Enhancing Survivability with Proactive Fault-Containment

Michael G. Merideth
School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3890
$mgm@cs.cmu.edu$

***Abstract*— Realistic survivable systems must assume that faults will occur within the system. When a malicious fault is activated, it may work to cause damage and to spread; until the system has recovered from this damage, it will have a lower degree of survivability than it did before the fault occurred. By proactively containing faults that would otherwise spread throughout the system, we can reduce the amount of potential damage to the system, and thereby maintain system survivability. Enabling proactive survivability carries with it a number of challenges, including the need to quantify survivability in order to justify the potential overhead of the proactive mechanisms, the need to select appropriate fault detection strategies, and the need to address runtime problems like deciding when and where to focus proactive effort.**

I. Introduction

When a malicious fault is activated in a system, it may work to cause damage and to spread to other parts of the system. Until the system has recovered from this damage, it will have a lower degree of survivability than it did prior to the fault. Survivable systems are designed to prevent and tolerate faults through the employment of four categories (resistance, recognition, recovery, and adaptation) of mechanisms [1]. However, previous survivable systems have tended to focus on stationary (non-propagating) faults, and, therefore, have been designed to recover reactively, *i.e.*, after a fault occurs. In the Starfish system [2], we additionally consider faults that can self-propagate, over covert/abnormal channels, to other parts of the system. If these self-propagating faults spread at a rate faster than the rate at which the traditional reactive mechanisms are able to recover the system, then, the survivability of the system may ultimately be compromised. However, it may be possible to monitor the system proactively for fault propagation, paying particularly close scrutiny to the parts of the system where faults have recently been detected or are suspected to exist. Based on this monitoring data, we can enable *proactive fault-containment*, whereby we take action to stop the spread of the faults, before they are able to taint other parts of the system. This may be done through a combination of *proactive-notification*, *proactive-isolation*, *proactive-adaptation*, and *proactive-recovery* mechanisms.

A *proactively survivable* system contains the spread of malicious behavior and adaptively sustains survivability. Our research on the Starfish system is focused on the creation of realistic survivable distributed infrastructures. Starfish is a proactively survivable system that employs an epidemiological model (whose terminology is summarized in Table I) [3] for analyzing and containing the spread of malicious faults. Such faults, if they propagate unchecked, can bring the system down. Another ramification of the unchecked spread of malicious behavior is the possibility of collusion; multiple corrupted nodes might lie in wait, and strike collectively when the system is vulnerable. One example of this is distributed denial of service, where several subverted computers can coordinate an attack to damage the system.

Castro and Liskov [4] have designed one strategy for proactive recovery, in which parts of the system are

Term	Definition
fault-transmission	propagation of fault
fault-containment	identification of the fault, isolation and restriction of the faulty nodes, and recovery
incubation-period	time from fault's initial presence to symptomatic-period
symptomatic-period	time during which fault may exhibit untoward behavior
communicable-period	time during which fault-transmission may occur
fault-recovery	faulty process is reinstated to correct operations
outbreak	one or more tainted nodes
epidemic	outbreak from which the system cannot recover

TABLE I
EPIDEMIOLOGICAL FRAMEWORK FOR PROPAGATING FAULTS

This work is supported by the Army Research Office through grant number DAAD19-02-1-0389 ("Perpetually Available and Secure Information Systems") to the Center for Computer and Communications Security at Carnegie Mellon University.

conservatively reinitialized periodically to remove any faulty behavior that might exist. A primary difference between their approach and ours, is that, under Starfish, proactive recovery needs to occur only if a fault exists in the system. Epidemiological models, similar to that of Starfish, have been previously explored in the context of computer-virus research [5], and also in the context of the transmission of malicious faults [6].

Because Starfish is designed to work in a realistic environment, we have attempted to make our assumptions as non-restrictive as possible. We assume a distributed asynchronous system with unbounded communication latencies and an inherently unreliable transmission medium. Our fault model considers processor- and process-crash faults, communication faults such as message losses and message corruption, and Byzantine/arbitrary faults in processes and processors.

This paper presents a number of research challenges and our candidate solutions; both will impact our ability to enable proactive survivability. Each section explores the state of our current work on the topic, and then presents any open issues. Section II surveys a number of techniques that we are evaluating for the purposes of building proactively survivable infrastructures. Section III cites the lack of survivability metrics as a problem in analyzing the effectiveness of a survivable system, and explains the need for new fault injection mechanisms, in order to provide the capability to simulate malicious faults at the system level. Section IV discusses when and where it is most appropriate for the system to focus its proactive fault-containment efforts.

II. Mechanisms for Proactive Fault-Containment

A. Recognition of Faults and of Fault-Transmission

Enabling proactive fault-containment hinges on the capacity to recognize faulty behavior and fault-transmission, and the ability to discriminate between a malicious process and a correct one. A combination of approaches for fault-detection are needed, because a malicious process that is intelligent enough to corrupt other processes might well appear, via the employment of covert channels for fault-transmission, to behave correctly to any number of fault detectors.

We expect that systems might employ Byzantine-fault tolerance mechanisms like [4] for their most critical processing. A key feature of many Byzantine-fault tolerant state-machine replication algorithms is that they detect, as well as tolerate, malicious faults. This Byzantine-fault detection can be used as one step in proactive fault-transmission containment. However, the weakness with this strategy is that the faulty component could output correct values whenever its outputs are to be subject to a vote, and thereby escape Byzantine-fault detection.

Strategies based on statistical techniques to detect the anomalous use [7] of any common resource (*e.g.*, the network or the file system) that could potentially impact other processes, must, in general, trade accuracy for coverage. The system would need to create logs of all sensitive operations, so that proof of the anomalous behavior could be presented to interested parties. These logs would need to be stored in such a way that the history of actions about a certain process could be retrieved by other processes, even if the process in question is corrupted or malicious; for security reasons, we would also want these logs to be tamper-resistant. Minimizing the intrusiveness of this log-based approach would be important for reasons of both system performance and privacy.

In addition to attempting to detect faults and their transmission, we might adopt a sand-boxing approach to strictly limit the use of certain functions, so that these functions could not be used for covert fault-transmission. To enable ease of programming, we might not want a rigidly restrictive environment with static policies; instead, functions might be restricted based on the context of their operations.

B. Containment, Recovery, and Adaptation

Once a fault is detected, we require some means of isolating and restricting it, as well as recovering from it, and adapting our system to handle the same or similar faults more effectively in the future. The system could make use of a trusted channel for proactive notifications about suspected or known fault-transmissions and faulty processes. Additionally, we might need a mechanism for increasing the priority of the proactive notifications over that of all other traffic. The danger with such a channel, is that it could be a target for abuse by an attacker; for example, a denial of service attack could be launched against the system itself through the repeated triggering of the proactive mechanisms.

Understanding the behavioral characteristics of a malicious fault is dependent on the system's ability to diagnose the corruption. Taking the correct proactive action would be assisted by having some sort of reference (of prior intrusions) to consult for each instance of a fault.

Biologically inspired immunization techniques [8] could allow for system adaptation toward increased survivability. Mechanisms to discriminate *self* from *non-self* could be used both for fault detection and advice during adaptation. Databases of signatures could be

built dynamically from characterization of system-call patterns, or built statically using techniques akin to those employed in virus-scanning programs. Such databases of attack signatures need not necessarily be built only through the occurrence of real attacks. Instead, similar to the process of vaccination in immunology, we might deliberately inject a controlled (and therefore, less dangerous) version of the propagated fault into the system, in order to trigger/educate the system's containment mechanisms. This training mechanism could increase the system's resistance to a real version of the attack.

III. Metrics and Evaluation

It is difficult to anticipate or to model realistically all of the types of faults that may occur in a real system. We feel that this is a primary cause of the dearth of faulty-case measurements and metrics. However, in order to measure the effectiveness of proactive fault-containment in improving the survivability of a system, we require ways to express and quantify survivability benefits.

A. Metrics

We consider three general contexts for which measurements would be of use: (i) the *fault-free case*, (ii) the *faulty reactive case*, in which the system is sustaining faults and using only reactive mechanisms to recognize, recover, and adapt, and (iii) the *faulty proactive case*, which is much like the faulty reactive case, except that the system may additionally be employing *proactive* fault-containment mechanisms. A survey of the literature has shown that, while many survivable systems have measured the performance overhead of their survivability mechanisms under the fault-free case, they very rarely consider either the faulty reactive case or the faulty proactive case. Thus, there is currently no objective way to (i) compare any two survivable systems/strategies, or (ii) measure how "good" a survivable system really is, under attack. To rectify this situation, in order so that we can evaluate the effectiveness of Starfish's proactive fault-containment strategy, we are developing and evaluating a number of metrics.

B. Fault Injection

Realistic survivable systems should assume that faults may represent malicious system-compromises that are *actively* being exploited by an intelligent adversary. Therefore, to show that our survivability mechanisms are of real-world value, we feel that it is important to be able to inject maliciously intelligent, faulty behavior into the system.

Manually orchestrating malicious behavior on a case-by-case basis is not scalable, would require the participation of a skilled human, and suffers from problems of ensuring representative fault-coverage. What we are really after, is a way to automate the fault-injection process.

IV. Allocating Proactive Effort

In a proactive system, if a fault is detected, then, the Starfish system must decide whether containment is appropriate. Candidate factors for consideration in making this decision include the runtime cost of the proactive mechanisms, the potential damage that the malicious fault can cause, and the speed of the fault-containment $vs.$ the speed of fault-transmission. If the decision to use containment is affirmative, then, Starfish must judiciously decide where to allocate its containment efforts, and at what rate. Assuming that the fault is detected in a highly connected section of the system, then, due to either resource constraints or cost-benefit analysis, it might not be practical to attempt containment universally throughout the system.

References

[1] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead, "Survivable network systems: An emerging discipline," tech. rep., CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, 1997.

[2] K. P. Kihlstrom and P. Narasimhan, "The Starfish system: Providing intrusion detection and intrusion tolerance for middleware systems," in *IEEE Workshop on Object-oriented Real-time Dependable Systems*, 2003.

[3] M. G. Merideth and P. Narasimhan, "Proactive containment of malice in survivable distributed systems," in *Proceedings of the 2003 International Conference on Security and Management (SAM'03)*, 2003.

[4] M. Castro and B. Liskov, "Proactive recovery in a Byzantine-fault-tolerant system," in *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI '00)*, 2000.

[5] C. Wang, J. C. Knight, and M. C. Elder, "On computer viral infection and the effect of immunization," in *16th Annual Computer Security Applications Conference (ACSAC)*, (New Orleans, Louisiana), December 2000.

[6] M.-J. Lin, A. M. Ricciardi, and K. Marzullo, "A new model for availability in the face of self-propagating attacks," in *New Security Paradigms Workshop*, (Charlottesville, VA), September 1998.

[7] R. A. Maxion and T. N. Townsend, "Masquerade detection using truncated command lines," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN '02)*, pp. 219–228, 2002.

[8] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer immunology," *Communications of the ACM*, vol. 40, pp. 88–96, October 1997.