinctly the patent or patent applications related to intrusion tolerance or diversity as defense mechanisms. We provide the reader with the corresponding references for further detail.

8.5.1 Intrusion Tolerant Communication Networks and Associated Methods

This patent (Goseva-Pospstojanova et al. June. 11, 2002) describes an intrusion tolerant communication method and a set of associated methods. This patent aims at a continuity of operation and the survival of communication network to attacks. The intrusion tolerance is defined using various state transitions.

8.5.2 Diversity-based Security System and Method

In this US patent application (Li and Just Jul. 12, 2007) a method based on artificially introduced diversity is defined to provide a suitable defense against the threat posed by the identical vulnerabilities across software mono-culture which fosters large scale attacks against systems. The method described in this patent application supports address-space randomization of the Windows OS.

9 Conclusions and Future Work

Software systems security is a critical issue because of the increasing complexity of these systems, their connectivity and the significant similarity in the software used in such systems. This is called IT mono-culture.

The mitigation of this issue consists in using diversity which aims at reducing the common vulnerabilities and consequently increasing the difficulty of breaking systems built with diversity in mind.

The state of the art reported in this document shows that for security purposes the principle of diversity can be applied using three main approaches:

- An automated diversity approach, which consists in applying automated procedures to randomize either the code, the address space layout, or both, to provide a probabilistic defense against unknown threats.
- Diversity-based behavior monitoring and intrusion detection
- Diversity-based architectures for intrusion tolerance

Several issues related to the usage of diversity for security purposes are still in need of further investigation. In the following, we discuss some of these options:

- Software architecture based on diversity for intrusion detection or tolerance needs to be expressed using standard software modeling language. UML is the *de facto* standard in this regard. However, *UML does not specifically support diversity modeling*. UML is fortunately designed with built-in extensibility mechanisms. Therefore, it is interesting to investigate the possibility of extending UML to enable the modeling of diversity in soft-ware system architecture. This can be achieved, for instance, through the definition of a specific UML profile for diversity modeling. This can alternatively be achieved through the extension of an existing well-established UML profile. In general, the process of defining the profile will require the definition of a metamodel capturing the main concepts relevant for diversity modeling. In addition, the main constraints on the meta-model will need to be expressed formally using a language such as OCL. Finally, the main concepts of the metamodel would be mapped to the UML metamodel to define the new stereotypes that will be used by the designer.

- It is important to support the control and management of a software system which uses diversity to detect and tolerate cyber-attacks. This control and management can potentially be achieved through the definition of different configurations. The modeling and analysis of such configurations is therefore crucial. An important aspect of such configurations is a model of diversity that would permit a choice among different diversity settings. As far we know, this concept of a model of diversity does not yet exist in the state of the art. It might be similar to the redundancy models defined in the high availability purposes. This thread of research might result in the definition of an information model for the configuration of redundant and diverse software architecture.

- Linux kernel comprehensive monitoring is challenging. It yields massive traces, which are very difficult to deal with (e.g., difficult to abstract correctly to arrive at systematically meaningful information). The principle of diversity can potentially help with this issue. The idea is to deploy a set of redundant Linux nodes running in parallel including a subset of the replicas that might be deliberately vulnerable. All the replicas are monitored differently (i.e. using diversity). Indeed, the focus of each Linux kernel replica is turned towards different (predetermined) perspectives. These include the main kernel services such as memory management, file system management, networking sockets, interrupts, etc. This monitoring configuration would yield *a diverse* set of much more lightweight traces. Several research questions need to be studied with regard to the monitoring setting, including the correlation of different lightweight traces in a healthy system, the correlation of the traces of a healthy system with a compromised one and the identification of patterns.

References

Allman, M., Paxson, V., and Stevens, W. (1999), TCP Congestion Control. RFC #2581.

Atighetchi, Michael, Rubel, Paul, Pal, Partha Pratim, Chong, Jennifer, and Sudin, Lyle (2005), Networking Aspects in the DPASA Survivability Architecture: An Experience Report, In *Fourth IEEE International Symposium on Network Computing and Applications (NCA 2005)*, pp. 219–222, IEEE Computer Society.

Avizienis, Algirdas, Laprie, Jean-Claude, Randell, Brian, and Landwehr, Carl E. (2004), Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Trans. Dependable Sec. Comput.*, 1(1), 11–33.

Bain, Charles, Faatz, Donald B., Fayad, Amgad, and Williams, Douglas E. (2001), Diversity as a defense strategy in information systems. Does evidence from previous events support such an approach?, In Gertz, Michael, Guldentops, Erik, and Strous, Leon, (Eds.), *Fourth Working Conference on Integrity, Internal Control and Security in Information Systems, IICIS'01*, Vol. 211 of *IFIP Conference Proceedings*, pp. 77–94, Kluwer.

Barbacci, Mario (1996), Survivability in the age of vulnerable systems, *Computer*, 29(11), 8.

Barrantes, Elena G. and Forrest, Stephanie (2006), Increasing Communications Security through Protocol Parameter Diversity, In *the XXXII Latin-American Conference on Informatics (CLEI 2006)*, Santiago, Chile.

Barrantes, Elena Gabriela, Ackley, David H., Palmer, Trek S., Stefanovic, Darko, and Zovi, Dino Dai (2003), Randomized instruction set emulation to disrupt binary code injection attacks, In Jajodia, Sushil, Atluri, Vijayalakshmi, and Jaeger, Trent, (Eds.), *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp. 281–289, ACM.

Bessani, A., Daidone, A., Gashi, I., Obelheiro, R., Sousa, P., and Stankovic, V. (2009), Enhancing Fault / Intrusion Tolerance through Design and Configuration Diversity, In *3rd Workshop on Recent Advances on Intrusion-Tolerant Systems WRAITS 2009*, Estoril, Lisbon, Portugal.

Bessani, Alysson Neves, Reiser, Hans P., Sousa, Paulo, Gashi, Ilir, Stankovic, Vladimir, Distler, Tobias, Kapitza, Rüdiger, Daidone, Alessandro, and Obelheiro, Rafael R. (2008), FOREVER: Fault/intrusiOn REmoVal through Evolution & Recovery, In Douglis, Fred, (Ed.), *ACM/IFIP/USENIX 9th International Middleware Conference*, pp. 99–101, ACM.

Bhatkar, Sandeep and Sekar, R. (2008), Data Space Randomization, In Zamboni, Diego, (Ed.), *Detection of Intrusions and Malware, and Vulnerability Assessment, 5th*

International Conference, DIMVA 2008, Vol. 5137 of *Lecture Notes in Computer Science*, pp. 1–22, Springer.

Carzaniga, Antonio, Rosenblum, David S., and Wolf, Alexander L. (2001), Design and evaluation of a wide-area event notification service, *ACM Trans. Comput. Syst.*, 19(3), 332–383.

Castro, Miguel and Liskov, Barbara (1999), Practical Byzantine fault tolerance, In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pp. 173–186, Berkeley, CA, USA: USENIX Association.

Castro, Miguel and Liskov, Barbara (2002), Practical byzantine fault tolerance and proactive recovery, *ACM Trans. Comput. Syst.*, 20(4), 398–461.

Chew, M. and Song, D. (2002), Mitigating Buffer Overflows by Operating System Randomization, (Technical Report CMU-CS-02-197) Carnegie Mellon University.

Chong, Jennifer, Pal, Partha Pratim, Atighetchi, Michael, Rubel, Paul, and Webber, Franklin (2005), Survivability Architecture of a Mission Critical System: The DPASA Example, In *21st Annual Computer Security Applications Conference (ACSAC 2005)*, pp. 495–504, IEEE Computer Society.

Cox, Benjamin, Evans, David, Filipi, Adrian, Rowanhill, Jonathan, Hu, Wei, Davidson, Jack, Knight, John, Nguyen-Tuong, Anh, and Hiser, Jason (2006), N-variant systems: a secretless framework for security through diversity, In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA: USENIX Association.

Cukier, Michel, Ren, Jennifer, Sabnis, Chetan, Henke, David, Pistole, Jessica, Sanders, William H., Bakken, David E., Berman, Mark E., Karr, David A., and Schantz, Richard E. (1998), AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects, In *SRDS '98: Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, p. 245, Washington, DC, USA: IEEE Computer Society.

D. Corman, T. Herm and Satterthwaite, C. (2001), Transforming legacy systems to obtain information superiority, In *Proceedings of the Sixth International Command and Control Research and Technology Symposium (6th ICCRTS)*, Department of Defense CCRP.

Deswarte, Y., Kanoun, K., and Laprie, J.-C. (1998), Diversity against accidental and deliberate faults, In Ammann, P., Barnes, B. H., Jajodia, S., , and Sibley, E. H., (Eds.), *Computer Security, Dependability, and Assurance: From Needs to Solutions*, p. 171181, Williamsburg, VA, USA: IEEE Computer Press.

Deswarte, Yves and Powell, David (2004), Intrusion tolerance for Internet applications, In *Building the Information Society, IFIP 18th World Computer Congress,*, pp. 241–256.

Dolev, Danny and Malki, Dalia (1996), The Transis approach to high availability cluster communication, *Commun. ACM*, 39(4), 64–70.

Eckhardt, D. E. and Lee, L. D. (1985), A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors, *IEEE Trans. Softw. Eng.*, 11(12), 1511–1517.

emerging risks team, Lloyds (2009), Digital Risks: Views of a Changing Risk Landscape, (Technical Report Volume XIV) Lloyd's.

Fischer, Michael J. (1983), The Consensus Problem in Unreliable Distributed Systems (A Brief Survey), In *Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory*, pp. 127–140, London, UK: Springer-Verlag.

Forrest, Stephanie, Somayaji, Anil, and Ackley, David H. (1997), Building Diverse Computer Systems, In *Workshop on Hot Topics in Operating Systems*, pp. 67–72.

Fray, J.-M., Deswarte, Yves, and Powell, David (1986), Intrusion-Tolerance Using Fine-Grain Fragmentation-Scattering, In *IEEE Symposium on Security and Privacy*, pp. 194–203.

Fujimoto, Ryutaro, Okamura, Hiroyuki, and Dohi, Tadashi (2009), Security Evaluation of an Intrusion Tolerant System with MRSPNs, In *Proceedings of the The Forth International Conference on Availability, Reliability and Security (ARES 2009)*, pp. 427–432, IEEE Computer Society.

Gao, Debin, Reiter, Michael K., and Song, Dawn Xiaodong (2006), Behavioral Distance for Intrusion Detection, In Valdes, Alfonso and Zamboni, Diego, (Eds.), *Recent Advances in Intrusion Detection, 8th International Symposium, RAID'2005*, Vol. 3858 of *Lecture Notes in Computer Science*, pp. 63–81, Springer.

Gao, Debin, Reiter, Michael K., and Song, Dawn Xiaodong (2006), Behavioral Distance Measurement Using Hidden Markov Models, In Zamboni, Diego and Krügel, Christopher, (Eds.), *Recent Advances in Intrusion Detection, 9th International Symposium, RAID'06*, Vol. 4219 of *Lecture Notes in Computer Science*, pp. 19–40, Springer.

Gao, Debin, Reiter, Michael K., and Song, Dawn Xiaodong (2009), Beyond Output Voting: Detecting Compromised Replicas Using HMM-Based Behavioral Distance, *IEEE Trans. Dependable Sec. Comput.*, 6(2), 96–110.

Gashi, Ilir, Popov, Peter T., Stankovic, Vladimir, and Strigini, Lorenzo (2004), On Designing Dependable Services with Diverse Off-the-Shelf SQL Servers, In de Lemos, Rogério, Gacek, Cristina, and Romanovsky, Alexander B., (Eds.), *Architecting Dependable Systems II - ICSE 2003 Workshop on Software Architectures for Dependable Systems*, Vol. 3069 of *Lecture Notes in Computer Science*, pp. 191–214, Springer.

Gashi, Ilir, Popov, Peter T., and Strigini, Lorenzo (2004), Fault Diversity among Off-The-Shelf SQL Database Servers, In *International Conference on Dependable Systems and Networks (DSN 2004)*, pp. 389–398, IEEE Computer Society.

Gashi, Ilir, Popov, Peter T., and Strigini, Lorenzo (2007), Fault Tolerance via Diversity for Off-the-Shelf Products: A Study with SQL Database Servers, *IEEE Trans. Dependable Sec. Comput.*, 4(4), 280–294.

Gashi, Ilir, Stankovic, Vladimir, Leita, Corrado, and Thonnard, Olivier (2009), An Experimental Study of Diversity with Off-the-Shelf AntiVirus Engines, In *NCA '09: Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, pp. 4–11, Washington, DC, USA: IEEE Computer Society.

Giffin, Jonathon T., Jha, Somesh, and Miller, Barton P. (2006), Automated Discovery of Mimicry Attacks, In Zamboni, Diego and Krügel, Christopher, (Eds.), *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006*, Vol. 4219 of *Lecture Notes in Computer Science*, pp. 41–60, Springer.

Goseva-Popstojanova, K., Wang, Feiyi, Wang, Rong, Gong, Fengmin, Vaidyanathan, K., Trivedi, K., and Muthusamy, B. (2001), Characterizing intrusion tolerant systems using a state transition model, In *DARPA Information Survivability Conference & Exposition II, 2001. DISCEX '01. Proceedings*, Vol. 2, pp. 211–221.

Goseva-Pospstojanova, Katerina, Wang, Feiyi, Wang, Rong, Gong, Fengmin, Vaidyanathan, Kalyanaraman, Trivedi, Kishor, and Muthusamy, Balamurugan (June. 11, 2002), Intrusion Tolrant Communication Networks and Associated Methods. U.S. Patent #7,350,234 B2 issued Mar.25, 2008.

Hiltunen, Matti A., Schlichting, Richard D., and Ugarte, Carlos A. (2003), Building Survivable Services Using Redundancy and Adaptation, *IEEE Trans. Computers*, 52(2), 181–194.

Joseph, M.K. and Avizienis, A. (1988), A fault tolerance approach to computer viruses, In *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on*, pp. 52–58.

Just, James E. and Cornwell, Mark R. (2004), Review and analysis of synthetic diversity for breaking monocultures, In Paxson, Vern, (Ed.), *Proceedings of the 2004 ACM Workshop on Rapid Malcode, WORM'2004*, pp. 23–32, ACM Press.

Kc, Gaurav S., Keromytis, Angelos D., and Prevelakis, Vassilis (2003), Countering code-injection attacks with instruction-set randomization, In Jajodia, Sushil, Atluri, Vijayalakshmi, and Jaeger, Trent, (Eds.), *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp. 272–280, ACM.

Keromytis, Angelos D. (2009), Randomized Instruction Sets and Runtime Environments Past Research and Future Directions, *IEEE Security and Privacy*, 7(1), 18–25.

Kienzle, Jörg (2003), Software Fault Tolerance: An Overview, In Rosen, Jean-Pierre and Strohmeier, Alfred, (Eds.), *Reliable Software Technologies - Ada-Europe 2003, 8th Ada-Europe International Conference on Reliable Software Technologies*, Vol. 2655 of *Lecture Notes in Computer Science*, pp. 45–67, Springer.

Kihlstrom, Kim Potter, Moser, L. E., and Melliar-Smith, P. M. (1998), The SecureRing Protocols for Securing Group Communication, In *HICSS '98: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences*, p. 317, Washington, DC, USA: IEEE Computer Society.

Knight, John, Heimbigner, Dennis, Wolf, Alexander L., Carzaniga, Antonio, Hill, Jonathan, Devanbu, Premkumar, and Gertz, Michael (2001), The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications, (Technical Report CU-CS-926-01) University of Colorado – Department of Computer Science.

Kotla, Ramakrishna, Alvisi, Lorenzo, Dahlin, Mike, Clement, Allen, and Wong, Edmund (2007), Zyzzyva: speculative byzantine fault tolerance, In *SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pp. 45–58, New York, NY, USA: ACM.

Kuzmanovic, Aleksandar and Knightly, Edward W. (2003), Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants, In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 75–86, New York, NY, USA: ACM.

Lala, Jaynarayan H. and Schneider, Fred B. (2009), IT Monoculture Security Risks and Defenses, *IEEE Security & Privacy*, 7(1), 12–13.

Lala, J.H. (2003), Introduction, In *Foundations of Intrusion Tolerant Systems, 2003 [Organically Assured and Survivable Information Systems]*, pp. x–xix, IEEE Computer Society.

Lamport, L. (2001), Paxos Made Simple,, *ACM SIGACT News*, 32(4), 51–58.

Lamport, Leslie (1998), The part-time parliament, *ACM Trans. Comput. Syst.*, 16(2), 133–169.

Laprie, J-C. (1995), DEPENDABLE COMPUTING AND FAULT TOLERANCE : CONCEPTS AND TERMINOLOGY, In *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, pp. 2–11.

Leita, Corrado and Dacier, Marc (2008), SGNET: A Worldwide Deployable Framework to Support the Analysis of Malware Threat Models, In *Seventh European Dependable Computing Conference, EDCC-7*, pp. 99–109, IEEE Computer Society.

Li, Lixin and Just, James Edward (Jul. 12, 2007), Diversity-based Security System and Methods. U.S. Patent Application Publication #2008/0016314 A1 issued Jan.17, 2008.

Littlewood, Bev and Miller, Douglas R. (1989), Conceptual Modeling of Coincident Failures in Multiversion Software, *IEEE Trans. Software Eng.*, 15(12), 1596–1614.

Littlewood, Bev, Popov, Peter T., and Strigini, Lorenzo (2001), Modeling software design diversity, *ACM Comput. Surv.*, 33(2), 177–208.

Littlewood, Bev and Strigini, Lorenzo (2004), Redundancy and Diversity in Security, In Samarati, Pierangela, Ryan, Peter Y. A., Gollmann, Dieter, and Molva, Refik, (Eds.), *ESORICS 2004: Proceedings of 9th European Symposium on Research Computer Security*, Vol. 3193 of *Lecture Notes in Computer Science*, pp. 423–438, Springer.

Madan, Bharat B., Goseva-Popstojanova, Katerina, Vaidyanathan, Kalyanaraman, and Trivedi, Kishor S. (2004), A method for modeling and quantifying the security attributes of intrusion tolerant systems, *Perform. Eval.*, 56(1-4), 167–186.

Martin, Jean-Philippe and Alvisi, Lorenzo (2006), Fast Byzantine Consensus, *IEEE Trans. Dependable Secur. Comput.*, 3(3), 202–215.

Min, Byoung-Joon and Choi, Joong-Sup (2004), An approach to intrusion tolerance for mission-critical services using adaptability and diverse replication, *Future Generation Comp. Syst.*, 20(2), 303–313.

Min, Byoung-Joon, Kim, Sung Ki, and Choi, Joong-Sup (2004), Secure System Architecture Based on Dynamic Resource Reallocation, In Chae, Kijoon and Yung, Moti, (Eds.), *Information Security Applications, 4th International Workshop, (WISA 2003)*, Vol. 2908 of *Lecture Notes in Computer Science*, pp. 174–187, Springer.

Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., Budhia, R. K., and Lingley-Papadopoulos, C. A. (1996), Totem: a fault-tolerant multicast group communication system, *Commun. ACM*, 39(4), 54–63.

Moser, L. E., Melliar-Smith, P. M., and Narasimhan, P. (1999), A Fault Tolerance Framework for CORBA, In *FTCS '99: Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, p. 150, Washington, DC, USA: IEEE Computer Society.

Obelheiro, Rafael R., Bessani, Alysson N., Lung, Lau C., and Correia, Miguel (2006), How Practical are Intrusion-Tolerant Distributed Systems?, (Technical Report TR0615) Departamento de Informatica Faculdade de Ciencias da Universidade de Lisboa.

Oberheide, Jon, Cooke, Evan, and Jahanian, Farnam (2008), CloudAV: N-Version Antivirus in the Network Cloud, In *SS'08: Proceedings of the 17th USENIX Security Symposium*, pp. 91–106, Berkeley, CA, USA: USENIX Association.

Ortalo, Rodolphe, Deswarte, Yves, and Kaâniche, Mohamed (1999), Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security, *IEEE Trans. Software Eng.*, 25(5), 633–650.

Pal, Partha Pratim, Rubel, Paul, Atighetchi, Michael, Webber, Franklin, Sanders, William H., Seri, Mouna, Ramasamy, HariGovind V., Lyons, James, Courtney, Tod, Agbaria, Adnan, Cukier, Michel, Gossett, Jeanna M., and Keidar, Idit (2006), An architecture for adaptive intrusion-tolerant applications, *Softw., Pract. Exper.*, 36(11-12), 1331–1354.

PITAC (2005), Cyber Security: A Crisis of Prioritization, Technical Report President's Information Technology Advisory Committee, PITAC.

Reynolds, James C., Just, James E., Lawson, Ed, Clough, Larry A., Maglich, Ryan, and Levitt, Karl N. (2002), The Design and Implementation of an Intrusion Tolerant System, In *International Conference on Dependable Systems and Networks (DSN 2002)*, pp. 285–292, IEEE Computer Society.

Rodrigues, Rodrigo, Kouznetsov, Petr, and Bhattacharjee, Bobby (2007), Large-scale byzantine fault tolerance: safe but not always live, In *HotDep'07: Proceedings of the 3rd workshop on on Hot Topics in System Dependability*, p. 17, Berkeley, CA, USA: USENIX Association.

Saïdane, Ayda, Nicomette, Vincent, and Deswarte, Yves (2009), The Design of a Generic Intrusion-Tolerant Architecture for Web Servers, *IEEE Trans. Dependable Sec. Comput.*, 6(1), 45–58.

Salamat, Babak, Jackson, Todd, Gal, Andreas, and Franz, Michael (2009), Orchestra: intrusion detection using parallel execution and monitoring of program variants in user-space, In *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, pp. 33–46, New York, NY, USA: ACM.

Sames, David, Matt, Brian, Niebuhr, Brian, Tally, Gregg, Whitmore, Brent, and Bakken, David E. (2002), Developing a Heterogeneous Intrusion Tolerant CORBA System, In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pp. 239–248, Washington, DC, USA: IEEE Computer Society.

Sellers, P.H. (1974), On the Theory and Computation of Evolutionary Distances, *SIAM Journal on Applied Mathematics*, 26(4), 787–793.

Shacham, Hovav, Page, Matthew, Pfaff, Ben, Goh, Eu-Jin, Modadugu, Nagendra, and Boneh, Dan (2004), On the effectiveness of address-space randomization, In Atluri, Vijayalakshmi, Pfitzmann, Birgit, and McDaniel, Patrick Drew, (Eds.), *roceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004*, pp. 298–307, ACM.

Shen, Y.P., Tsai, W.-T., Bhattacharya, S., and Liu, T. (2000), Attack tolerant enhancement of intrusion detection systems, Vol. 1, pp. 425–429.

Siqueira, Lindonete and Abdelouahab, Zair (2006), A Fault Tolerance Mechanism for Network Intrusion Detection System based on Intelligent Agents (NIDIA), In *SEUS-WCCIA '06: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pp. 49–54, Washington, DC, USA: IEEE Computer Society.

Stroud, Robert J., Welch, Ian S., Warne, John P., and Ryan, Peter Y. A. (2004), A Qualitative Analysis of the Intrusion-Tolerance Capabilities of the MAFTIA Architecture, In *International Conference on Dependable Systems and Networks (DSN 2004)*, pp. 453–, IEEE Computer Society.

Sullivan, Kevin J., Knight, John C., Du, Xing, and Geist, Steve (1999), Information Survivability Control Systems, In *21st International Conference on Software Engineering (ICSE)*, pp. 184–192.

Symantec (2009), Symantec Global Internet Security Threat Report – Trends for 2008, (Technical Report Volume XIV) Symantec.

Taylor, Carol and Alves-Foss, Jim (2005), Diversity as a computer defense mechanism, In *NSPW '05: Proceedings of the 2005 workshop on New security paradigms*, pp. 11–14, New York, NY, USA: ACM.

Totel, Eric, Majorczyk, Frédéric, and Mé, Ludovic (2006), COTS Diversity Based Intrusion Detection and Application to Web Servers, In Valdes, Alfonso and Zamboni, Diego, (Eds.), *Recent Advances in Intrusion Detection, 8th International Symposium, RAID'05*, Vol. 3858 of *Lecture Notes in Computer Science*, pp. 43–62, Springer.

USAF (1999), Report on Building the Joint Battlespace Infosphere, Volume 1: Summary, (Technical Report SAB-TR-99-02) United States Air Force Scientific Advisory Board.

Valdes, Alfonso, Almgren, Magnus, Cheung, Steven, Deswarte, Yves, Dutertre, Bruno, Levy, Joshua, Saïdi, Hassen, Stavridou, Victoria, and Uribe, Tomás E. (2003), Dependable Intrusion Tolerance: Technology Demo, In *3rd DARPA Information Survivability Conference and Exposition (DISCEX-III 2003)*, pp. 128–130, IEEE Computer Society.

van Renesse, Robbert, Birman, Kenneth P., and Maffeis, Silvano (1996), Horus: a flexible group communication system, *Commun. ACM*, 39(4), 76–83.

Veríssimo, Paulo, Neves, Nuno Ferreira, Cachin, Christian, Poritz, Jonathan A., Powell, David, Deswarte, Yves, Stroud, Robert J., and Welch, Ian (2006), Intrusion-tolerant middleware: the road to automatic security, *IEEE Security & Privacy*, 4(4), 54–62.

Veríssimo, Paulo, Neves, Nuno Ferreira, and Correia, Miguel (2003), Intrusion-Tolerant Architectures: Concepts and Design, In de Lemos, Rogério, Gacek, Cristina, and Romanovsky, Alexander B., (Eds.), *Architecting Dependable System*, Vol. 2677 of *Lecture Notes in Computer Science*, pp. 3–36, Springer.

Wang, Dazhi, Madan, Bharat B., and Trivedi, Kishor S. (2003), Security analysis of SITAR intrusion tolerance system, In *SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*, pp. 23–32, ACM.

Wang, Feiyi, Jou, Frank, Gong, Fengmin, Sargor, Chandramouli, Goseva-Popstojanova, Katerina, and Trivedi, Kishor (2003), SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services, In *Foundations of Intrusion Tolerant Systems*, Los Alamitos, CA, USA: IEEE Computer Society.

Weatherwax, Eric, Knight, John, and Nguyen-Tuong, Anh (2009), A Model of Secretless Security in N-Variant Systems, In *Workshop on Compiler and Architectural Techniques for Application Reliability and Security (CATARS), In the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Network (DSN2009)*.

Whitehouse, Ollie (2007), An Analysis of Address Space Layout Randomization on Windows Vista, Technical Report Symantec.

Xu, Jun, Kalbarczyk, Zbigniew, and Iyer, Ravishankar K. (2003), Transparent Runtime Randomization for Security, In *22nd Symposium on Reliable Distributed Systems (SRDS 2003)*, IEEE Computer Society.

Yin, Jian, Martin, Jean-Philippe, Venkataramani, Arun, Alvisi, Lorenzo, and Dahlin, Mike (2003), Separating agreement from execution for byzantine fault tolerant services, In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 253–267, New York, NY, USA: ACM.

Yu, Dong and Frincke, Deborah A. (2004), Towards Survivable Intrusion Detection System, In *37th Annual Hawaii International Conference on System Sciences*.

List of Acronyms

ADFNIC Autonomic Distributed Firewall Network Interface Card

ASR Address Space Randomization

AVI Attack Vulnerability Intrusion

CORBA Common Object Request Broker Architecture

DIT Dependable Intrusion Tolerance

DPASA Designing Protection and Adaptation into a Survivability Architecture

DSR Data Space Randomization

FOREVER Fault/instrusiOn REmoVal through Evolution and Recovery

GCCS Global Command and Control Systems

HACQIT Hierarchical Adaptive Control for QoS Intrusion Tolerance

IDS Intrusion Detection System

ISR Instruction Set Randomization

ITDOS Intrusion Tolerant Distributed Object Systems

ITUA Intrusion Tolerance by Unpredictable Adaptation

JBI Joint Battlespace Infosphere

LAN Local Area Network

MAFTIA Malicious and Accidental Fault Tolerance for Internet Applications

MTTSF Mean Time To Security Failure

MVEE Multi Variant Execution Environment

ORB Object Request Broker

OOB Out-Of-Band

OTS Out-Of-The-Shelf

PSQ Publish-Subscribe-Query

SITAR Scalable Intrusion Tolerant Architecture

STM State Transition Model

TCP Transmission Control Protocol

TRR Transparent Runtime Randomization

VLAN Virtual Local Area Network

Annex A: Related Research Projects

The different aspects of the research presented in this report have been conducted in the following list of research projects:

Project	Home Page
Diversity with Off-The-Shelf components (DOTS)	http://www.csr.city.ac.uk/projects/dots.html
ReSIST	http://www.resist-noe.org/index.html
Honeynet Project	http://www.honeynet.org/
IBM Autonomic Computing	http://www.research.ibm.com/autonomic/index.html
MAFTIA	http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/
SNORT	http://www.snort.org/
HACQIT	http://seclab.cs.ucdavis.edu/projects/HACQITsum.html
DIT	http://www.sdl.sri.com/projects/dit/
Willow Survivability Architecture	http://dependability.cs.virginia.edu/research/willow/
ITUA	http://www.perform.csl.illinois.edu/itua.html
SITAR	http://people.ee.duke.edu/~kst/sitar.html

This page intentionally left blank.

Annex B: Researchers and Research Groups/Labs:

The following table include the main researchers, research labs and/or groups which conducted the research presented in this report.

Researchers	**Home Page**
Stephanie Forrest	http://www.cs.unm.edu/~forrest/
Partha pal	http://www.dist-systems.bbn.com/people/ppal/
Kishor Trivedi	http://people.ee.duke.edu/~kst/
Illir Ghashi	http://www.csr.city.ac.uk/staff/gashi/
Bev Littlewood	http://www.csr.city.ac.uk/staff/littlewood/
Babak Salamt	http://www.babaks.com/
Paulo Sousa	http://paulosousa.me/
Katerina Goseva-Popstojanova	http://www.csee.wvu.edu/~katerina/
John C. Knight	http://www.cs.virginia.edu/~jck/
Dave Bakken	http://www.eecs.wsu.edu/~bakken/
Ayda Saidane	http://www.dit.unitn.it/~saidane/

Research Group	**Home Page**
DEPENDABILITY Research Group	http://dependability.cs.virginia.edu
Center for Software Reliability	http://www.csr.city.ac.uk/index.html
Computer Science Laboratory - CSL SRI	http://www.sdl.sri.com/

This page intentionally left blank.

Annex C: Industries:

In this state of the art study, we have identified a few industrial companies which collaborated in conducting research which has been presented in this report. In the following table we present the list of this companies and the projects they were involved in:

Company	**Home Page**	**Project**
BBN Technologies	http://www.bbn.com/	ITUA and DPASA
CoSine Communications Inc.	Unavailable	SITAR
IntruVert Networks Inc	Unavaivable	SITAR
Network Associates Inc	Unavaivable	ITDOS
The Boeing Company	http://www.boeing.com/	ITUA
Teknowledge	http://www.teknowledge.com/	HACQIT

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

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 R&D Canada – Valcartier 2459 Pie-XI Blvd. North Val-Bélair, Quebec, Canada G3J 1X5	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED (NON-CONTROLLED GOODS) DMC A REVIEW: GCEC April 2011

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.)

Redundancy with Diversity Based Software Architectures for the Detection and Tolerance of Cyber-Attacks: State of the Art

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

Abdelouahed Gherbi, P.; Charpentier, R.; Couture, M.

5. DATE OF PUBLICATION (Month and year of publication of document.) February 2012	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 84	6b. NO. OF REFS (Total cited in document.) 94

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 R&D Canada – Valcartier
2459 Pie-XI Blvd. North Val-Bélair, Quebec, Canada G3J 1X5

9a. PROJECT NO. (The applicable research and development project number under which the document was written. Please specify whether project or grant.) 15BA02	9b. GRANT OR 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 2010-287	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.)

(X) Unlimited distribution
(   ) Defence departments and defence contractors; further distribution only as approved
(   ) Defence departments and Canadian defence contractors; further distribution only as approved
(   ) Government departments and agencies; further distribution only as approved
(   ) Defence departments; further distribution only as approved
(   ) Other (please specify):

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.)

Software systems security remains a critical issue. This is evidenced by the ever in-creasing number and sophistication of cyber-attacks. This situation is the result of the combination of several factors. The software-based functionality of these systems is in-creasingly complex. The systems are often connected through open networks such as the Internet, which is increasingly accessible to potentially malicious users. Finally, these systems run software which is substantially similar. This is called *IT monoculture*. The mitigation against this issue requires implementation of the principle of diversity. The principle of diversity aims to reduce the common vulnerability in software and, in turn, increase the difficulty of violating the security of the systems that use diversity. The objective of this document is to present the state of the art in terms of approaches which use diversity for security purposes. Three different approaches can be distinguished: automated diversity, diversity-based behavior monitoring and diversity-based intrusion tolerance.

La sécurité des systèmes informatique demeure une problématique ardue. Ceci est confirmé avec la croissance continue de cyber-attaques en nombre et en sophistication. Cette situation est le résultat de la combinaison de plusieurs facteurs. La fonctionnalité de ces systèmes basée sur les logiciels est de plus en plus complexe. Les systèmes sont souvent reliés par des réseaux ouverts comme Internet qui est de plus en plus accessibles aux utilisateurs potentiellement malveillants. Enfin, ces systèmes exécutent des logiciels qui sont substantiellement similaires. Ceci est communément appelé IT monoculture. La solution de ce problème est basée sur le principe de la diversité. Cette dernière vise à réduire les vulnérabilités communes dans un ensemble redondant et divers de logiciels. Ceci augmente la difficulté de violer la sécurité des systèmes qui utilisent la diversité. L'objectif de ce document est de présenter l'état de l'art en termes d'approches qui utilisent la diversité à des fins de sécurité. Trois approches différentes peuvent être distinguées : la diversité automatisée, la surveillance de comportement basée sur la diversité et la tolérance d'intrusion basée sur la diversité.

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.)

Cyber Attack, Security, Software Architecture, Redundancy, Diversity, Tolerance, Survivability, Dependability