Challenging Malicious Inputs with Fault Tolerance Techniques

Bruno L. C. Ramos
brunolcr@yahoo.com.br

Black Hat Europe 2007

Abstract
Most of the attacks attempt an initial activation which means the first occurrence of an error provoked by the fault. If its unable to stop the propagation, a fault will be transformed into a failure, causing consequences. This paper presents an exploratory research about the integration of fault tolerance aiming defenses against malicious inputs. When a fault occurs, these techniques provide mechanisms to prevent the occurrence of software systems' failures. Several programming methods that are used by several software, fault tolerance techniques include: assertions, checkpointing and atomic actions. Some basic and classic techniques provided by software fault tolerance that will be covered are: Recovery Block, N-Version Programming, Retry Blocks and N-Copy Programming.

1. Introduction

While the techniques used in the prevention of flaws work anticipating the occurrence of them, the fault tolerance techniques doesn't work with flaws anticipation. Even if the best staff, practices, and tools are used, it would be very risky to assume the software developed is error-free.

In spite of the intensive use of tools for verification and validation of the software, this will always be submitted to flaws. The fault prevention conditions (intolerance), want be successfully reached, independently of the great effort and resources involved.

A programmer's mistake leads to a flaw. The consequence of these fails, is a latent error in the program. When this error is activated, through certain conditions, it leads to an effective error; in case this program generates an erroneous result, that it comes to affect the offered service, a failure is produced.

The operation of the system will be reliable until the occurrence of the manifestation of a simple error that is latent. Latency is the meantime between the fault occurrence and its initial activation as an error. An error occurs in a module when its actual state deviates from its desired or intended state. An active fault is one that produces an error. The transformation of a fault into an error is called the fault activation.

2. Threats

The intruders: malicious entities that have no authority but attempt to intrude into the system and to alter service or halt it, alter the system's functionality or performance, or to access confidential information. They are hackers, malicious insiders, agents of hostile governments or organizations, and info-terrorists.

Any input received from users and external systems, and the influence upon the data by that input cannot be trusted. It may be structured to cause malicious effects into the program's behavior.

Malicious logic faults, that encompass development faults such as Trojan horses, logic or timing bombs, and trapdoors, as well as operational faults such as viruses, worms, or zombies. Definitions for these faults are as:

• **logic bomb:** malicious logic that remains dormant in the host system till a certain time or an event occurs, or certain conditions are met, and then deletes files, slows down or crashes the host system, etc.
• **Trojan horse:** malicious logic performing, or able to perform, an illegitimate action while giving the impression of being legitimate; the illegitimate action can be the disclosure or modification of information (attack against confidentiality or integrity) or a logic bomb;
• **trapdoor:** malicious logic that provides a means of circumventing access control mechanisms;
• **virus:** malicious logic that replicates itself and joins another program when it is executed, thereby turning into a Trojan horse; a virus can carry a logic bomb;
• **worm:** malicious logic that replicates itself and propagates without the users being aware of it; a worm can also carry a logic bomb;
• **zombie:** malicious logic that can be triggered by an attacker in order to mount a coordinated attack.

3. Fault Tolerance

The classic definition of software fault tolerance is: using a variety of software methods, faults are detected and recovery is accomplished [1].

All fault tolerance it bases on the redundancy generation, either for the detection or for the error recovery. The redundancy in software is not a simple copy of programs, but the redundancy specification, in other words, the generation of equivalent programs or functionality routines, of a same initial specification.

The key for the acquisition of a high degree of fault tolerance is the ability to detect errors with the most possible brevity, after its occurrence, and to avoid the propagation of the erroneous information, through the system. The process of fault tolerance involves many defined stages, from the phase of errors detection, through the confinement phase and evaluation of the propagation of erroneous information, until the recovery phase and treatment of the flaw.

The techniques are based on error detection and correction mechanisms, necessitate an explicit detection of errors produced by the faults, then the usage of means allows the correction of these errors.

3.1 Error Detection

It is possible the usage of the fault tolerance techniques, but, in first place, it is necessary implementation verification tests in which erros would be detected in the application. This is the most important stage and it is the starting point of the fault tolerance techniques.

The redundancy implied in this approach is qualified as active redundancy, as it requires an explicit activity to detect the appearance of errors and to handle them.

3.1.1 Replication

In these tests copies of the system are used or modules that it composes, being confronted with the exits of the copies when submitted the same entrances. In case the results are distinct a mistake is detected. In case the confrontation of the results is among two responses the verification is executed through comparison. In case the confrontation of the results is between three or more responses, the verification is accomplished through voting. The measure that increase the amount of responses, the costs can be high in change of a small increment in the reliability.

3.1.2 Temporal

They are tests that make watch for measurement of an interval of time expected, associated wich the operation of a module or the system as a completely. The detection of a mistake happens when the watch expires without there is any answer of the module or system.

One of the watch types this related with the fixation of an interval of time for accomplishment of a certain task. Another watch type quite used it is known as watchdog timer.

3.1.3 Diagnosis

They are tests foundations in the application of incentives in the entrance of the system for which the correct exits are known. The exits of the system are compared with the expected values, signaling the presence of flaws in case of discrepancy.

In these types of tests, the interest concentrates on the conduct of the system and not in the conduct of the modules. It is a test that consumes a time of quite long processing, besides consuming resources of the system. For these reasons, it is used as a test second moment, that should be executed in the moments free from the processor (" idle times")

3.2 Types of Error Recovery

Once detected the error the appraised ones your consequences, the system needs to be placed in a very defined state and error free, in way the one that can continue your operations.

3.2.1 Backward Recovery

This recovery type undoes the executed actions and it comes back to a consistent state in which already has past, or for which could have past, in way to continue starting from that point. The backward recovery perform with a mechanism of independent general use of the application, meantime could be implemented as a returned extension the an application it specifies.

The backward recovery offers the implementation possibility as a mechanism or an extension. The implementation is inside a construction of the translator which implements the easiness specified in the interpreted interface; The implementation through an extension makes use of a group of instructions of the necessary system for the accomplishment of some specific task. The backward recovery process demands the establishment of a point, during the execution of the system, denominated recovery point, which is responsible for the preservation of appropriate information for subsequent recovery, in case of error.

The recovery points are said active starting from the moment in that are established until the moment in that are discarded. The recovery areas are the periods corresponding to the interval between the establishment and the discard of the recovery points.

3.2.2 Forward Recovery

The forward recovery technique, it is not used as a mechanism, in spite of a larger implementation simplicity, for being dependent of the application. The forward recovery acts when an abnormal state is detected, with the suspension of the processing to that more favorable conditions are found.

This recovery type ignores part or the whole work, and it continues with what it is supposed correct, it corrects the erroneous behavior for the sending of compensation information. The forward recovery is dependent of the application one time that requests the previous knowledge of the structure of the system, in way to be early the flaw.

3.3 Redundancy

The redundancy in software consists of the introduction of such extra elements as instructions, segments of programs, programs etc..., in way to assure that fail they can be tolerated, for the substitution or masking of the faulty element.

3.3.1 Masking

The method maintains the copies of a certain element permanently connected and energized, and has advantage the instantaneous in the elimination of faulty elements.

The use of this technique bases on the hypothesis of the flaws of the redundant elements they be independent of statistics. The action of this technique is transparent to the users, and your performance is instantaneous and immediate.

3.3.2 Substitution

The use of this technique involves two different stages: The first stage the system detects the presence of flaws, and the stage following search your elimination through recovery actions. The execution of these two stages demands from the system an extra time of processing, time that should be foreseen in the moment of the specification of the system. In case this additional time is not enough for the execution of the recovery actions, extra actions should be considered. This technique ignores the human action in the recovery process, and therefore it turns the system solemnity-reparable.

3.4 Robust Software

Although most of the techniques and approaches to software fault tolerance use some form of redundancy, the robust software approach does not [1].

An advantage of robust software is that, since it provides protection against established, input-related problems, these erros are typically detected early in the development and test process.

A disadvantage of using robust software is that, since its checks are specific to input-related faults as defined in the specification, it usually cannot detect and tolerate any oher less specific faults.

4. Fault Injection for Fault Tolerance Assessment

Software fault injection is the process of testing software under anomalous circumstances involving erroneous external inputs or internal state information [2].

The main objective is to test the fault tolerance capability through injecting faults into the system and analyze if the system can detect and recover from faults as specified by the system or anticipated by the customers of the system. The results can lead to either fixing of individual software bugs or discovery of design deficiencies in system fault tolerance.

5. Programming Techniques

5.1 Assertions

Assertions are used to help specify programs and to reason about program correctness. Several modern programming languages include an assertion statement, which is essentially an assertion that is checked at runtime. If an assertion evaluates to false, an "assertion failure" results, which may cause execution to abort or cause the failure to be recognized in some other way. The use of assertions helps the programmer design, develop, and reason about a program. The use of assertion statements provides additional help during testing, for if an expected assertion evaluates to false, the programmer knows there is a bug somewhere. A major advantage of this technique is that when an error does occur it is detected immediately and directly, rather than later through its often obscure side-effects. Since an assertion failure usually reports the code location, one can often pin-point the error without further debugging.

5.2 Checkpointing

Strategy of recovery of the data based on the state of the system, that is, in case of flaw, the current state is discarded and changed by the state stored in the checkpoint. The information saved by checkpoints includes the values of variables in the process, its environment, control information, register values, and so on. The information should be saved on stable storage so that even if the node fail, the saved checkpoint information will be safe. For single node, single process systems, checkpointing and recovery are simpler that in system with multiple communicating and recovery are simpler than in system with multiple communicating processes on multiple nodes.

5.3 Atomic Actions

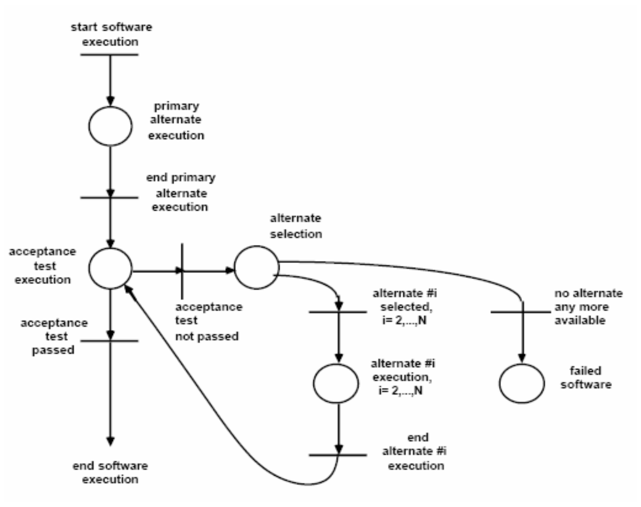
An atomic action is an "all-or-nothing" activity. It should either complete or leave the state of the system unchanged. This corresponds to the idea that there is no "visible" intermediate state for the process performing the action.

An atomic action should not interfere with any other one. There is no visible intermediate state for "other activities". Therefore the result of executing such actions concurrently should be the same as if they were executed sequentially.

6. Basic and Classic Techniques

6.1 Recovery Blocks

In this method, several alternatives are proposed for a certain block of instructions of the program. In case a mistake is detected, there will be the activation of the several alternatives, one per time, under the command of an acceptance routine, until the correct execution of the block or liberation of an error message.

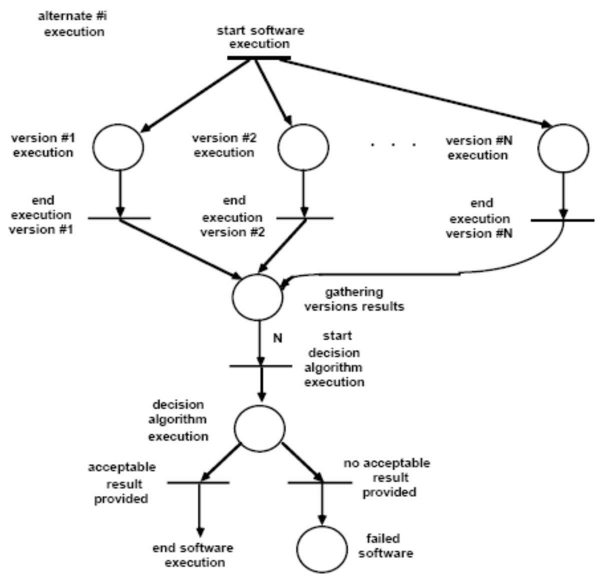

start software execution
primary alternate execution
end primary alternate execution
acceptance test execution
alternate selection
acceptance test not passed
acceptance test passed
alternate #i selected, i= 2,...,N
no alternate any more available
alternate #i execution, i= 2,...,N
failed software
end alternate #i execution
end software execution

The first providence to be electric outlet is segmentary the program in several blocks (modules) in way to facilitate the programming, maintenance and the introduction of the technique of recovery blocks, mainly in those blocks whose processing is critical in the time. The first action to be developed when penetrating in block a fault tolerant it is the establishment of a recovery point, which is responsible for the rescue of the global variables of the system they be altered inside of the block. Soon after the alternative is executed considered primary or main and submits the obtained results the an acceptance test for determination or not of your validity. In case the main alternative is rejected by the acceptance test the system will request the execution of the same program space through a secondary alternative that is available breaking same state that the previous alternative, in other words, of the recovery point initially established, so that all the alternatives are submitted the same initial conditions. In case this new alternative is rejected by the acceptance test, another alternative will be requested until that one of the alternatives is accepted by the acceptance test, or in the case of not more an available alternative to exist, that a mistake is signaled. In case one of the alternatives is accepted as valid, the flow of the program will go to another program block, giving pursuit the execution of the software.

6.2 N-Version Programming

In this method, starting from a same functional specification they are defined two or more versions of the software, being accepted with valid the versions whose results, for comparison, represent most. In this programming type the presence of erroneous versions is masked by the vote of majority of the other versions.

The principal objective of the N-version programming, is to mask the effects of the software flaws in the exits of the modules. This technique proposes the generation of different N-versions for a same software, breaking of a common specification. Once executed all the N-versions,



your individual results are compared, being accepted the result that has the vote of most and eliminated all that differ, being the corresponding erroneous versions extracted for subsequent flaw analysis.

After all the versions they have generated your results, these are submitted the a voting process.

Two are the factors that cause the abnormal end of a version:

- For watch end: Inside of an interval of pré-established time the version is not capable to liberate result;

- For error condition: Errors detected by the operating system during the execution of the version, as for instance, division for zero or overflow;

6.3 Retry Blocks

In addition to being a data diverse technique, the RtB technique is also categorized as a dynamic technique.

The RtB technique uses acceptance tests and backward recovery to accomplish fault tolerance. The technique typically uses one DRA and one algorithm. A watchdog timer (WDT) is also used and triggers execution of a backup algorithm if the original algorithm does not produce an acceptable result within a specified period of time. The algorithm is executed using the original system input. The primary algorithm's results are examined by an AT that has the same form and purpose as the AT used in the RcB. If the algorithm results pass the AT, then the RtB is complete.

```
...
START(max_no) {
    statements;
}
FINISHBY(post-condition);
```

6.4 N-Copy Programming

The processes can run concurrently on different computers or sequentially on a single computer, but in practice, they are typically run concurrently. NCP is the data diverse complement of N-Version programming (NVP).

The NCP technique uses a decision mechanism (DM) and forward recovery to accomplish fault tolerance. The technique use one or more DRAs and at least two copies of a program. The system inputs are run through the DRA(s) to re-express the inputs. The copies execute in parallel using the re-expressed data as input (each input is different, one of which may be the original input value). A DM exemines the results of the copy executions and selects the "best" resul, if one exists.

7. Decision Mechanisms

7.1 Voters

Voters compare the results from two or more variants. If there are two results to examine, the DM is called a "comparator." The voter decides the correct result, if one exists. There are many variations of voting algorithms. Voters tend to be single points of failure for most software fault tolerance techniques, so they should be designed and developed to be highly reliable, effective, and efficient [1].

7.2 Acceptance Tests

The acceptance tests are verifications of last instance. It is not important to know which generates the results for the program of the alternatives, however it is necessary that these are inside of the acceptance limit.

The acceptance tests should be simpler than the alternatives that intends to test, although it is not rarely of larger complexity. In case the acceptance test is not projected conveniently, two types of flaws can happen: disregard of correct alternative or acceptance of incorrect alternative.

Like this, the acceptance tests supply conditions so that the programmer can incorporate your own validation mechanisms for erroneous conducts of the program.

7.2.1 Reasonableness Tests

Reasonableness tests are used as ATs to determine if the state of an object in the system is reasonable. The results of many computations are bounded by some constraints. These constraints (e.g., precomputed ranges, expected sequences of program states, or other expected relationships) can be used to detect software failures.

7.2.2 Computer run-time tests

These test only for anomalous states in the program without regard to the logical nature of the program specification. Run time tests detect anomalous tates such as divide-by-zero, overflow, underflow, undefined operation code, end of file, or write-protection violations.

8. Applications of the Techniques

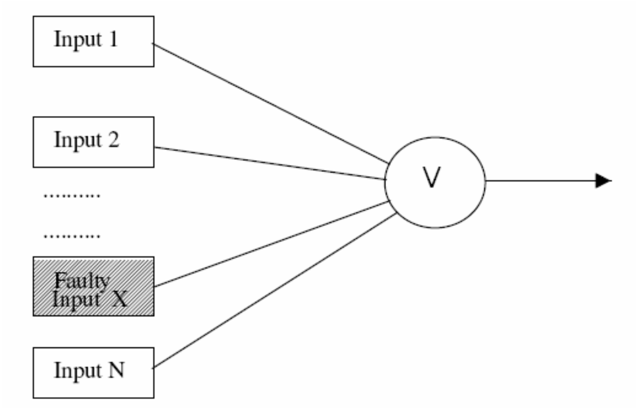
8.1 Recovering Exploration

The technique has as purpose to detect explorations and to recover the application of the flaw. When an exploration occurs, the execution of program is deviated. It is possible to use an interval of time for accomplishment of a certain task. If the time expires without an answer then the verification test fails and the error recovery happens.

The recovering exploration technique uses RcB to accomplish fault tolerance. When a checkpoint is established the values of data in memory, the processor context (register and instruction pointer) and the stack are saved. Watchdog time and backward recovery are used to detect and to recover the error respectively. Time-out via the watchdog occurs, resets the watchdog time, and restores the checkpoint

8.2 Anti-Fuzzing

Technique to prevent hacker discover zero day vulnerabilities in vendors. The idea is the vendors it accomplishes your tests, more hacker not. To use reusable software components that can be linked into an application. The logical elements are the active modules, bases on the hypothesis of the flaws of the redundant elements they be independent. The modules should be the most independent possible, so that the flaw in one of them doesn't cause flaw in the others. The inputs are distributed for the modules and in case the results are differents a error is detected. Use N-Version Programming in which each version is an module.



9. Implementation Methodology

Through experiences of allied decades to the works of researches and analysis, some procedures for implementation of fault tolerant systems, in hardware and software, they were established. The methodology of implementation of fault tolerance, it follows the following steps:

(A) It is defined an initial architecture and a technique for your implementation, under the hypothesis that the system is free from flaws;
(B) The goals are defined in terms of operation safety;
(C) They identify the classes of susceptible flaws to happen, and that they should be tolerated, being reduced this way, the specification costs and of implementation. The extension of the tolerance to be applied to each case is specified;
(D) They are specified as goals, operation safety's values, for the classes of individual flaws, or for all the classes jointly;
(E) A method is specified for the evaluation the reliability obtained by the system, in full detail;
(F) To the initial project they incorporate the mechanisms of detection of mistakes in hardware and in software, necessary to the attendance of all the classes of important flaws. The techniques of detection of mistakes are based on the specification of the project and they should satisfy to the reliability requirements and readiness wanted;
(G) Recovery algorithms are defined that will be worked after the greeting of the originating from sign the detection mechanisms. The recovery mechanisms involve such actions as; location of the flaw, restart of the operation, restoration of the state recovery initial, etc..
(H) It is come the evaluation soon after from the fault tolerant system, through:

    (1) Analytic modelling;
    (2) Simulation
    (3) Combination of the analytic method with simulation;

(I) The reliability is compared of the fault tolerant system with the same measures of the system non tolerant. In the analytic model the physical parameters of the components, the structural parameters, the repair possibility and the quality of the covering are used;
(J) It is proceeded to the refinement of the project: The initial evaluation shows the several contributions of reliability presented by the subsystems. Through this knowledge, it can be reinforced the subsystems of low reliability with fault tolerance techniques, and on the other hand to reduce the demands of other subsystems with super dimension redundancy.

10. Conclusions

The fault-error-failure model is central to the understanding and mastering of the various threats that may affect a system, and it enables a unified presentation of these threats, while preserving their specification via the various fault classes that can be defined.

Most, if not all, software fault tolerance techniques are based on some type of redyndancy (software, information, time). The selection of which type of redundancy to use is dependent on the application's requirements, its available resources, and the available techniques. The detection and tolerance of software faults usually require diversity.

Software fault tolerance is not a panacea for all our software problems. Since, at least for the near future, software fault tolerance will primarily be used in critical systems, it is even more important to emphasize that "fault tolerant" does not mean "safe," nor does it cover the other attributes comprising dependability.

Reference:

[1] Geffroy, Jean-Claude. 1Design of Dependable Computing Systems
[2] NASA. Software Fault Tolerance: A Tutorial http://library-dspace.larc.nasa.gov/dspace/jsp/bitstream/2002/12633/1/NASA-2000-tm210616.pdf
[3] Wikipedia. Fault-tolerant system http://en.wikipedia.org/wiki/Fault_tolerance
[4] A Conceptual Framework for Systems Fault Tolerance http://hissa.nist.gov/chissa/SEI_Framework/framework_1.html
[5] The Evolution of the Recovery Block Concept http://www.cs.ncl.ac.uk/research/pubs/books/papers/101.pdf
[6] Software for Safety-Critical Systems http://www.cis.strath.ac.uk/teaching/ug/classes/52.422/fault_tolerance.pdf
[7] Navigation Recovery Block Design Description http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19880008638_1988008638.pdf