Proactive Containment of Malice in Survivable Distributed Systems*

Michael G. Merideth and Priya Narasimhan
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3890

Abstract *The uncontrolled propagation of faults due to malicious intrusion can severely decrease system performance and survivability. Our goal is to employ available information about known or suspected faults in order to provide* collision-avoidance *and* epidemic-avoidance*. We* proactively *make use of knowledge of faults to notify potentially damaged areas of the system, in order to contain the tainted parts. Our objective is to lessen the impact of an intrusion, by spreading the performance cost of recovery over a controlled period of time.*

Keywords: Byzantine, collusion, intrusion, proactive, replication

1 Introduction

Survivability is the ability to continue to operate correctly, despite the presence of malicious or arbitrary faults; this ability is crucial to systems that can impact human life, $e.g.$, the electric power grid. Most survivable systems detect a malicious fault, remove the fault from the system, and then attempt to recover from the fault. This approach is feasible as long as the fault is isolated, and has not pervaded into the system.

In distributed systems, which contain interconnected components that communicate with each other, it is possible for a fault to taint other parts of the system before being detected and removed. Thus, not accounting for the propagation of a fault has four consequences: (i) it underutilizes valuable information, which is probably available in the system, that, if applied, might check the spread of the fault, (ii) the system waits to detect a fault in possibly tainted components (the "innocent until proven guilty" strategy) instead of proactively seeking out these components to initiate their recovery, (iii) a series of such propagated faults could lead to the number of tainted components being sufficiently large to prohibit the reliable use of the system, and (iv) even a small number of components with undetected faults could severely disrupt the system through collusion and coordinated action.

We borrow, from epidemiology, existing terminology that we adapt in order to discuss some of the key aspects of our system. In this paper, *fault-transmission* refers to the propagation of a fault, analogous to the spread of an infectious disease. *Outbreak* refers to the case when the fault-transmission has led to the tainting of one or more nodes in the system. An *epidemic* is an outbreak from which the system cannot recover. *Containment* involves identifying the fault, isolating or restricting the faulty nodes, and recovering from any adverse effects. There are four phases in the lifecycle of a faulty process: (i) *incubation-period*, the time from the fault's initial presence in the system, up to the (ii) *symptomatic-period*, when the fault starts to exhibit malicious/untoward behavior, until (iii) *fault-removal*, when the fault is removed from the system, and finally, (iv) *fault-recovery*, when the faulty process is reinstated to correct operation. Fault-detection can occur anytime during (ii). The *communicable-period*, when the fault is capable of being transmitted to the rest of the system, can occur anytime between the onset of (i) and the completion of (iii).

In this paper, we propose that by proactively probing for, and using, information extracted from the system when a fault is detected, we can improve both system performance and survivability. In addition, if we incorporate mechanisms into survivable infrastructures to enable them to yield this information readily and rapidly, we can develop systems that exhibit *collusion-avoidance* and *epidemic-avoidance*. Our primary contribution is the recognition that intrusion information can be (but unfortunately, is often not) employed to enable the containment of fault-transmission, and, thereby, to maintain control over the behavior of the system.

*The authors wish to acknowledge support through the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298.

The rest of this paper is organized as follows. Section 2 describes our working assumptions, and introduces proactive-containment as a solution to the fault-transmission problem. Section 3 provides a way of reasoning about the tradeoffs between performance and proactive recovery. Section 4 discusses mechanisms to support proactive containment. In Section 5, we show how our work builds on existing survivability foundations, and Section 6 concludes this paper.

2 Proactive Containment

We consider a distributed asynchronous system with unbounded 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. Using an underlying secure reliable totally-ordered group communication system [1, 2, 3], we can tolerate the communication faults and the processor-level arbitrary faults of interest to us. Our previous research [4] has shown us that the process-level fault-tolerance guarantees need to be layered over the processor-level fault-tolerance guarantees to ensure Byzantine-fault tolerance at the process and the processor levels.

Byzantine-fault detection [5], provided by the underlying secure group communication system, allows us to identify a malicious processor. Using active replication [6] with majority voting, we can tolerate processor- and process-level crash faults, as well as arbitrary faults and value faults at the process level. As is common with systems that employ active replication, we assume that the application is deterministic.[1]

We define *inter-process distance (IPD)* as the virtual distance between any two processes in the system, *e.g.*, an IPD of one from a faulty process refers to any process that is one "hop" downstream from the fault, *i.e.*, it receives messages directly from the faulty process, whereas an IPD of two refers to a process that is two "hops" downstream from the fault. The concept of fault-transmission is illustrated in Figure 1 (the fault-free case is not shown in the figure). The example shows communication between Byzantine-fault tolerant process groups. The replicas in a process group work together to provide a single reliable service.

Figure 1(a) shows that a replica in group A has become maliciously faulty, through some external or internal event or interaction. At this point, the malicious process has not yet been detected as faulty and, thus, the fault is latent. In Figure 1(b), the malicious replica in group A transmits its fault to a replica in group B. This is an instance of the fault-transmission problem on which we focus are energies; the general issue is that the fault in group A might spread unchecked, eventually resulting in an epidemic or a collusion-enabled coordinated attack. Figure 1(c) shows the detection of the malicious replica in group A; the more interesting part of this figure is that the correct replicas of group B are

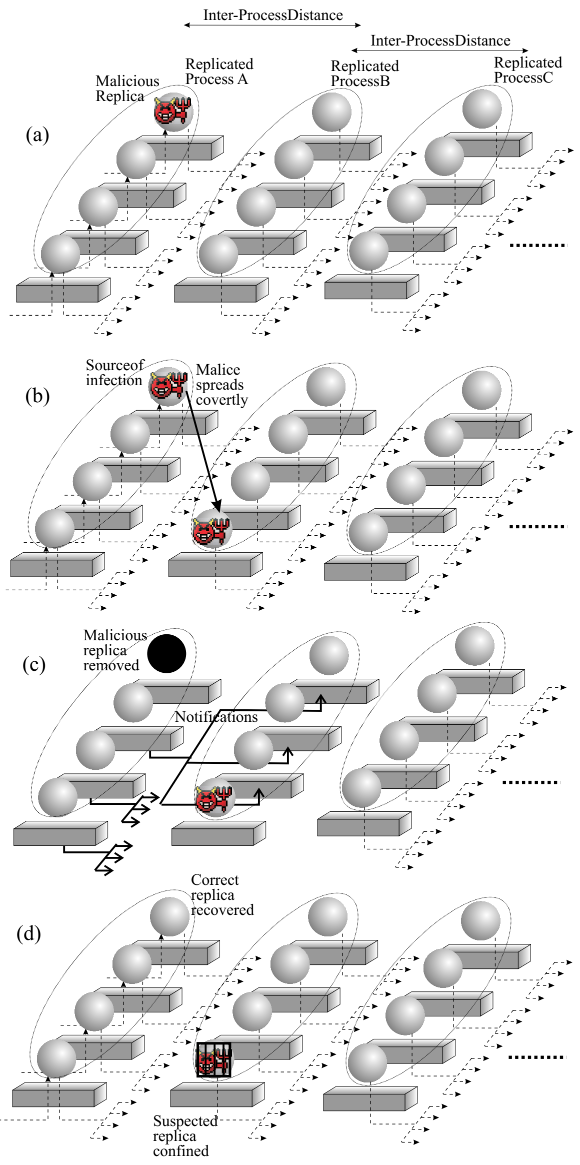

Figure 1: (a) Incubation period, (b) communicable period and fault transmission, (c) fault removal and proactive notification, (d) fault recovery and containment. Dashed lines denote secure reliable group communication.

[1] Some survivable systems relax the non-determinism requirement for state, but not for output values. This introduces the problem of being unable to distinguish between non-determinism and malice.

notified of the presence of a malicious member replica in their group. This notification is an intrinsic part of the proactive containment strategy. In Figure 1(d), fault-recovery is initiated, restoring group A to its Byzantine-fault tolerant strength, in terms of the number of replicas. Furthermore, the malicious replica in group B has been contained—in the sense that it has now fallen under suspicion; therefore, its range of activities might be restricted until it displays trustworthy behavior. Note that our proactive fault-containment strategy hinges on our ability to identify a malicious replica within a group.

Proactive containment seeks to minimize the communicable-period of the faulty process and to limit the outbreak radius. In the example just given, the communicable-period was reduced, via value-fault detection and recovery, and the outbreak radius was restricted (to an IPD of one), via proactive notification. Thus, the potential damage to the system, caused by the fault source in group A, has now been mitigated. Over time, the replica in group B might be detected to be malicious; it would then be removed from the system as well.

The key part of the proactive containment strategy is the determination that a replica in group B has been tainted by a replica in group A. This is possible only through the tracking of the possible avenues—secure or covert—of infection that stem from the malicious replica in group A. The most interesting cases in proactive containment occur if the fault-transmission takes place during the incubation-period for the malicious replica in group A, $i.e.$, even before this faulty replica begins to exhibit symptoms.

Note that malice can be transmitted via covert point-to-point channels between any two replicas either in the same group or in different groups. However, malice cannot be transmitted from one replica to another via the secure group communication system, because the infrastructural voting mechanisms support value-fault detection, which make it possible to detect an anomalous transmission from a malicious replica. To handle the case when the theoretical Byzantine-fault tolerance guarantees are violated (one-third or more of the replicas in the group are malicious) and existing voting mechanisms do not suffice, we will develop new ways of detecting malice.

3 Analysis

3.1 Modeling Fault-Transmission

Figure 2 shows fault-transmission starting from a single malicious replica, which acts as the fault source. Here, the malicious replica can communicate with three other processes—each located at an IPD of one away from the fault source—via distinct covert channels. These three processes are, in turn, connected to other processes—each located at an IPD of two away from the fault source. In the figure, we focus on three distinct paths of fault-transmission—Paths A, B and C—each radiating from the fault source, and then spanning a specific set of processes that may be tainted successively by fault-transmission.

The directed path from any process P_1 to a process P_2 is labeled with a number that represents P_2's *relative susceptibility* with regard to P_1, $i.e.$, the probability that the receiving process P_2 will be tainted, given that the transmitting process P_1 has been tainted. For example, the process on Path A at an IPD of one away from the fault source has a 45-percent chance of being tainted by the malicious replica; if this process becomes corrupted, then its downstream neighbor, at an IPD of two away from the fault source, has a 60-percent chance of being tainted.[2] Note that some of the paths (Paths B and C) contain firewalls that might partially filter any malicious traffic, that is, the relative suscep-

[2]These numbers are intended to be conceptually illustrative, and should not be construed as real data. We intend, however, to gather data in order to assign relative susceptibilities to various parts of the system.

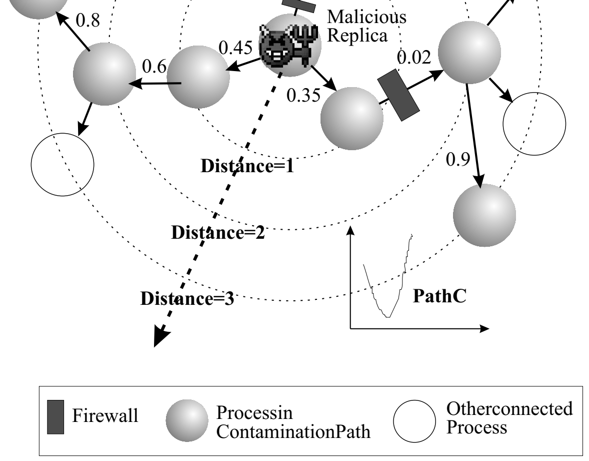

Figure 2: Three possible corruption paths, each with varying degrees of relative susceptibility.

tibility of a downstream process might be lowered by placing a firewall before upstream traffic reaches it. On the other hand, processes on Path A display increasing relative susceptibilities; this means that the risk of fault-transmission (and therefore, the penetration of the fault into the system) is higher, because each process, once tainted, does little to protect the downstream processes in its path.

The three paths are presented in order to show examples of three possible scenarios that we might expect to encounter in the real world. For instance, Path C might represent the route, from an Internet-enabled web client, to server processes inside an industrial corporation. On Path C, the process at IPD=1 might represent the company's front-end web-server, the process at IPD=2 might represent the company's internet-to-intranet gateway machine behind a firewall, while the processes at IPD=3 might represent the back-end processing servers within the company's intranet. Corrupting the web-server might be a difficult task (relative susceptibility = 35 percent), as such systems are typically designed with security in mind. Even if it were corrupted, the web-server would find it even harder (relative susceptibility = 2 percent) to corrupt the gateway machine, given that the gateway is located behind a firewall. However, if the gateway were breached, then, the corruption might become rampant within the intranet, given that the relative susceptibilities within the intranets of companies are usually high (about 90 percent, in the example shown). Thus, the last line of defense in this case is likely to cause the most damage, if corrupted. Path A models applications where security guarantees diminish with increasing distance from a highly protected core, given that the core has already become tainted. On the other hand, Path B represents an application with increasingly stringent lines of defense spanning the connected processes; this would be useful in modeling military applications where layer upon layer of security and access control are used to protect a critical resource, such as a target-engagement system.

Apart from allowing us to model the relative contamination of various paths in the system, the notion of relative susceptibility is useful in deciding if, and how much, recovery action ought to be taken. For one, it allows us to examine the vulnerable points in each possible path of fault-transmission. It also equips the survivable infrastructure with an estimate of the speed with which a fault might taint processes in its path. This form of advance warning, if propagated faster than the fault itself, can result in proactive fault-tolerance and fault-recovery without needless loss of resilience. Furthermore, the advance warning might also enable untainted downstream processes to deploy additional security mechanisms dynamically, $e.g.$, increase access control and filtering, in preparation for their possible infection.

3.2 Performance Issues

Proactive fault-containment mechanisms are not inexpensive. Enough information needs to be extracted from the system in order to enable untainted parts of the system to defend themselves from impending malice. Identifying this information in distributed systems is difficult, because it can not be found at a centralized location; therefore, it needs to be first aggregated from different sources, and then interpreted to yield something of value.

There are two parts to the performance cost of proactive fault-containment. One part arises primarily from the continuous book-keeping that needs to occur at various parts of the system, in terms of (i) recording incoming/outgoing invocations and responses at processes, (ii) discovering the different forms of communication that are usable by processes, and (iii) profiling the "normal" behavior of processes, in order to be able to detect any anomalous behaviors. The other contributor to the performance cost is the proactive notification/recovery; the period of reinforcement, at the untainted parts of the system, could result in increased resource usage and reduced performance.

Thus, indiscriminate use of the proactive mechanisms is wasteful of resources and of useful work that the system might instead perform. The relative susceptibility of the different processes in the system is a useful parameter in determining how quickly, and to what extent, the proactive fault-containment must happen. Thus, we need to determine the relative susceptibility, as well as the criticality, of every process in our system, so that we might contain the fault-transmission effectively, but only when needed.

In this context, it is worth mentioning that, without the advance notification of the possible tainting of the system, we would have to wait for existing Byzantine-fault detection mechanisms to detect malice. Unfortunately, Byzantine-fault-tolerant service-replication protocols are expensive both in terms of time and in terms of bandwidth. Thus, the more frequently the Byzantine-fault detector is exercised, the worse the performance. If malice spreads unchecked, without proactive notification, it is likely that the Byzantine-fault detector will be exercised frequently, thereby resulting in degraded performance.

Knowing the speed of the fault-transmission would enable us to estimate the outbreak radius, in order to focus the proactive recovery efforts. The communicability of the corruption can affect the speed of the fault-transmission; other factors include the number of paths radiating from the fault source, the number of processes connected (directly or indirectly) to the fault source, and the rate at which the fault source can send messages. It is also worth considering the severity of the consequences of ignoring the advance warnings. In some cases, particularly when the application is not performing mission-critical operations, ignoring the warnings for a period of time might be acceptable. Finally, it is possible that the corruption is transient, meaning that it could disappear on its own. Therefore, if we were certain of a fault's transience, proactive recovery might not be worth the cost.

4 Candidate Mechanisms

Enabling proactive recovery hinges on the capacity to detect faulty behavior, and to distinguish a malicious process from a correct process. Byzantine-fault detection alone is not sufficient for all situations or systems. A malicious process that is intelligent enough to corrupt other processes via covert channels, might well behave correctly when its outputs are voted on; in this case, the fault would escape Byzantine-fault detection.

Logging the use of any common resource that could potentially impact other processes, be it the network or the file system, is a possible and attractive approach. The logs could be scanned and analyzed for anomalous behavior [7]. Regardless, some malicious behavior might go unnoticed, and some correct behavior might be flagged wrongly as malicious. Handling the latter case in an automated system, without human intervention, is particularly challenging. Creating the logs would require all sensitive operations to be recorded. The logs would need to be stored in a way such 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 to make these logs tamper-resistant.

Instead of modifying methods to log their actions, we might adopt a sand-boxing approach: to restrict the use of certain functions. 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.

Understanding the characteristics of the class of corruption is dependent on being able to diagnose the corruption. Taking the correct proactive action would be assisted by having some sort of reference (of prior intrusions) to consult in each case. Other issues of concern with diagnosis include quantifying the strength of the accuracy of the diagnosis, and being able to detect when the corruption has been completely removed.

5 Related Work

BFT [1], SecureRing [2], and Rampart [3] are designed to tolerate Byzantine failures of up to f processors in a system containing $(3f + 1)$ replicas; our proposed system would likely build on a similar substrate. SecureRing and Rampart both use secure group membership protocols, which enable them to recognize and vote on failed or compromised processors. BFT does not require this feature for liveness; for our purposes this might mean that the client would need to be involved in more decisions concerning the detection of faulty processes. ITDOS [8] and Immune [4] are survivable systems that use BFT and SecureRing, respectively, to add survivability to CORBA.

The Hive [9] system partitions shared memory into cells. These cells are monitored for corruption. The goal is to ensure proactively that data are not read from a corrupted cell, as, if they were, they might then corrupt other cells that are accessed by the process reading the data. The proactive action to prevent corruption is conceptually similar to the proactive containment in our system. Hive, however, was built for multi-processing systems that are not necessarily distributed.

Staniford *et al.* [10] discuss and give examples of the *contagion* class of worms, which stealthily infect computers over a network. This spreading is a real-world example of the type of infection we are considering. They also suggest that it is likely that we will soon see, on public networks, worms that use cryptographic or hidden channels to communicate. Theoretically, this would allow for the worms to collude actively in the manner with which we are concerned.

6 Conclusion

We have introduced the fault-transmission problem in the context of survivable systems that are composed of actively-replicated Byzantine-fault tolerant processes. We have outlined techniques for controlling corruption, given its detection in some part of the system—even if the corruption is undetected in other parts of the system. We have discussed the performance trade-offs associated with proactive fault-containment. Finally, we have presented candidate mechanisms for collusion-avoidance and epidemic-avoidance in survivable systems.

References

[1] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the Third Sym-*

posium on Operating Systems Design and Implementation (OSDI '99), 1999.

[2] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The SecureRing protocols for securing group communication," in *Proceedings of the 31st Annual Hawaii International Conference on System Sciences (HICSS)*, vol. 3, pp. 317–326, IEEE Computer Society Press, 1998.

[3] M. K. Reiter, "The Rampart toolkit for building high-integrity services," in *Theory and Practice in Distributed Systems*, vol. 938, pp. 99–110, Springer-Verlag, Berlin Germany, 1995.

[4] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Providing support for survivable CORBA applications with the Immune system," in *International Conference on Distributed Computing Systems*, pp. 507–516, 1999.

[5] D. Malkhi and M. Reiter, "Unreliable intrusion detection in distributed computations," in *Proceedings of the 10th Computer Security Foundations Workshop (CSFW97)*, (Rockport, MA), pp. 116–124, 1997.

[6] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.

[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] D. Sames, B. Matt, B. Niebuhr, G. Tally, B. Whitmore, and D. Bakken, "Developing a heterogeneous intrusion tolerant CORBA system," in *Proceedings of the International Conference on Dependable Systems and Networks, 2002 (DSN '02)*, pp. 239–248, 2002.

[9] J. Chapin, M. Rosenblum, S. Devine, T. Lahiri, D. Teodosiu, and A. Gupta, "Hive: Fault containment for shared-memory multiprocessors," in *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 12–25, ACM Press, 1995.

[10] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002.