Int. J. Appl. Math. Comput. Sci., 2022, Vol. 32, No. 4, 701–719
DOI: 10.34768/amcs-2022-0048

REDUNDANCY–BASED INTRUSION TOLERANCE APPROACHES MOVING FROM CLASSICAL FAULT TOLERANCE METHODS

FELICITA DI GIANDOMENICO [a], GIULIO MASETTI [a,*], SILVANO CHIARADONNA [a]

[a]Institute of Information Science and Technologies
National Research Council
Via G. Moruzzi 1, 56124 Pisa, Italy
e-mail: {felicita.digiandomenico,giulio.masetti,silvano.chiaradonna}@isti.cnr.it

Borrowing from well known fault tolerant approaches based on redundancy to mask the effect of faults, redundancy-based intrusion tolerance schemes are proposed in this paper, where redundancy of ICT components is exploited as a first defense line against a subset of compromised components within the redundant set, due to cyberattacks. Features to enhance defense and tolerance capabilities are first discussed, covering diversity-based redundancy, confusion techniques, protection mechanisms, locality policies and rejuvenation phases. Then, a set of intrusion tolerance variations of classical fault tolerant schemes (including N Version Programming and Recovery Block, as well as a few hybrid approaches) is proposed, by enriching each original scheme with one or more of the previously introduced defense mechanisms. As a practical support to the system designer in making an appropriate choice among the available solutions, for each developed scheme a schematic summary is provided, in terms of resources and defense facilities needed to tolerate f value failures and k omission failures, as well as observations regarding time requirements. To provide an example of more detailed analysis, useful to set up an appropriate intrusion tolerance configuration, a trade-off study between cost and additional redundancy employed for confusion purposes is also carried out.

Keywords: intrusion tolerance, cyberattack, diversity-based redundancy, protection mechanisms.

1. Introduction

Cybersecurity is increasingly considered a fundamental requirement to IT assets protection, with particular emphasis on assets that cover essential roles from both social and economic perspectives (generically indicated as critical infrastructures). Therefore, in parallel with research in more traditional fault tolerant systems (including the works of Mejdi *et al.* (2020), Puig *et al.* (2018) or Majdzik (2022), among recent publications on fault tolerant control), the research community has also proposed a plethora of studies and solutions regarding intrusion tolerance, ranging from attack prevention techniques to countermeasures applied in response to identified compromised components (e.g, Ylmaz and Gänen, 2018; Zhou *et al*., 2018; Alladi *et al*., 2020; Zhang *et al*., 2019; Khraisat *et al*., 2019).

Among them, intrusion tolerance was proposed already a couple of decades ago. Borrowing from the well known fault tolerance principles, intrusion tolerance aims to guarantee that a system works correctly even when some of its parts are compromised.

This paper focuses on redundancy-based intrusion tolerance, where redundancy of components is exploited as a first defense line against a subset of compromised components within the redundant set. In this perspective, the offered original contribution consists in revisiting classical redundancy-based fault tolerant approaches from the security perspective. Specifically, first a set of features is proposed to help improving the tolerance to the effect of intrusions, namely diversity-based redundancy, confusion techniques, protection mechanisms, locality policies and rejuvenation phases. Then, the exploitation of such features to originate intrusion tolerant alternatives from classical fault tolerance schemes is explored. Practical advice to guide system designers in the choice and configuration of the appropriate architecture is also developed.

In particular, focusing on a single system component

*Corresponding author

that requires tolerance to the effects of intrusions generated by attacks, five fault tolerant redundancy schemes are considered: the two well-known and widely adopted N Version Programming (Avizienis, 1985) and Recovery-Block (Randell, 1975), and, in addition, N Self-checking Programming (Laprie *et al*., 1990), Consensus Recovery Block (Randell and Xu, 1994) and Self-Configuring Optimistic Programming (Bondavalli *et al*., 1993). We believe this constitutes a basic, but versatile set of solutions, which covers the extremes of the spectrum and representative intermediate schemes, from which additional variants can be easily derived. Indeed, the reasoning conducted on the five addressed schemes are sufficiently general and complete to be applicable also in slight variants of such schemes.

The rest of the paper is organized as follows. Section 2 provides the background on intrusion tolerance, with specific focus on the role of *diversity*, and overviews related work. Section 3 discusses the directions in which standard fault tolerance architectures can be extended to address intrusions and presents the attack model. Then, Section 4 focuses on N Version Programming like architectures, Section 5 on those similar to Recovery Blocks and Section 6 on a few hybrid architectures. Section 7 provides a practical summary of the proposed redundancy-based intrusion tolerance schemes, as well as an example of detailed analysis, focused on confusion. In Section 8, considerations on intrusion tolerance solutions with respect to other ICT system components that are typically the target of cyberattacks are first presented, followed by a brief discussion on future research lines.

2. Context and related work

To better position the offered research contribution, background on Intrusion Tolerance and related work are presented in this section.

2.1. Background on intrusion tolerance. Similar to fault tolerance, intrusion tolerance (IT) is comprehensive of several techniques aiming at counteracting an error from turning into a failure. As for accidental faults, intentional attacks generate erroneous behavior in the attacked component, which can degenerate into failure if not properly managed. Here, the specific emphasis on *intrusion* instead of the generic *fault* refers to the malicious characterization of an intruder launching an attack, exploiting at best the gained knowledge to compromise the system as much as possible, instead of random combinations of adverse events accidentally originated. The attack model in Section 3.4 discusses in detail how an attacker can compromise a component. Therefore, the developed intrusion tolerance approaches are equipped with specific measures to contrast a skilled attacker bent on compromising the system, as discussed in the following.

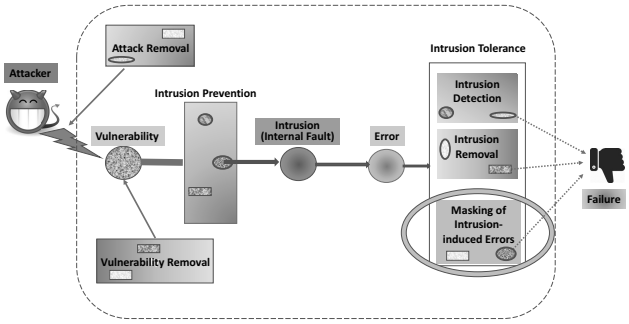

Fig. 1. Representation of the attack-vulnerability-intrusion-error-failure chain (inspired by Veríssimo *et al*. (2003)) and technique categories to cope with it.

In this paper, we refer the more sophisticated chain illustrated in Fig. 1 rather than the classical fault-error-failure of Avizienis *et al*. (2004). As illustrated in the figure, an attacker exploits a system/component vulnerability to launch an attack that, if successful, leads to an intrusion, seen as an internal fault. This fault may generate one or more errors that, if not properly managed, can lead to the failure of the service provided by the system/component. This is in line with the attack-vulnerability-failure fault model by Veríssimo *et al*. (2003).

To avoid/mitigate the potential failures, intrusion prevention and tolerance are put in place to cope with the attacks and their consequences. Prevention is the first defense against intrusions, but since prevention cannot be assured to be totally effective, intrusion tolerance is also needed. In Fig. 1, small "holes" within the area illustrating a defense technique represent weaknesses of the technique itself in accomplishing its task. Attack removal and vulnerability removal are radical measures to cope with the source of the intrusion, so avoiding that similar attacks can be perpetrated again.

IT techniques primarily include intrusion detection (Scarfone and Mell, 2010; Khraisat *et al*., 2019), intrusion removal (Sousa *et al*., 2008) and masking of intrusion induced errors (Veríssimo *et al*., 2003).

In this paper, the focus is on the masking category of IT means, seen as a first line of defense while the other techniques collect enough knowledge to deal more radically with the presence of the intrusion. Moreover, here the emphasis is on a single system component that is selected for higher protection through redundancy-based IT. In the redundant structure that replaces the single system component, redundant components do not interact with each other, but only provide their output to the unit in charge of collecting them and select the final

result. Hence, tolerance paradigms applicable in contexts where multiple interacting components are in place, such as solutions resilient to byzantine faults in distributed systems (see the work of Distler (2022) for a survey) or secure multi-party computation (Archer *et al*., 2018) are out of scope.

When the redundancy is obtained employing just replicas, i.e., identical copies, if the attacker is successful in compromising one replica, he/she has also the knowledge to successfully intrude into all the other replicas, since they share the same vulnerabilities. Therefore, to contrast intrusion propagation through common vulnerabilities, diversity is advocated as a basic and powerful instrument. In practice, instead of replicas, the redundant structure adopts variants, which are functionally equivalent components developed with some form of diversity.

Common mode vulnerabilities in redundant software structures can have different origins (see Fig. 2):

- functional influences, which have primarily to do with the same design vulnerabilities present in the redundant components, or the adoption of a common execution environment exposing the same vulnerabilities; a variety of means to cope with functional influence have been proposed in the literature (e.g., Littlewood and Strigini, 2000);

- locational influences, which expose all the redundant components located in the same physical or logical partition of a system to be isolated by an attacker (e.g., through intrusion in the communication network), without the possibility to receive inputs and to provide outputs to the adjudicator unit; measures to cope with locational influence mainly consist in adopting redundant (diverse) communication channels, and/or distribution of the redundant components in different physical/logical sites;

- administrative influences, potentially leading to massive intrusions by exploiting social engineering, when the same security policies apply to all the redundant components; measures to cope with such problem include the adoption of different security management policies for the different redundant components, and possibly different administrative domains; then, some form of coordination among such different policies/administration domains turns out to be needed.

Given its central role, all the proposed intrusion tolerance mechanisms described in the following embody some form of diversity. However, exploration of how to induce diversity is out of the scope of this paper. Indeed, addressing the diversity techniques, briefly outlined in

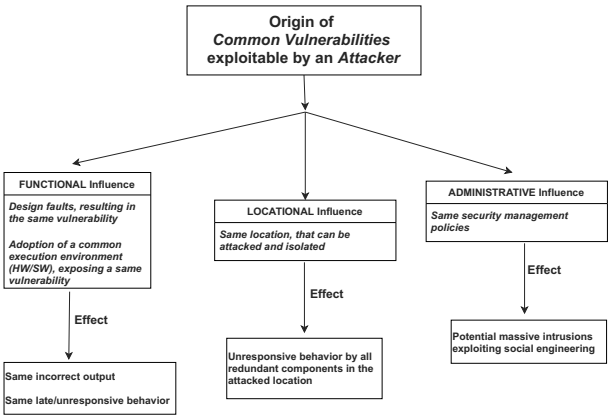

Fig. 2. Types of correlation and their effects on the affected system/component.

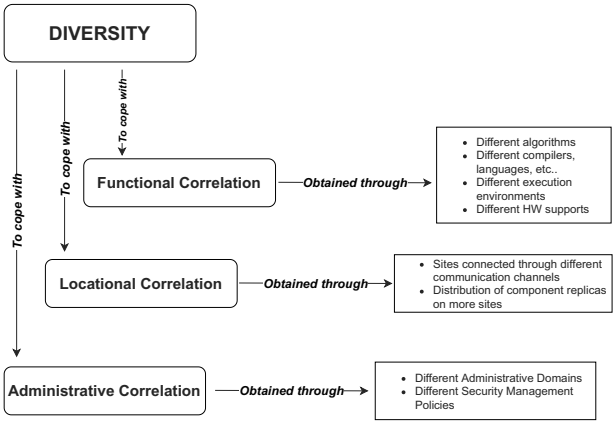

Fig. 3. Diversity approaches to cope with correlation types.

Fig. 3, at a level of detail that allows us to quantify their effectiveness on mitigating a common-mode failure probability is challenging and requires significant efforts and individual studies in focused contexts. For example, Garcia *et al.* (2014; 2020) address this kind of study, but concentrated on attacks to operating systems, and propose architectural solutions to cope with residual common mode vulnerabilities resulting from the adoption of different operating systems. Therefore, in the following it is assumed that variants are already available and they are satisfactory for the purpose of their usage in the referred intrusion tolerance scheme. Accounting for degrees of diversity is of course needed when dependability analysis is targeted, a subject for future studies.

2.2. Related work. Initial intrusion tolerance approaches are dated in the early 1990s, although they appear as isolated works, mainly focusing on protocols (see the work of Veríssimo *et al*. (2003) for an overview of initial research on this subject). Many other papers tackled this topic in more recent years. With no intention

of being exhaustive, the list mainly includes works that addressed intrusion tolerance from specific perspectives. In fact, some studies focused on analysis and evaluation of security oriented strategies to manage system resources in order to reach desired reliability/availability levels (Wang *et al*., 2014; Haphuriwat and Bier, 2011; Tarraf *et al*., 2017). Others, instead, concentrated on architectural solutions for specific application domains, smart grids (Babay *et al*., 2018), autonomous driving (Vöelp and Verissimo, 2018), or web servers (Saidane *et al*., 2009). Specific security architectures tailored to problems generated by specific vulnerabilities are the subject of the works of Gorbenko *et al*. (2020) and Garcia *et al*. (2014), while investigations on trade-offs between competing requirements are conducted by Gashi *et al*. (2016).

Differently from the above referred papers, here a set of logical intrusion tolerant architectures, exploitable in a variety of application contexts, is developed. With a similar objective, in the work of Rodriguez *et al*. (2015) preliminary redundant architectures for intrusion tolerance have been proposed, obtained by adopting minimal extensions to classical architectures, according to a set of defined failure classes for secure services. In particular, N-Version Programming, standby sparing with error detection (SED) and Self-Checking Programming are considered and extended. However, this paper targets a more ambitious aim, by conducting a more general and structured discussion on the definition of intrusion tolerance schemes and specifically proposing a variety of means to incrementally enhance intrusion tolerance abilities.

Finally, Di Giandomenico and Masetti (2021) present the various aspects around redundancy-based intrusion tolerance, concerning both system and redundancy characteristics. In particular, four dimensions have been identified and discussed as major drivers to the design of intrusion tolerant solutions, tailored to the specific system at hand. They are: (i) the attack model, characterizing the cyberattacks considered; (ii) the categories of ICT system components targeted by attacks; (iii) the system model and failure assumptions, namely whether the system at hand is a single system component, or a structured, distributed system; (iv) the characterization of the type of redundancy that can be put in place. This framework paves the way for the development of the articulated approaches carried out in the following as an original new contribution.

3. Revisiting fault-tolerance architectures to address intrusion tolerance

In this section, the design and enrichment aspects around intrusion tolerance are addressed, moving from fault tolerance.

The attack model underlying the developed redundancy-based intrusion tolerance schemes is also included.

3.1. Design issues. Slightly extending the taxonomy from the works of Nascimento *et al*. (2013) and Pullum (2001) presented in the fault tolerance context, a diversity-based intrusion tolerance scheme to mask the presence of intrusion induced errors is characterized by four major design issues: (i) the decision on the measures to adopt for enforcing diversity in the redundant structure under development; (ii) selection of the variants to employ in the redundant configuration (e.g., how many variants to employ, each developed according to which diverse methodology, thus showing a required reliability level); (iii) the decision on the execution pattern of the selected versions; (iv) the decision on the adjudicator function to adopt for selecting the only output from the set provided by the employed variants. The designer mainly addresses these issues based on specific needs of the application under development (including dependability requirements, as well as other requirements in the time domain and operational context), available development environment facilities, and reference fault tolerance architectures.

The distinctive role played by diversity in redundancy-based IT schemes and related means to promote it have been already addressed in Section 2.1.

Regarding the execution model for the variants, both *sequential* and *parallel* execution are implemented in the reference schemes. When adopting sequential execution, as typically done in the Recovery Block approach, a reliable checkpointing mechanism is needed, to save the state of the system before any variant starts executing and from which an alternate variant starts its execution, should the previous variant fail. Parallel execution implies concurrent execution of the variants, thus requiring adequate computer resources and the use of mechanisms to assure that the same input is provided to all the variants.

The *adjudicator* component plays a very critical role in the overall redundant organization, being the entity that takes the final decision on the outcome of the redundant computation. A variety of adjudication functions have been proposed in the literature. They belong to two major categories: *Voters* and *Acceptance Tests*. Voters make a judgement on the set of variants results and are typically employed when the variants follow the parallel execution pattern. Several kinds of voters have been proposed, such as majority voter, consensus voter, median voters (Pullum, 2001; Nascimento *et al*., 2013). Note that the presence of diversity requires more sophisticated voting solutions than simple bitwise comparison (correct variant results are expected to be not perfectly coincident). Acceptance tests make an absolute judgement on each single variant result. This kind of adjudicator is typically used when the execution of variants is sequential. Popular

acceptance tests are based on satisfaction of requirements or reasonableness test (see the work of Pullum (2001) for more examples). Similarly to the voter, resorting to diverse techniques to generate variants may in general require a specific acceptance test for each variant. Hybrid adjudicator forms that employ combinations of voter and acceptance test have been also explored (Randell and Xu, 1994).

To help addressing omission failures experienced by variants, the adjudicator component is typically equipped with a *timeout* mechanism that terminates the waiting on a variant's result, after a predefined time interval determined on the basis of maximum execution time for the variant under consideration. In the following, it is assumed that each addressed intrusion tolerance scheme adopts a timeout for this purpose.

3.2. Enhancing intrusion tolerance abilities. To further enhance efficacy of the offered IT solutions, a few other features that have been proposed to support fault tolerance and/or security properties in general are exploited. Among them, the concepts of *rejuvenation*, *locality*, *access control* and *confusion* have been selected as promising candidates. A brief recall of each of them, with considerations on their employment in the proposed IT schemes, is in the following.

As discussed by Di Giandomenico and Masetti (2021), attack consequences are classified into three categories: *Functionality Change*, *Performance Degradation* and *Information Leakage*, and the security attributes that are primarily impacted, are *integrity*, *availability* and *confidentiality*, respectively. In the work of Rodriguez *et al*. (2015), all the three attributes are considered; in particular, confidentiality is explicitly addressed by deploying an Information Leak Detector close to each variant. Although this same solution would be applicable also for the configurations developed in this work, it is not adopted since it targets detection instead of tolerance. Therefore, confidentiality is not directly addressed here, since the interest is in tolerance capabilities, intended as the ability to continue providing correct operation exploiting redundancy, without further intervention. Specific mechanisms devoted to confidentiality protection are expected to be easily added to the proposed architectural approaches. However, it needs to be recognized that the presence of redundant components exposes confidentiality to higher risk because the attack surface increases, thus requiring in general more sophisticated and costly solutions than just addressing confidentiality as a primary objective, without caring about availability and integrity. Although unpleasant, this is not surprising as this kind of situation typically occurs when contrasting properties need to be accounted for, for which optimization of one degrades the performance of the others.

3.2.1. Locality. Location diversity, consisting in placing several physical components of a system at different sites, is recognized since long time as a good practice to cope with physical threats, like natural disasters (e.g., floods or fires) or basic service outages (e.g., electrical outage). When deliberate attacks are considered, as in intrusion tolerance, this measure becomes even more relevant. Interestingly, location diversity can be easily joined with diverse administration domains characterizing the different sites, so further improving the defense against attackers (Obelheiro *et al*., 2006). In the following it is assumed that the defender has s sites at disposal and can distribute the variants among the sites. Utility functions (e.g., input scatter, output gather, adjudicator, etc), deployed on a special site, are not counted.

Notice, though, that scattering data and code among on-premises and/or commercial data centers may degrade confidentiality, depending on the accessibility conditions to the chosen diverse sites (see the works of Gashi *et al*. (2016) or Khan and Babay (2021) for specific examples). Thus, side effects of location diversity have to be carefully analyzed and managed. However, since such analysis is application specific, it is not addressed here.

3.2.2. Rejuvenation. For long-living systems, rejuvenation (Dohi *et al*., 2020) enhances fault tolerance: once in a while, each replica is subject to some form of clearance/rejuvenation in order to reduce its failure rate or the frequency of intermittent faults. While relevant to contrast any kind of malfunctions producing erroneous behaviours of a system/component that tend to increase along time, the benefit of rejuvenation is essential in the context of intrusion tolerance. In fact, if clearance actions are effective enough, rejuvenation reduces the time window for an attack to be successful to be just the time between two consecutive clearances. To this purpose, rejuvenation should: (i) take place with high frequency, (ii) be coupled with diversity, i.e., each rejuvenation introduces as much changes as possible for reducing the correlations between pre- and postclearance.

Rejuvenation can be either scheduled at predefined time intervals, or activated when some critical event is perceived (e.g., the variant itself could apply internal checks to detect suspicious behavior). It is in general a costly operation since, to be effective, the rejuvenated variant should be significantly diverse from the original one. Acting on the functional level is expected to assure a higher degree of diversity, although simpler *automatic diversity* (like a change in the name or the position of files in the file system) or *obfuscation* techniques (e.g., adopt sophisticated compilation features that make reverse engineering difficult) can be considered as well.

Considering the general case of an interruptible system, variants under rejuvenation have to be added on

top of those needed by the scheme to assure the required tolerance level. In fact, the rejuvenation procedure requires an interval of time to be completed; therefore, when under rejuvenation, a variant skips one or more executions performed by the redundancy scheme it is involved in, until the rejuvenation phase completes.

Of course, since rejuvenation does not guarantee full independence between pre- and postversions of the variant from the attacker perspective, it cannot be the only defense mechanism in place.

3.2.3. Access control policies. In computer security, access control has been widely investigated (e.g., by Qiu *et al*. (2020) with reference to Internet of Things (IoT) technology). Through authentication and authorization, access control policies make sure users are who they say they are and that they have appropriate access to the intended resource.

The redundancy-based solutions for Intrusion Tolerance that will be detailed in the following include components of different criticality: the functionally equivalent variants show lower criticality than adjudicator components responsible for selecting the outcome from the variants outputs. It derives that the adjudicator component needs a higher protection level than individual variants, in terms of reducing the ability to an intruder to access it as a resource to compromise. So, the need of adequate protection mechanisms and access control techniques is even more exacerbated in the intrusion tolerance context, to avoid defeating the effort of costly redundancy.

For embedded or IoT systems, it is common to exploit a Root-Of-Trust to enhance security, and also IT architectures can be complemented with such a mechanism (Rodriguez *et al*., 2015). Other solutions, such as resorting to a distributed adjudicator component to avoid the single point of intrusion are possible.

In this paper, two layers of protection will be considered, indicated as L_0, L_1, where L_0 is the most stringent one, as also done in other studies (e.g., Rodriguez *et al*., 2015; Gashi *et al*., 2016; Hardekopf *et al*., 2001). Of course, a higher number of protection levels could be needed, to properly address requirements of specific contexts/application domains.

Deciding which part of the intrusion tolerant architecture has to be assigned a given layer is in general the result of several considerations, and there is no solution that is always preferable to the others. For instance, having most of the architecture in L_0 and only the communication channels in L_1 can appear on one hand too expensive, and on the other hand insecure because for an attacker it is easier to address the communications than the logic because this way almost no domain specific knowledge is required. However, this may not be the case because in some contexts (e.g., cyber-physical systems) both variants' and adjudicators' logic can be simple enough to be implemented in microcontrollers that are relatively cheap and easy to protect, or heavy and complex but implemented in containers that run on machines physically located in secure places, and communication channels can in turn be made intrusion-tolerant to reduce exposition to attacks.

3.2.4. Confusion. An accidental fault just happens. Conversely, an intentional fault (an attack) is the result of rational choices made by one or more adversaries, and usually strikes the variants that the attackers suppose to be the weakest ones. Thus, in intrusion tolerant systems it is common to find *confusion* strategies aimed at decreasing the confidence the attackers have in their decisions or increase the attack cost. Available strategies have been developed for different mitigation purposes and so show different degrees of effectiveness. For instance, replacing some variants with *camouflage* ones, i.e., components that perform no operations but mimic the interactions that operating variants have with the environment (Wang *et al*., 2014), can add a sufficient level of confusion only if the attackers have limited resources, in particular limited time. For a cyber-physical infrastructure, such as a Smart Grid, where the attackers can study the system and plan the intrusions for years, and where it is expected that foreign adversaries are willing to invest huge resources in the attack, camouflages are less effective. In the referred context, camouflages are of great help to set up honey pots aimed at gaining information about the attacks or to do detection, but to tolerate intrusions the most effective choice is to use extra but working variants and configure the tolerance scheme such that the adjudicator component decides (probabilistically or deterministically) which results to consider among the set of received ones.

To promote higher efficacy (although at higher cost), in the proposed IT schemes, confusion is applied adopting fully operative variants, similarly to solutions by Babay *et al*. (2018) or Khan and Babay (2021).

3.3. Organized summary. The above presented mechanisms to enhance intrusion protection and tolerance are exploited in the several redundancy-based intrusion tolerance schemes described in Sections 4–6, all following the general scheme depicted in Fig. 4. In the figure and in the rest of the paper, h indicates the minimum number of variants needed to tolerate the failures as indicated by the attack model (see Section 3.4), in accordance with the adopted redundancy-based scheme; s indicates the number of sites where the components of the adopted redundancy-based scheme are deployed; c indicates the number of additional variants employed for confusion purposes; r indicates the the maximum number of variants per site that can be under rejuvenation at each instant of

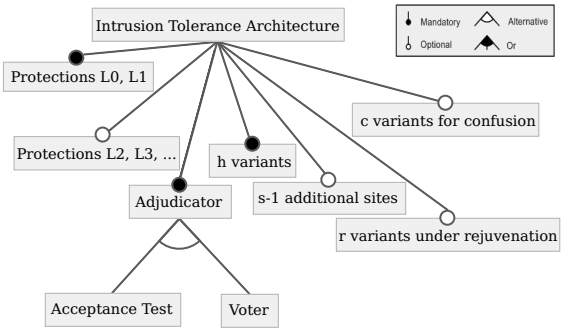

Fig. 4. Classification of involved mechanisms in the discussed intrusion tolerant architectures as mandatory, optional or alternatives.

time; L_i indicates the i-th protection layer applied to the components of the adopted redundancy-based scheme (in this paper, only two protection layers, L_0 and L_1, are assumed, but more than two can be in general employed). Then, Fig. 4 graphically summarizes which mechanisms are mandatory and which ones are optional when configuring an intrusion tolerant architecture among those proposed in this paper. In particular, one adjudicator, h variants, protection layers L_0 and L_1 are mandatory; $s-1$ additional sites, r variants under rejuvenation, c variants for confusion and additional protection layers are optional.

It is clarified that the proposed schemes are not meant to be an exhaustive intrusion tolerance set; rather, they are examples of how the features discussed in this section can be exploited to adapt the traditional fault tolerance organization to cope with intentional attacks. So, there is room for other interesting alternatives.

3.4. Attack model. The assumptions on the attack model are detailed in the following. The first statement is that only cyberattacks are considered. Therefore, an ICT component, even when composed of a physical device and software managing/controlling its operational life, can be compromised only through the software part. Adopting strategies that exploit the execution context of a software component, an attacker has the ability to perform the following:

- Intrude the variants only when they are running and
  - alter their result (value failure). The best strategy for the attacker is to try to compromise as much variants as possible, making them deliver the same (wrong) result, thus inducing a common-mode failure. f indicates the number of value failures (possibly of kind common-mode) generated during an execution of an intrusion tolerant scheme;
  - make their result unavailable, that is the compromised variants experience an omission failure. k indicates number of omission failures generated during an execution of an intrusion tolerant scheme.
- Isolate a site among the s where the variants are deployed. The effect is that the results of all the variants located on that site became unavailable, and the adjudication function perceives an omission failure from these variants. The assumption of no more than site under potential isolation by an attacker is in line with the works of Babay *et al.* (2018) or Khan and Babay (2021), and is made here for the sake of simplifying the presentation, but can be relaxed without invalidating the following developments.

The intrusion tolerant architectures considered can tolerate f arbitrary value failures (common-mode value failures, in the worst case) and k omission failures (comprehensive of both those intentionally caused by the attacker and those due to accidental causes).

The protection layer L_0 is assumed to be unattackable, so those functionalities put under this protection layer do not experience successful cyberattacks. The adjudication functions and possibly some of the variants are assigned the protection layer L_0, in addition to the input and output mechanisms. However, variants subject to higher protection from cyberattacks can be still affected by accidental faults. Instead, for what concerns the adjudication functions, given their higher simplicity and expected high reliability, no measures are included in the proposed schemes to tolerate their potential failures (both in selecting a wrong result and in not recognizing the exiting of a correct result). Of course, their reliability needs to be taken into account when assessing dependability properties of the scheme employing such adjudication components.

4. Family of NVP-like architectural proposals

This section is dedicated to the family of redundancy architectures that follow the N-Version Programming (NVP) organization. After a brief description of the classical NVP fault tolerant architecture (Avizienis, 1985), a few solutions obtained from its adaptation in the context of intrusion tolerance are discussed.

4.1. Reference N Version Programming. The N Version Programming (NVP) comprises an adjudicator that collects the results provided by the n variants (within a maximum specified time interval) and checks if there is a result that satisfies the adjudication. If

such a result is found, it is the output of the redundant structure. Otherwise, depending on the failure model, the component can switch to a benign failure state (e.g., in the context of Safety) or send in output a default value or choose one of the results exploiting other kinds of information, such as past knowledge about recurring errors (e.g., in the context of Reliability).

In the original formulation by Avizienis (1985), the adjudicator is a *simple voter*, but other kinds of adjudicator, such as variants of the simple majority (as presented by Pullum (2001)), or sophisticated mechanisms exploiting more complex *syndromes*, such as additional information of the reliability of the variants (like for the optimal adjudicator in the work of Di Giandomenico and Strigini (1990)) have been adopted. The variants are usually executed in parallel, although sequential execution has been investigated in contexts where computational resources are limited.

Many specific NVP-like redundant schemes have been proposed in the literature, as those reported by Nascimento *et al*. (2013), which focuses on fault tolerant Service-Oriented Architectures. Efficient organizations have been also pursued, for instance start performing result comparisons as soon as they arrive so to wait only m to agree, thus improving on time performance, relevant in particular in real-time systems. Of course, this scheme is fully recursive, meaning that a variant can in turn be implemented following the NVP schema.

4.2. Intrusion NVP counterparts. Starting from the classic NVP fault tolerant organization just recalled, a few adaptations to the fault intrusion context are proposed in the following, exploiting the defense features discussed in Section 3. Specifically, the usage of protection mechanisms at level of communication channels (mainly to detect side-channel information leak) and encapsulating the adjudicator component within a secure module (root-of-trust), are the minimum additions to build a basic intrusion tolerant NVP-like scheme, here called iNVP, as proposed by Rodriguez *et al*. (2015).

Adding extra variants to the h variants strictly needed introduces a form of confusion to weaken the attacker's strategy. Then, depending on how the adjudicator selects the h results to consider, out of the n provided by the variants, slightly different solutions are originated. Specifically, the adjudicator may choose the h results either uniformly at random (originating the the iNVP-R scheme), or deterministically (originating the iNVP-D scheme). In iNVP-R, the variants have no feedback about whether their result has been selected or not. Notice that there are $\binom{n}{h}$ possible ways to select h results among the n available. The idea at the heart of iNVP-R is to make impossible for the attacker to know if the resources invested in compromising a specific variant are wasted and then, to maintain the same probability

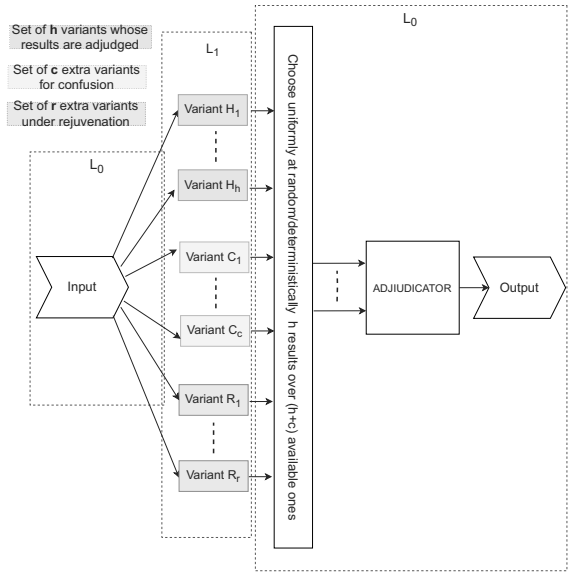

Fig. 5. Logical structure of the intrusion tolerant NVP-based schemes (iNVP, iNVP-R and iNVP-D). For graphical convenience, the variants within the same set (indicated with different shades of grey) are depicted sequentially, but in real deployments their order is mixed. As executions progress, each variant can move from one set to another.

of overall attack success, the attacker has to invest—in the mean—more resources. The iNVP-D alternative, where the choice of those variants not considered by the adjudicator is made deterministically, is competitive when the rejuvenation mechanism is also put in place. In fact, in such an organization, the potential higher knowledge from the attacker perspective (identity of those variants not considered by the adjudicator) is counteracted by the defender's power to accomplish rejuvenation of variants, so that those selected by the adjudicator include the more recently rejuvenated ones.

Of course, rejuvenation can be profitably applied to variants also in the previous iNVP-R strategy. However, since the choice of the variant results to adjudge is random, rejuvenation cannot be fully controlled to bring the highest benefit, as in iNVP-D.

The architecture depicted in Fig. 5 is representative of the NVP-based intrusion tolerance schemes discussed above. It includes protection layers (L_0 for the adjudicator, and L_1 for the variants and communication channels), the set of h variants whose results are considered by the adjudicator (randomly by the voter in iNVP-R, or pre-selected in iNVP-D), the set of c extra variants to generate confusion, and the set of extra r variants under rejuvenation at each execution.The basic iNVP is simply obtained by removing the variants impacted by rejuvenation and confusion.

Of course, the enhancements against attackers

brought by implementing the principles of confusion and rejuvenation are payed by the additional c plus r needed variants, respectively. The choice of suitable values for c and r need to be supported by a quantitative analysis devoted to assess a good tradeoff among contrasting aspects (dependability assurance and implied cost, depending on the criticality of the application).

A final consideration is about the $adjudicator$ component. Depending on its complexity, resorting to a full L_0 protection level (as in Fig. 5) could be not possible/convenient. Therefore, an alternative is to distribute its logic in such a way that only a small kernel of the adjudication algorithm needs to be protected in L_0 and the rest can stay in L_1. For the simple voter, in the work of Hardekopf *et al*. (2001) a distributed algorithm is presented, where only an interface is in L_0 and the rest of the logic is distributed among the variants, that are in L_1.

The interface is designed to be sufficiently small and simple to be, on one hand, formally verifiable for correctness and, on the other hand, be easily hosted within a Root-of-Trust.

However, despite the availability of advanced solutions and practice provided by the distributed systems community over decades, resorting to a distributed but less protected adjudication logic needs to be cautiously considered, since the higher exposure to the risk of attack could be not affordable for the application under development (e.g., is not recommended for a safety critical component).

5. Family of RB-like architectural proposals

This section is dedicated to the family of redundancy architectures that follow the Recovery Blocks (RB) organization. After a brief description of the classical RB fault tolerant architecture (Randell and Xu, 1994), a solution obtained from its adaptation in the context of intrusion tolerance is discussed.

5.1. Reference recovery block with n variants. The original formulation of the Recovery Block (RB) scheme (Randell and Xu, 1994) consists of n variants (also called alternates) that are executed sequentially, according to a predefined order, and the adjudicator that takes the form of an *acceptance test* (AT), applied to each individual result provided by the variants. On entry to a recovery block, the state is saved to permit backward error recovery (i.e., to establish a checkpoint). The execution starts with the first variant and then its AT checks the produced result. If the check is successful, the RB terminates its execution by releasing this (assumed to be) correct outcome and the taken checkpoint is deleted. Otherwise, the next variant is executed after restoring the state to the taken checkpoint, repeating the AT on the obtained result, and so on, until a successful check is encountered (RB terminates with a judged-to-be-correct outcome) or all the variants have completed their computation (RB terminates with a default outcome, or just a notification that no correct outcome was found). Of course this scheme is fully recursive, meaning that a variant can in turn be implemented following the recovery block structure.

As for the NVP adjudicator, here the acceptance test is a crucial component. On one hand, the acceptance test must be simple enough to assure higher correctness than the variant it checks, but on the other hand not so trivial to ignore variants' specificities and guarantee significance of the performed check. Coverage of an acceptance test, such that reliance can be put on the result of its check, depends on the application domain it is called to operate. Therefore, resorting to an RB structure strongly depends on the availability of acceptance tests characterized by enough coverage. Notice that employing diverse ATs for the variants, as typically required to cope for variants diversity, promotes intrusion tolerance of ATs themselves.

With respect to NVP, RB saves in computing resources, since in most cases only the primary is executed, while NVP exercises all the variants. However, this advantage poses also an additional implementation problem: how to synchronize the internal states of alternates that performed executions with those that did not. In fact, while the sequential execution paradigm of the RB variants (possibly involving a subset of the variants only) is fully adequate in the case of stateless components, a problem arises when the variants exploit their internal state in the computations they perform along time. In this latter case, synchronization at the state level is needed, to assure consistency of the computations. A Parallel Recovery Block, where the primary and all the alternates are executed although only a subset of them would be strictly needed to assure termination of the RB execution, is a simple although costly solution to cope with consistency of alternates internal state.

5.2. Intrusion Recovery-Block. The RB for intrusion tolerance purposes (iRB) requires high protection of crucial elements, so a basic solution consists in enclosing the checkpoint update/restore mechanism, the acceptance test and the switch that selects in turn the alternates under the protection level L_0, and the variants in L_1, as depicted in Fig. 6 (similarly to what has been proposed by Rodriguez *et al*. (2015)).

Many alternative organizations can be built, such as placing also the first alternate in L_0. This is more expensive, but guarantees that the most crucial variant from the attack perspective is adequately protected.

Being the acceptance test not perfect in detecting erroneous results (especially, intentionally counterfeited ones), confusion is relevant also for iRB. To contrast the potential ability of an attacker to monitor the communications between the variants and the switch

to identify the alternate whose result is sent to the acceptance test in the current phase (reading the content is unnecessary, only knowing the sender and the receiver is enough), the parallel execution of a subset of the alternates, or even executing additional variants, appears another suitable solution. In fact, it increases the attacker's confusion and also saves in overall execution time in case the variant fails the acceptance test, but requires more execution resources than the pure sequential execution. Notice, though, that executing other variants in parallel to the one whose result is considered in a given phase exposes them to attacks too. Therefore, to avoid depleting the tolerance ability of the scheme configuration too quickly, it is advisable to employ additional variants.

Mechanisms similar to those already discussed for the NVP scheme to enhance protection against attacks are applicable also to the RB scheme. They primarily include: (i) the random selection of the alternate(s) to execute at each phase (instead of a pre-defined deterministic choice); (ii) rejuvenation phases to enhance the *health* of the alternates, so as to nullify the effort previously made by the attacker to compromise the rejuvenated variant, coupled with having the most recently rejuvenated alternates to act as the first alternates to execute at the next scheme execution.

As for the previous family of NVP-like techniques, a variety of other intrusion tolerant RB-like schemes can be built, in addition to the examples just shown, by exploiting redundancy-by-diversity and the features presented in Section 3.

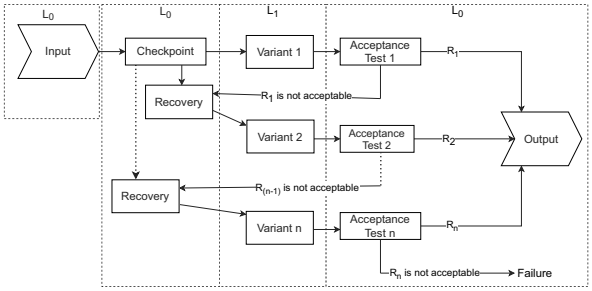

Fig. 6. Basic iRB configuration, employing n variants and 2 protection layers.

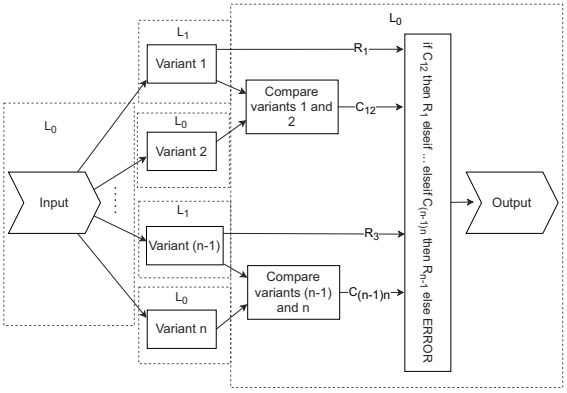

Fig. 7. Logical organization of the iSCP scheme with 2 protection layers and n variants grouped in $n/2$ self-checking components.

6. Family of hybrid architectural proposals

NVP and RB are recognized as the two extremes of redundancy-based fault tolerance techniques: exploitation of maximum execution resources to achieve minimum execution time (NVP) and minimum execution resources to be potentially payed by maximum execution time (RB). In between, hybrid solutions that try to combine the best aspects of each of the two have been proposed in the literature. Three of them have been selected, briefly recalled in the following (SCP (Laprie *et al.*, 1990), CRB (Randell and Xu, 1994) and SCOP (Bondavalli *et al.*, 1993)), and for each of them an alternative suitable to address intrusion tolerance is presented.

6.1. N Self-Checking Programming. The N Self-Checking Programming (SCP) architecture consists in the parallel execution of n_{SC} self-checking components, ordered according to some criteria (typically, based on performance and accuracy considerations). The outcome of the NSCP structure is the result provided by the first self-checking component, starting from the first one in the ordered list. A self-checking program results from the addition of redundancy into a program to check its own dynamic behavior during execution (Lyu, 1995). As reported in (Laprie *et al.*, 1990), a self-checking component consists of either a variant and an acceptance test or two variants and a comparison algorithm, such that an output is produced only if either the acceptance or the comparison between the results of the two variants is successful, respectively

6.2. Intrusion Self-Checking Programming. Borrowing ideas from intrusion tolerant NVP and RB, a basic architecture for the intrusion tolerance counterpart of SCP, called iSCP, consists in exploiting different protection layers to enhance defense against attacks. The configuration proposed in Fig. 7 employs self-checking components consisting of two variants and a comparator. Note that the common mode failure between the two coupled variants could be a rare event when accidental faults are assumed, but intentional attacks are expected to be smart, thus making the becoming a phenomenon that needs to be mitigated from the security perspective (as also pointed out in (Rodriguez *et al.*, 2015)). Therefore, in the proposed iSCP configuration one variant is assigned the highest protection level L_0, as for the comparator component. Actually, within the scheme in Fig. 7, since

variants in L_0 are protected against attacks, but can still suffer from an accidental fault, it is relevant to distinguish a value failures due to accidental causes from i value failures due to intentional attacks, so that $f = a + i$, where $0 \leq a < n$ and $0 \leq i < n_{\mathrm{SC}}$. Moreover, it is needed to assume that the wrong results produced by accidental causes are different from the ones produced by intrusions; however, this is a reasonable assumption, since otherwise the attacker needs to read the accidentally wrong value.

In case the self-checking component results from a couple *variant and acceptance test*, for the same reason discussed above it is appropriate that the acceptance test receives a higher protection (L_0).

The execution pattern of SCP with respect to selecting the final outcome is sequential, similarly to that of the RB scheme. Therefore, the same considerations already made for iRB also apply to iSCP for what concerns both the addition of confusion (through additional self-checking components and the random order in checking the self-checking components' result) and in scheduling variants rejuvenation. Note that, since a self-checking component based on a couple of variants requires both variants to be actively working, it appears advantageous that both variants undergo rejuvenation at the same time, and then having the most recently rejuvenated self-checking component to act as the first component at each execution.

Confusion can be exploited in different forms. One can be to randomly build the order of the self-checking components, so that from one execution to another the final outcome is provided by a different one. However, this is feasible only if the variants have similar quality of service characteristics. Another can be add c variants all under L_1. If the attackers cannot identify the protection layer a variant belongs to, this makes also possible to hide the number of participating variants; otherwise, this adds confusion only to the n_{SC} variants that are already under L_1, allowing the attackers to focus on those. Finally, additional variants can be deployed in couples, one in L_0 and the other in L_1, as the participating ones.

6.3. Consensus Recovery Block. As reported by Randell and Xu (1994), the Consensus Recovery Block (CRB), attempts to combine aspects of the RB and NVP schemes, by reducing on one side the importance of the acceptance test used in RB and handling the case where NVP does not employ a sophisticated voter able to recognize multiple correct outputs. In CRB the variants are ranked and, on invocation, all variants are executed in parallel and their results submitted to a voter. The original formulation of the scheme (Scott *et al.*, 1985) is based on the assumption that there is no common mode failure, so erroneous results do not coincide. Therefore, agreement between the outcomes of two variants is sufficient to deliver this value as the final result. However, in a more general formulation, which is comprehensive of less restrictive failure model assumptions, the voter can be based on a simple majority of m variants or another plurality criterion to consider an outcome to be successful. If there is no majority, then the result of the variant with the highest ranking is submitted to the corresponding acceptance test. If this fails then the next variant in the order is selected. This continues until all variants are exhausted or one passes the acceptance test.

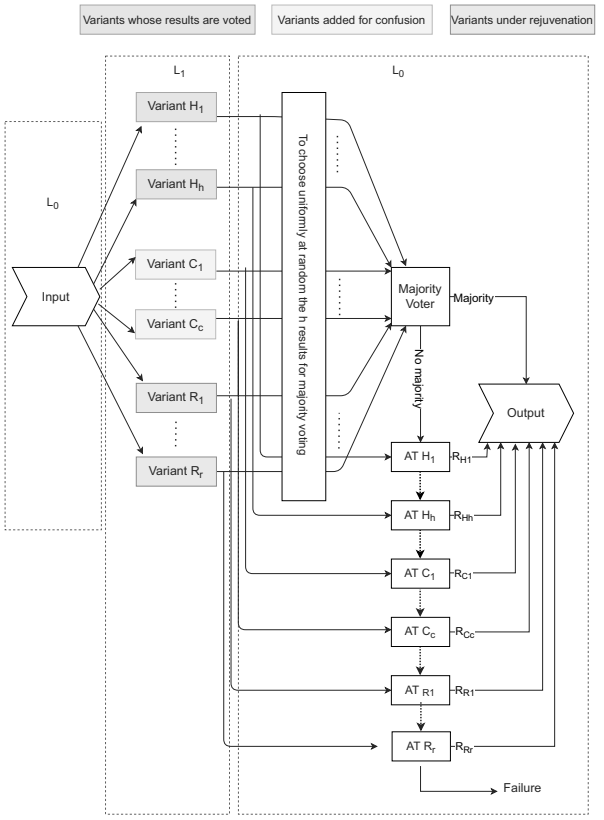

Fig. 8. Logical organization of the iCRB scheme, including two protection layers, and n variants in the roles of: h variants contributing results to the adjudicator, additional c variants to produce confusion and additional r variants to allow rejuvenation phases. As executions progress, each variant typically changes role from one to another.

Notice that this schema is, from one hand, a parallel recovery block with a pre-test about consensus, and from the other hand an NVP with an adjudicator that is more sophisticated than the simple voter.

6.4. Intrusion Consensus Recovery Block. As for the other proposed intrusion tolerant alternatives to basic fault tolerance strategies, also for an intrusion version of CRB (iCRB) a first measure to adopt is higher protection of the most critical components of the scheme, i.e.,

the implementation of the two-step logic (voter and acceptance test, which are assigned protection level L_0) with respect to variants (which are assigned protection level L_1).

Since CRB is an hybrid between NVP and RB, protection techniques already discussed when presenting intrusion tolerance alternatives of NVP and RB could be considered for iCRB proposals. In particular, the architecture depicted in Fig. 8 is suggested, where confusion aspects obtained through addition of extra variants whose outputs are not considered by the voter component are exploited. In the most favourable case where variants of the same quality of service are employed, the outputs considered by the voter can be randomly chosen at each execution, to enhance the attacker's confusion level. Then, in case the voting phase is not successful and acceptance tests are activated, the output of previously non participating variants can be checked by the respective acceptance test (provided they are available) or not, depending on the degree of reliance that can be put on them.

If affordable from the overall budget perspective, the presence of extra redundancy favours the usage of rejuvenation actions, as a further protection measure, with expected benefits as already previously discussed. In addition, changing the logic of the overall adjudication function to have the acceptance test applied to the result selected by the voter, in case this happens, strengthen the scheme to a greater extent. In fact, taking advantage of the availability of both the voter and the acceptance test, making such double check enhances the chance to counteract potential intrusions.

6.5. Self-Configuring Optimistic Programming. The Self-Configuring Optimistic Programming (SCOP), proposed by Bondavalli *et al*. (1993), maintains the logic of NVP unaltered but schedules the execution of the variants in phases, instead of the parallel execution of all the variants, to promote efficiency. SCOP starts executing the minimum number of variants that, if all correct, satisfy the adjudicator criterion and the scheme terminates, based on an optimistic vision that high quality versions are typically employed to build redundant organization for critical domains. If this is not the case, a new execution is started, involving the minimum number of variants among the still to execute ones, such that, if successful, will contribute together with the variants already executed in the previous phases to satisfy the adjudicator criterion and terminate the overall execution. This pattern is repeated until a successful result is found, or all the variants are exhausted. To make clearer how SCOP works, consider a configuration using 5 variants to tolerate 2 errors, with a majority voter as adjudicator. In the first phase, 3 variants are executed (minimum number sufficient to satisfy the majority criterion over 5 variants). If the phase ends with

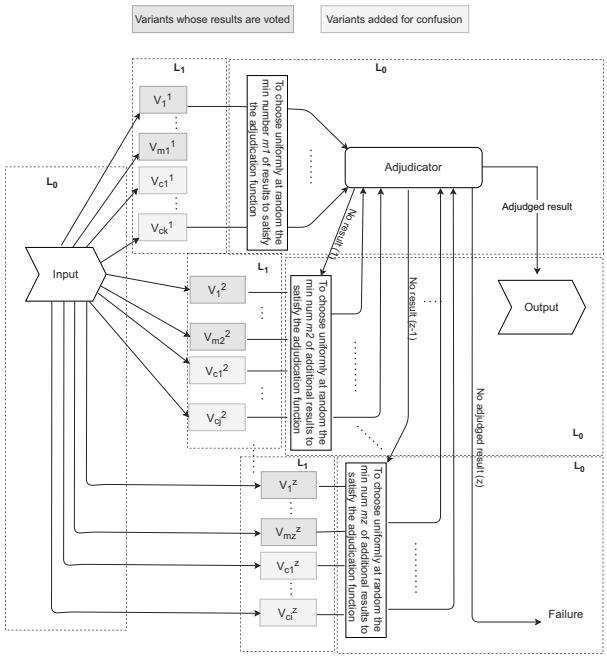

Fig. 9. Logical organization of iSCOP with execution in more phases, two protection layers (L_0 and L_1) and extra variants for confusion. V_i^j indicates variants executed at phase j, m_j is the (variable) minimum number of variants that need to be executed at phase j to attempt satisfying the adjudication criterion, given the pattern of results obtained in the previous phases.

3 consistent results, such result is the final output and the scheme ends. Instead, if only 2 results are consistent, a second phase is started, where one of the two remaining variants is executed and its result voted with those of the first phase. Then, upon 3 consistent results out of the 4 obtained so far, the scheme ends, otherwise a final phase employing the last remaining variant is performed. Depending on the last obtained result, the scheme can end with a majority result, or with a failure notification (or default value). Instead, in case the first phase ends with 3 different results, in the second phase both the 2 remaining variants are executed, at the end of which the 5 results are voted and the scheme ends, again either with a majority result, or with a failure notification (or default value). When the majority voter is used, the number of executed phases ranges between one (as it would be expected in most of the cases) and $\lfloor (n+1)/2 \rfloor$ (the worst case, when each execution after the first phase always involves just one variant).

6.6. Intrusion SCOP. To cope with intrusions, iSCOP maintains the same logical organization as SCOP, but adopts both high protection layer and additional variants for confusion. Figure 9 illustrates the scheme.

iSCOP execution consists of a variable number

of phases, where the variants to execute at each phase are optimistically chosen on the basis of the adjudication function and the observed syndrome of results accumulated along the executions.

Similarly to iNVP, additional variants for confusion may be employed at each phase, so a mechanism is needed to uniformly select at random the results needed at each phase for adjudication purposes.

The introduction of protection levels follows the same rationale as for the previous schemes: the variants are assigned protection level L_1, while the selector of variant results to be adjudged, the adjudicator and the communication channels are assigned the higher protection level L_0.

For the sake of simplicity, in Fig. 9, the rejuvenation mechanism has not been included. Of course, for this scheme it is appropriate to treat variants with periodic rejuvenation phases. Then, the same considerations already provided when discussing the iNVP scheme regarding the opportunity to exploit the most rejuvenated variants in execution phases instead of random selection apply also to iSCOP.

7. Practical advice

To better support the understanding and selection of the appropriate intrusion tolerance scheme to adopt for specific purposes, practical advice is provided in this section, including a study with focus on confusion.

7.1. Selection of schemes' parameters. To help configuring the appropriate scheme, in the following the solutions introduced in the previous sections are formulated in terms of their parameters. In particular, indications about redundancy levels, degrees of variants under rejuvenation, as well as relation with available sites are provided.

First, Table 1 reports the number of variants that are needed by each of the five fault tolerance schemes considered in this study, to tolerate f (in the worst case common-mode) value failures and k omission failures, occurring simultaneously. Of course, f or k can be 0, in case only omission failures or only value failures are assumed, respectively. Also, indication about the kind of decision function adopted by the scheme is included. For NVP, the simple majority voting is assumed, and for SCP the self checking component is obtained through comparison of two variants outcomes. Observe that, when only omission failures are considered, the decision function based on voting is simplified to be just the selection of the received variant's value (in accordance with the omission failure assumption, if a variant output is issued, it is correct). Moreover, similar formulations can be easily derived for determining the number of required variants for NVP and SCP if a different voting function or realization of a self-checking component with respect to the one in Table 1 are adopted.

Table 1. Comparison of classical architectures: $n = h$ is the number of variants, f is the number of value faults, k is the number of omitted results.

Scheme	$n = h$	Decision mechanism
NVP	$2f + k + 1$	Relative, simple majority
RB	$f + k + 1$	Absolute, based on ATs
SCP	$2(f+k+1)$ with $f \leq 1$	Relative, compare two results
CRB	$2f + k + 1$	First relative and then absolute
SCOP	$2f + k + 1$	Relative, simple majority

Notice that, for NVP, CRB and SCOP, the majority is $m = \lceil (n+1)/2 \rceil$. Thus, if $k = 0$ then n is odd, so setting $f = m - 1 = (n-1)/2$, means that the scheme can tolerate f value faults. If $k > 0$ then n can be either odd or even. The k additional variants indeed promote not only omission failures tolerance, but also contribute to strengthen the scheme when there are only value failures. In general, if n is even then the scheme can tolerate up to $f = n - m$, whereas $f = n/2$ still guarantees fault detection (or, in the safety context, the scheme results in a benign failure). As an example consider the case $n = 8$, and then $m = 5$, with no omissions: if 4 variants agree on the result and other 4 agree on a different one then NVP cannot decide which one deliver in output (but can do detection); otherwise, if 5 variants agree on the result (and $f = 3$ agree on a different one) then the scheme can select the correct result. The new proposed intrusion tolerant alternatives to the schemes in Table 1, as recalled in Table 2, are formulated in terms of their parameters in Table 3. These schemes take advantage of additional features to better cope with intentional attacks, as deeply discussed in Section 3. Specifically, they consist in: (i) additional redundancy used as a stratagem to confuse the attacker; (ii) distribution of the variants on more sites; and (iii) periodic rejuvenation of variants, to contrast potential partial compromise of a variant already in place, or anyway to nullify potential gathered knowledge by an attacker about a variant. As previously introduced, c indicates the number of additional redundancy for confusion (and therefore h variants vote in the NVP-like schemes, or are considered in RB-like ones), s indicates the number of available sites, and r indicates the number of variants under rejuvenation. The formulas in Table 3 for the number of variants required by each scheme include these parameters c, s, r, in addition to f, a, i, k connected with the failure types. In particular, $k \geq \lceil n/s \rceil$ assures

Table 2. Intrusion tolerant schemes.

Acronym	Full name	Section	Figure
iNVP	intrusion N Version Programming	Section 4.2	Fig. 5
iNVP-R	iNVP with Random Participation	Section 4.2	Fig. 5
iNVP-D	iNVP with Deterministic Participation	Section 4.2	Fig. 5
iRB	intrusion Recovery Block	Section 5.2	Fig. 6
iSCP	intrusion Self-Checking Programming	Section 6.2	Fig. 7
iCRB	intrusion Consensus Recovery Block	Section 6.4	Fig. 8
iSCOP	intrusion Self-Configuring Optimistic Programming	Section 6.6	Fig. 9

Table 3. Number of variants $n = h + r + c$ employed in intrusion tolerant schemes, to tolerate k omission failures and f value failures (among which a are accidental and i intentional, i.e., $f = a + i$), accounting for rejuvenation (r), more location sites (s) and additional variants for confusion (c).

Scheme	h	
iNVP{-R,-D}, iSCOP and iCRB	$2f + \max\left\{k, \lceil \frac{2f+1}{s-1} \rceil\right\} + 1,$	where $s > 1$
iRB	$f + k + 1,$	where $n > s$
iSCP	$2\left(f + \max\left\{k, \left\lceil \frac{2f+2}{s-2} \right\rceil\right\} + 1\right),$	where $s > 2$ and $a \leq 1$

that the architectures continue to behave as expected if 1 site is disconnected, and then n is derived exploiting standard properties of the ceil function.

There is no exact indication on the amount of extra redundancy for confusion (i.e., c), since this choice is left to the system designer. What is expected is that at higher number of extra redundancy for confusion should correspond higher defense ability (and therefore higher dependability); however, this needs to be confirmed by quantitative analysis, that is planned as a future research study. Also the population of variants under rejuvenation is chosen by the system designer, trading between cost of rejuvenation and benefits in prolonging the life of correctly operating variants; so only its number r is accounted for in the formulas.

Starting from the observation that NVP with simple majority voting requires $n = 2f + k + 1$ as reported in Table 1, it is possible to gradually introduce sites, rejuvenation and confusion in iNVP. If the defender has s sites, the best strategy is to distribute as much uniformly as possible the variants among the sites. Thus, there are $\lceil n/s \rceil$ variants on the largest site, and then $k \geq \lceil n/s \rceil$, otherwise the isolation of the largest site produces more omissions than the tolerated ones. Applying standard properties of ceil function it is possible to relate n directly to f and k (datum), and s and r (designer choice), as in Table 3. However, the uniform distribution is not a compelling requirement, so other deployment policies can be adopted. As a general rule, the necessary condition to prevent the occurrence of a system failure, following the isolation of one site by an attacker, is that less than the number of variants whose results are needed to satisfy the adjudication function are allocated to any single site (a majority of variants, in case a majority voting is employed in the scheme, as for the case presented).

When confusion is adopted, n becomes the one reported in Table 3 for both deterministic and random strategies, and the same reasoning on how to distribute variants across sites applies to additional redundancy.

iRB requires a smaller number of variants, $n \geq f + k + 1$, with respect to iNVP, and n does not change when the isolation of a site is considered as long as $n > s$, that is usually the case, when both the $n-c$ participating variants and the c additional ones are distributed among the sites according to round robin policy. In fact, for $n \leq s$ the isolation of one site reduces the number of participating variants, and then the scheme is no more able to tolerate the required number of failures.

For degradable systems, i.e., where the variants produce result of different accuracy, and then are ordered accordingly, the best strategy to distribute the variants among sites is to deploy the primary on one site, the second alternate on another site, and so on till the first s variants are assigned; the remaining $n - s$ are then distributed round robin among the sites.

In iRB, even though only the result of a variant is selected and submitted to the corresponding acceptance test, other variants (chosen among those already available

Table 4. Comparison of the architectures with respect to time constraints (hard vs soft).

Scheme	Hard	Soft
iNVP{-R,-D}	OK (parallel exec.)	OK
iRB	KO (sequential exec.)	OK
iSC	OK (parallel exec.)	OK
iCRB	OK (parallel exec.)	OK
iSCOP	KO (sequential exec.)	OK

for tolerance or among additional ones) can be executed in parallel just to increase attackers' confusion. Thus, the number of variants in the iRB is $f + k + 1 + c$, and the c additional variants have to be deployed on different sites.

For iSCP, with $s = 1$ site, when $f = 0$, to tolerate k omission failures (that in the worst case are distributed one per couple in $n_{\mathrm{SC}} - 1$ self-checking components) $2(k+1)$ variants are required. If in addition there is $f = 1$ value failure then the required number of variants becomes $n = 2(k + 1)$. To tolerate $a \leq 1$ accidental and i intentional failures, iSCP requires $n = 2(k+1+a+i)$, with $a+i = 1$. For $s > 2$ the number of variants is reported in Table 3.

For iCRB, only the first phase, where iCRB behaves as iNVP, is relevant to determine the number of variants to employ. Thus, $n = 2f+k+1+c$. When considering s sites, the number of required variants is reported in Table 3. Regarding confusion in iCRB, even though the results of those variants that do not participate to the vote are considered in subsequent phases, being $n = 2f + k + 1 + c \geq f + k + 1 + c$, employment of further variants does not appear useful.

Finally, considerations about performance are summarized in Table 4. Without going in the detail of a huge variety of system organizations and application domains, the time requirements are abstracted at the level of *hard time constraints* and *soft time constraints*. The former indicates that violation of the time requirement has potentially heavy consequences for the system where the scheme is embedded, while the latter indicates a lower criticality of the time requirement.

Similarly to what can be observed for the original fault tolerant schemes that inspired the definition of the proposed intrusion tolerant alternatives, it can be roughly suggested that schemes based on parallel execution are adequate for hard time constraints, while schemes structured in sequential phases are risky from the hard time perspective. However, this is an indication, but not a definitive discrimination among the schemes considered. Indeed, while parallel execution allows to predetermine the worst case execution time of the slowest variant and so be sure of the maximum time required by an execution of the scheme, mechanisms structured in phases have variable execution time depending on the failures really experienced during the execution (they afford longer execution time in unfavourable scenarios, but save in executed variants in the more frequent favourable scenarios where no failures occur). However, also for these sequentially based solutions, the worst case execution time can be computed and, if adequate for the hard time constraint imposed by the application at hand, there is no objection on adopting one of them.

Of course, when soft time constraints are in place, any of the presented schemes can be applicable, and the choice will be in general operated in accordance to some other criterion.

7.2. Focus on confusion. Confusion increases at increasing of c, but scheme parameters impact on confusion is a complex subject. In order to isolate as much as possible the analysis of confusion from the other aspects addressed in the paper, and then promote the identification of simple trade-offs between cost and confusion, the following assumptions (corresponding to the worst case scenario) are made:

- an attacked variant is considered compromised;
- the attackers know all the scheme details, including parameters value;
- the evaluation does not explicitly consider accidental value failures nor omission ones, but only intentional value failures;
- the variants to be compromised are selected uniformly at random;
- no variant can undergo rejuvenation, i.e., $r = 0$.

Recall that for iNVP-R with simple voting the majority is $m = \lceil (h + 1)/2 \rceil$, and i indicates the number of compromised variants (that, under the assumptions considered, coincides with the attacked variants). Let t be the number of variants among the attacked ones that were actually participating to the voting, with $0 \leq t \leq i$, and recall that iNVP-R fails if $t \geq m$. For given n, c and i, and recalling that $r = 0$ (in this case $n = h + c$), the probability of successful intrusion tolerance is then

$$\mathbb{P}_i\{\text{iNVP-R OK}\} = 1 - \sum_{t=\max\{m,h+i-n\}}^{\min\{h,i\}} \frac{\binom{h}{t} \cdot \binom{n-h}{i-t}}{\binom{n}{i}},$$

where the addends are determined exploiting the hypergeometric distribution.

For instance, Table 5 compares $\mathbb{P}_i\{\text{iNVP-R OK}\}$ where $n = 8$ and for $h = 5, \ldots, 8$ at increasing of i.

Clearly, for $n = 8$ it makes no sense for the attacker to intrude more than 7 variants, and obviously for $h = 8$, i.e., iNVP without confusion, $i = 5$ is enough to defeat

the defender. Notice that, for $n = 8$, it is often reasonable to choose h equals to 4 or 5, even though the choice is strictly context dependent. For $h = 5$ the scheme can certainly tolerate $i = 2$, as the corresponding iNVP without confusion ($n = h = 5$), but in addition it can also tolerate $i = 3$ with reasonably high probability.

iSCOP is almost identical to iNVP-R, except for the fact that, being the number of executed variants dependent on the number of phases, the attacker has slightly less information than in iNVP-R. This difference, though, is negligible for the kind of analysis reported in this section, then $\mathbb{P}_i\{\text{iSCOP OK}\} = \mathbb{P}_i\{\text{iNVP-R OK}\}$.

Instead, applying confusion on iRB leads $\mathbb{P}_i\{\text{iRB OK}\} \approx 1 - i \cdot \mathbb{P}\{\text{AT}_1 \text{ fails}\}/(c+1)$, where only the primary is considered and $\mathbb{P}\{\text{AT}_1 \text{ fails}\}$ is the probability of its acceptance test failure. This is an approximation of the exact expression, that neglects the sum of products of alternates acceptance tests' failure, so applicable when $\mathbb{P}\{\text{AT}_j \text{ fails}\}$ is sufficiently small as we assume.

Similarly, $\mathbb{P}_i\{\text{iSCP OK}\} \approx 1 - 2 \cdot \mathbb{P}\{\text{AT}_1 \text{ fails}\}/(n+c)$ because in iSCP all the $n-c$ self-checking components run in parallel.

For iCRB, $\mathbb{P}_i\{\text{iCRB OK}\} \approx \mathbb{P}_i\{\text{iNVP-R OK}\}$ since the first phase is identical to iNVP-R, and subsequent phases add negligible contribution under the assumption that $\mathbb{P}\{\text{AT}_j \text{ fails}\}$ is sufficiently small, as done for iRB.

Finally, it is possible to characterize each scheme through averaging their probabilities of successful intrusion tolerance:

$$\mathbb{P}\{\text{iSCHEME OK}\} = \sum_{i=0}^{n} \frac{\mathbb{P}_i\{\text{iSCHEME OK}\}}{n+1},$$

and then compare the impact of confusion on the schemes. Consider, for instance, Fig. 10, where the bars are grouped by n, each bar corresponds to a scheme and reports min and max values of $\mathbb{P}\{\text{iSCHEME OK}\}$ when c lies between 0 and 2 for $n = 3$, and between 0 and 6 for $n = 8$. Here $P\{\text{AT}_1 \text{ fails}\} = 0.1$, i.e., the primary acceptance test coverage has been set to 0.9. Notice that $\mathbb{P}\{\text{iRB OK}\}$ and $\mathbb{P}\{\text{iSCP OK}\}$ increase at increasing of c, whereas $\mathbb{P}\{\text{iNVP-R OK}\}$, $\mathbb{P}\{\text{iCRB OK}\}$ and $\mathbb{P}\{\text{iSCOP OK}\}$ oscillate. Given the assumptions and setting of the analyzed scenario, iRB and iSCP are more intrusion tolerant than iNVP-R, iCRB and iSCOP, and confusion does not change much their tolerance. Instead, iNVP-R, iCRB and iSCOP are more sensitive to an increase of c, in particular for small value of n (that are more commonly employed).

Of course, relaxing the worst case assumption that an attacked variant is always compromised versus a more realistic situation where the attacker has a probability to succeed lower than 1 (as expected due to the positive impact of the employed diversity techniques), the probability of successful intrusion tolerance improves for each scheme. For example, when $i = 6$, the values of $\mathbb{P}_i\{\text{iNVP-R OK}\}$ are those corresponding to $i = 5$ in Table 5 if the attacker's probability to compromise a variant is 83%, and values corresponding to $i = 4$ if such probability lowers to 67%.

Table 5. $\mathbb{P}_i\{\text{iNVP-R OK}\}$ for $n = 8$ and $h = 3, \ldots, 8$.

	h					
i	3	4	5	6	7	8
0	1.00	1.00	1.00	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00	1.00
2	0.89	1.00	1.00	1.00	1.00	1.00
3	0.71	0.93	0.82	1.00	1.00	1.00
4	0.50	0.76	0.50	0.79	0.50	1.00
5	0.29	0.50	0.18	0.36	0.00	0.00
6	0.11	0.21	0.00	0.00	0.00	0.00

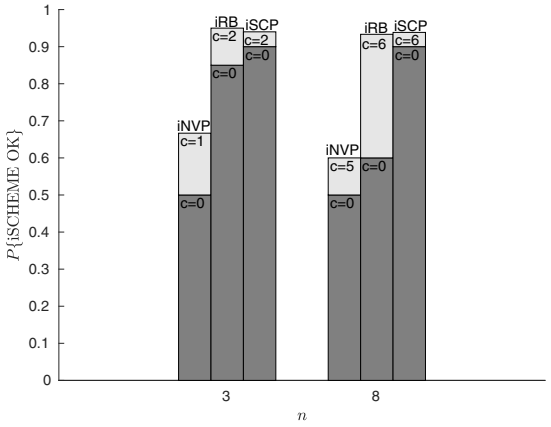

Fig. 10. Minimum and maximum value of $\mathbb{P}\{\text{iSCHEME OK}\}$ for $n = 3$ ($0 \le c \le 2$) and $n = 8$ ($0 \le c \le 6$), with a representative value of c.

The above analysis has shown a direction to assess the impact of confusion on the intrusion tolerance abilities of the proposed schemes. Of course, more scenarios need to be considered in future studies to draw more solid conclusions.

Although the number of variants called in place in this study appears rather high, obtaining $n = 8$ diverse system components is not difficult in principle, considering that systems are typically structured in hierarchical layers, from the application down to the middleware, operating system and hardware support. Therefore, adopting two diverse elements in three of the mentioned four layers (e.g., employing 2 operating systems, 2 middlewares or libraries and 2 implementations of the same application) results in n=8 different combinations, which are the needed variants.

8. Final discussion and future work

Enlarging the view from a computing element to other ICT components targeted by cyberattacks, considerations about the applicability of the proposed redundancy-based intrusion tolerance schemes are discussed, before concluding with future research directions.

8.1. Redundancy-based intrusion tolerance from the different system components' perspective. In the following, the redundancy-based intrusion tolerance solutions, developed in Sections 4–6 and schematized in Section 7.1, are briefly considered from the perspective of the different components of an ICT system, to which such schemes are intended to be applied.

Recalling from (Di Giandomenico and Masetti, 2021), the ICT components that can be the target of a cyberattack are grouped in the following three categories:

- *Computing element*, i.e., a component that is devoted to perform some kind of functionality, to provide a service to the requesting entity (a user or another component). Operating systems primitives, software applications and enterprise software are typical examples of this category.
- *Communication element*, i.e., the means through which information is delivered to/from computing elements, users and storage. The internet and the several wireless networks technologies are typical examples of this category.
- *Data storage element*, which includes different storage technologies used to retain digital data within a computer system architecture. The term storage may refer both to a user's data generally and, more specifically, to the integrated hardware and software systems used to capture, manage and prioritize the data. This includes information in applications, databases, data warehouses, archiving, backup appliances and cloud storage.

These three component categories are characterized by hardware/physical supports and software programs, either devoted to perform specific functionalities (computing element category) or to manage/control the operation of the hardware/physical support (communication and data storage categories).

It is underlined that the interest in this work is on cyberattacks, so the impact of an attack on a physical component can only occur through the software facilities that control/act on it. Direct physical attack to corrupt a portion of a physical medium (as it could be a memory cell or sector) is considered out of scope.

Following this observation, in principle any of the proposed redundancy-based intrusion tolerance schemes would be adequate for enhancing resilience of ICT components belonging to the three categories, taking into account the aspects discussed in Section 3 to support the most suitable selection among the several alternatives. However, while functional components employed at application level are typically developed as ad-hoc components to accomplish the activity the application is called to perform, the software supporting the operation of physical devices, as well as operating systems, libraries and the execution environment are typically off-the-shelf components. This implies that, to obtain the diversity advocated to be a fundamental aspect characterizing redundancy-based intrusion tolerance, full control by system developer is possible for in-house developments, while for the other software components the only option is to rely on what is available on the market.

Luckily, there is a wide range of options made available by ICT companies, each one embedding some peculiar aspects that make their products equivalent from the service point of view, but with differences in terms of how such service is provided. Open source repositories also help significantly, especially for what concerns libraries and execution environments. Therefore, the diversity principle the intrusion tolerance schemes are based on can be easily satisfied. Moreover, resorting to employ a variety of couples *physical device, managing software*, as it would be for communication networks and data storage components, enhances system resilience also against faults affecting the hardware part.

Coming to a conclusion from this discussion, it can be inferred that the presented redundancy-based intrusion tolerance schemes can be profitably exploited to protect ICT components. The highlights elaborated in Sections 7.1 and 7.2 help a system designer in selecting a suitable solution for the faced requirements and constraints.

8.2. Future work. This work focused on redundancy-based approaches to tolerate intrusions in ICT system components. The developed solutions are obtained by revisiting classical techniques proposed for fault tolerance purposes, with specific emphasis on additional features to contrast the effect of intrusions. Namely, NVP, RB, SCP, CRB and SCOP schemes have been considered and advanced with intrusion tolerance features consisting in: (i) additional redundancy used as a stratagem to confuse the attacker; (ii) adoption of protection layers; (iii) distribution of the variants on more sites; and (iv) periodic rejuvenation of variants, to contrast potential partial compromise of a variant already in place, or anyway to nullify potential gathered knowledge by an attacker about a variant. Quantitative formulations of the developed intrusion tolerance schemes in terms of their characterizing parameters have been also elaborated, with a special focus on confusion aspects, as practical support

for system designers in understanding and selecting an appropriate alternative among the several offered ones.

Future extensions of this study include: a) proposal of additional schemes, obtained through further combinations of the intrusion tolerance features recalled above; b) identification of interesting application scenarios, to adopt for concrete demonstration of the utility of the proposed intrusion tolerance solutions; c) extension of the architectural definition of the schemes with quantitative analysis to assess the capacity of each scheme in addressing dependability properties, as well as to determine dependability level of employed components (variants and adjudicators) needed to satisfy required system dependability indicators.

Acknowledgment

This work was partially supported by the project BIECO (www.bieco.org), which received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement no. 952702.

References

Alladi, T., Chamola, V. and Zeadally, S. (2020). Industrial control systems: Cyberattack trends and countermeasures, *Computer Communications* **155**: 1–8.

Archer, D.W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J.I., Smart, N.P. and Wright, R.N. (2018). From keys to databases—Real-world applications of secure multi-party computation, *The Computer Journal* **61**(12): 1749–1771.

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

Avizienis, A., Laprie, J.-C., Randell, B. and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing, *IEEE Transactions on Dependable and Secure Computing* **1**(1): 11–33.

Babay, A., Tantillo, T., Aron, T., Platania, M. and Amir, Y. (2018). Network-attack-resilient intrusion-tolerant SCADA for the power grid, *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxemburg, Luxemburg*, pp. 255–266.

Bondavalli, A., Di Giandomenico, F. and Xu, J. (1993). A cost-effective and flexible scheme for software fault tolerance, *Computer Systems: Science & Engineering* **8**(4): 234–244.

Di Giandomenico, F. and Masetti, G. (2021). Basic aspects in redundancy-based intrusion tolerance, *14th International Conference on Computational Intelligence in Security for Information Systems/12th International Conference on European Transnational Educational, Bilbao, Spain,* pp. 192–202.

Di Giandomenico, F. and Strigini, L. (1990). Adjudicators for diverse-redundant components, *Proceedings of the 9th Symposium on Reliable Distributed Systems, Huntsville, USA*, pp. 114–123.

Distler, T. (2022). Byzantine fault-tolerant state-machine replication from a system's perspective, *ACM Computing Surveys* **54**(1): 1–38.

Dohi, T., Trivedi, K. S. and Avritzer, A. (2020). *Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions*, WSPC, Singapore.

Garcia, M., Bessani, A., Gashi, I., Neves, N. and Obelheiro, R. (2014). Analysis of operating system diversity for intrusion tolerance, *Software—Practice & Experience* **44**(6): 735–770.

Gashi, I., Povyakalo, A. and Strigini, L. (2016). Diversity, safety and security in embedded systems: Modelling adversary effort and supply chain risks, *12th European Dependable Computing Conference (EDCC), Gothenburg, Sweden*, pp. 13–24.

Gorbenko, A., Romanovsky, A., Tarasyuk, O. and Biloborodov, O. (2020). From analyzing operating system vulnerabilities to designing multiversion intrusion-tolerant architectures, *IEEE Transactions on Reliability* **69**(1): 22–39.

Haphuriwat, N. and Bier, V.M. (2011). Trade-offs between target hardening and overarching protection, *European Journal of Operational Research* **213**(1): 320–328.

Hardekopf, B., Kwiat, K. and Upadhyaya, S. (2001). Secure and fault-tolerant voting in distributed systems, *IEEE Aerospace Conference Proceedings, Gothenburg, Sweden*, pp. 1117–1126.

Khan, M. and Babay, A. (2021). Toward intrusion tolerance as a service: Confidentiality in partially cloud-based BFT systems, *51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN21), Taipei, Taiwan*, pp. 14–25.

Khraisat, A., Gondal, I., Vamplew, P. and Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges, *Cybersecurity* **2**(1): 1–20.

Laprie, J.-C., Arlat, J., Beounes, C. and Kanoun, K. (1990). Definition and analysis of hardware- and software-fault-tolerant architectures, *Computer* **23**(7): 39–51.

Littlewood, B. and Strigini, L. (2000). A discussion of practices for enhancing diversity in software designs, *Technical Report DISPO_LS_DI_TR-04_V1_1d*, Centre for Software Reliability, City University, London, https://openaccess.city.ac.uk/id/eprint/275/.

Lyu, M.R. (1995). *Software Fault Tolerance*, John Wiley & Sons Ltd, Hoboken.

Majdzik, P. (2022). A feasible schedule for parallel assembly tasks in flexible manufacturing systems, *International Journal of Applied Mathematics and Computer Science* **32**(1): 51–63, DOI: 10.34768/amcs-2022-0005.

Mejdi, S., Messaoud, A. and Ben Abdennour, R. (2020). Fault tolerant multicontrollers for nonlinear systems: A real

validation on a chemical process, *International Journal of Applied Mathematics and Computer Science* **30**(1): 61–74, DOI: 10.34768/amcs-2020-0005.

Nascimento, A.S., Rubira, C.M.F., Burrows, R. and Castor, F. (2013). A systematic review of design diversity-based solutions for fault-tolerant SOAs, *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, EASE'13, Porto de Galinhas, Brazil*, pp. 107–118.

Obelheiro, R., Bessani, A., Lung, L. and Correia, M. (2006). How practical are intrusion-tolerant distributed systems?, *Technical Report DI-FCUL TR 06–15*, Department of Informatics, University of Lisbon, Lisbon, `https://repositorio.ul.pt/handle/10451/14093`.

Puig, V., Sauter, D., Aubrun, C. and Schulte, H. (Eds) (2018). *Advanced Diagnosis and Fault-Tolerant Control Methods* (special section), *International Journal of Applied Mathematics and Computer Science* **28**(2): 233–333.

Pullum, L.L. (2001). *Software Fault Tolerance Techniques and Implementation*, Artech House, Inc., Canton St. Norwood.

Qiu, J., Tian, Z., Du, C., Zuo, Q., Su, S. and Fang, B. (2020). A survey on access control in the age of Internet of Things, *IEEE Internet of Things Journal* **7**(6): 4682–4696.

Randell, B. (1975). System structure for software fault tolerance, *IEEE Transactions on Software Engineering* **SE-1**(2): 220–232.

Randell, B. and Xu, J. (1994). The evolution of the recovery block concept, *in* M. Lyu (Ed), *Software Fault Tolerance*, Vol. 3, Wiley, Chichester, pp. 1–22.

Rodriguez, M., Kwiat, K.A. and Kamhoua, C.A. (2015). Modeling fault tolerant architectures with design diversity for secure systems, *IEEE Military Communications Conference (MILCOM), Tampa, USA*, pp. 1254–1263.

Saidane, A., Nicomette, V. and Deswarte, Y. (2009). The design of a generic intrusion-tolerant architecture for web servers, *IEEE Transactions on Dependable and Secure Computing* **6**(1): 45–58.

Scarfone, K. and Mell, P. (2010). Intrusion detection and prevention systems, *in* P. Stavroulakis and M. Stamp (Eds), *Handbook of Information and Communication Security*, Springer, Berlin/Heidelberg, pp. 177–192.

Scott, R., Gault, J. and McAllister, D. (1985). The consensus recovery block, *Total System Reliability Symposium, Gaithersburg, USA*, pp. 74–85.

Sousa, P., Bessani, A. and Obelheiro, R. (2008). The forever service for fault/intrusion removal, *Proceedings of the 2nd Workshop on Recent Advances on Intrusion-Tolerant Systems, Glasgow, UK*, p. 16.

Tarraf, D.C., Kamhoua, C.A., Kwiat, K.A. and Njilla, L. (2017). Majority is not always supreme: Less can be more when voting with compromised nodes, *IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), Singapore, Singapore*, pp. 9–12.

Veríssimo, P.E., Neves, N.F. and Correia, M.P. (2003). Intrusion-tolerant architectures: Concepts and design, *in* R. Lemos *et al.* (Eds), *Architecting Dependable Systems*, Springer, Berlin, pp. 3–36.

Vöelp, M. and Verissimo, P. (2018). Intrusion-tolerant autonomous driving, *IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC), Singapore, Singapore*, pp. 130–133.

Wang, L., Ren, S., Korel, B., Kwiat, K.A. and Salerno, E. (2014). Improving system reliability against rational attacks under given resources, *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **44**(4): 446–456.

Ylmaz, E.N. and Gänen, S. (2018). Attack detection/prevention system against cyber attack in industrial control systems, *Computers & Security* **77**: 94–105.

Zhang, F., Kodituwakku, H.A.D.E., Hines, J.W. and Coble, J. (2019). Multilayer data-driven cyber-attack detection system for industrial control systems based on network, system, and process data, *IEEE Transactions on Industrial Informatics* **15**(7): 4362–4369.

Zhou, Y., Han, M., Liu, L., He, J.S. and Wang, Y. (2018). Deep learning approach for cyberattack detection, *IEEE INFOCOM 2018—IEEE Conference on Computer Communications Workshops, Honolulu, USA*, pp. 262–267.

Felicita Di Giandomenico is currently a research director at ISTI-CNR, Pisa, Italy, where she is leading the Software Engineering and Dependable Computing Research Laboratory. Her research activities include the design of dependable computing systems, software implemented fault/intrusion tolerance, and the modeling and evaluation of dependability attributes, with the focus on critical infrastructures. She was the chair of the IEEE TC on Dependable Computing and Fault Tolerance (from January 2017 to December 2018), and the chair of the IEEE/IFIP DSN Steering Committee (from January 2017 to December 2018). She is a member of the IFIP WG10.4 on dependable computing and fault tolerance, and a member of the Steering Committee of the *IEEE/IFIP DSN* and *EDCC* conferences.

Giulio Masetti is currently a post-doc researcher with the Software Engineering and Dependable Computing Research Laboratory, ISTI-CNR, Pisa, Italy. His research activities include modeling and evaluation of dependability attributes as well as modeling and analysis of interdependencies in critical infrastructures, also studying numerical analysis problems originating from models design.

Silvano Chiaradonna received his MSc degree in computer science at the University of Pisa, Italy, in 1992. Since 1992, he has been working on dependable computing, and modeling and evaluation of dependable systems. He is also been teaching modeling and evaluation of dependable systems, both at the University of Pisa and the University of Florence. Since 1999, he has been a researcher with ISTI, Pisa, Italy. During these years, he has been involved in European and national projects. He had been a fellow student with IEI-CNR, now merged into ISTI. His research interests include design of dependable computing systems, software and system fault tolerance, stochastic methods and techniques for quantitative analysis of dependability, modeling and evaluation of dependability attributes such as reliability, availability and performability, interdependence modeling, and analysis in critical infrastructures (in particular, smart grids).

Received: 30 December 2021
Revised: 9 May 2022
Accepted: 30 June 2022