The FOREVER Service for Fault/Intrusion Removal

Paulo Sousa
Univ. de Lisboa
Portugal
pjsousa@di.fc.ul.pt

Alysson Neves Bessani
Univ. de Lisboa
Portugal
bessani@di.fc.ul.pt

Rafael R. Obelheiro
UDESC
Brazil
rro@joinville.udesc.br

ABSTRACT

This paper introduces FOREVER, a novel service that can be used to enhance the resilience of replicated systems, namely those exposed to malicious attacks. The main objective of FOREVER is to remove faults and intrusions that may happen during system execution, and such removal is done by combining both evolution and recovery techniques. The paper presents (i.) the challenges that systems exposed to malicious attacks need to address, and (ii.) how FOREVER can be used to tackle these challenges.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Distributed Systems**]: Distributed applications; K.6.2 [**Installation Management**]: Computing equipment management

General Terms

Security, Reliability

Keywords

Intrusion tolerance, rejuvenation, diversity

1. INTRODUCTION

Attackers are actively involved in the development of new techniques to inject or activate latent faults in existing systems [1], which implies that threats evolve during the lifetime of systems exposed to malicious attacks. This means that such systems need also to evolve in order that attacks never originate system failures. The ideal goal of such evolution would be the complete removal of vulnerabilities (thus eliminating any chances of an attack causing a failure), but it is well known that such goal is very difficult, if not impossible, to achieve. Nevertheless, during system execution, one can minimize the number of vulnerabilities by applying security patches to operating systems or by introducing newer (better) versions of application code.

The remaining vulnerabilities may be targeted by attacks and produce faults or intrusions. A resilient system needs to deal with such faults/intrusions, which may be masked through Byzantine fault-tolerant (BFT) replication protocols (e.g., [6, 18, 16]). These protocols are typically run on replicated systems and are able to tolerate the failure of a finite set of f replicas. The problem is that given a sufficient amount of time, a malicious adversary can find ways to compromise more than f replicas. Therefore, if one wants to build a resilient system that is continuously operating, then some sort of recovery mechanism will need to be added. The goal would be to detect and (reactively) recover compromised replicas at a pace faster than the time needed by an adversary to compromise more than f replicas. However, arbitrary faults are very difficult to detect [7, 11]. One alternative approach is to estimate the minimum time needed by an adversary to compromise more than f replicas and (proactively) trigger periodic replica recoveries at a faster pace [20, 30, 6]. Note that the time needed by an adversary to compromise more than f replicas is highly dependent on how different/diverse replicas are [19], and the use of diversity in security raises several concerns while a large amount of work remains to be done in order to get a workable solution [1].

The simplest recovery procedure is to boot a clean image containing the original operating system and application(s) code, and to obtain the current state (if any) from the surviving replicas. Clearly this technique removes the effects of any faults/intrusions that could have occurred before the recovery. However, the bugs/vulnerabilities that caused such faults/intrusions may remain or the adversary may have acquired knowledge before the recovery (e.g., user passwords, OS version) sufficient to deploy a more advanced attack after the recovery. Following this reasoning, the adversary may accumulate knowledge over time (days, weeks, months) until he is able to compromise more than f replicas between recoveries. This means that diversity in the space domain should be complemented with diversity in the time domain: *recoveries should introduce diversity.*

This paper presents the Fault/intrusiOn REmoVal through Evolution & Recovery (FOREVER) service, which addresses some of the challenges identified above. This service can be used to enhance the resilience of replicated systems, namely those that can be affected by malicious attacks. As the name implies, the main objective of FOREVER is to remove faults and intrusions by combining both evolution and recovery techniques.

2. CHALLENGES

In this section we analyze the main challenges that any system exposed to malicious attacks needs to address.

2.1 Threats Evolve During System Lifetime

If a system needs to be deployed in an environment (e.g., Internet) where malicious attackers may be present, then it is certainly not safe to assume that threats will be always the same. In fact, it is well-known that attackers are actively involved in the development of new techniques to inject or activate latent faults in existing systems [1].

The usual way of fighting such malicious behavior is by applying security patches to operating systems or by introducing newer (better) versions of the application code. Typically, these activities are done manually by a system administrator and may introduce unavailability periods in system operation.

We argue that money-critical (e.g., online banking, e-commerce websites) and safety-critical (e.g., critical infrastructures) systems should not depend on human intervention and that unavailability should be avoided at all costs. In fact, our main concern is with safety-critical systems, given the recent shift in attackers attention from classical money-critical targets, such as e-commerce websites, to safety-critical infrastructures (e.g., power, water, gas) [5, 17].

2.2 BFT Protocols are Not Enough

Byzantine fault-tolerant protocols may be used in replicated systems to deal with the arbitrary failure of a finite set of f replicas [6, 18, 16]. Such protocols have limited utility in long-lived systems where malicious adversaries are constantly deploying attacks and causing intrusions. In fact, fault/intrusion tolerant protocols are useful to delay the corruption of the overall replicated system by a certain interval of time that depends on the actual value of f and on how different/diverse the individual replicas are [19]. Unfortunately, while a higher f allows to tolerate more compromised replicas, it also means that there are more individual replicas that need to be different from each other. For instance, most Byzantine fault-tolerant consensus algorithms (a fundamental building block for replication protocols) for practical environments require a minimum of $3f + 1$ processes to tolerate faults on at most f of these. This means that when f is increased from 1 to 2, the minimum total number of replicas is increased from 4 to 7. Notice that if these three extra replicas are not different from the existing ones, we are in fact increasing the probability of the overall system becoming compromised.

We argue that fault/intrusion tolerant protocols are useful as long as they are complemented with *recovery mechanisms* able to reduce both the value of f and the time window for an adversary to compromise more than f replicas.

2.3 Recoveries may be Attacked

As explained in the previous section, recovery mechanisms may play an important role in ensuring that no more than a certain number of replicas is compromised at the same time. Some existing works implement proactive recovery making the system tolerate an unbounded number of malicious faults during its lifetime [6, 30].

If a system has these mechanisms deployed, they will be the natural first target of any attack: a malicious adversary will start by stopping, delaying, or even corrupting the recovery process, before deploying the actual attack that compromises the replicated system. A recent work shows how a set of existing systems may collapse precisely because the recovery mechanisms are not adequately protected [26]. This work analyzes systems that are designed under the asynchronous model, but end-up making a few synchrony assumptions, namely related to recoveries triggering and execution time. Such assumptions may have acceptable coverage in the presence of accidental (non-intentional) faults, but represent a vulnerability in a malicious environment.

We argue that recovery mechanisms must be architected in a way that their timeliness and effectiveness cannot be affected, namely by a malicious adversary.

2.4 Recoveries Affect Availability

When a replica is recovering, there exists normally a non-negligible interval of time during which messages (from clients or other replicas) are lost, and thus a recovering replica behaves as being crashed during that time. The unavailability of individual replicas may provoke the unavailability of the overall replicated system. Consider for instance a Byzantine-fault-tolerant replicated system with 4 replicas, and thus able to tolerate one faulty replica (see Section 2.2). If this system is enhanced with recovery mechanisms, then its availability will be affected if (i.) more than one replica recovers at the same time, or (ii.) one replica is recovering while another one is compromised. In both cases, the sum of arbitrary faulty replicas and purposely crashed (recovering) replicas is greater than f, and thus the replicated system will become unavailable during those periods.

This means that recovery mechanisms cannot be added in a straightforward manner to existing Byzantine-fault-tolerant replicated systems. In order to avoid unavailability, recoveries need to be coordinated in an agreed schedule, and the system needs a higher total number of replicas to tolerate both Byzantine (arbitrary) faults and recovery-induced (crash) faults.

2.5 Recoveries Should Introduce Diversity

If a recovery procedure restores a replica to a previous known-to-be-good state (e.g., by rebooting the system from clean media), this replica is still vulnerable to whatever caused its compromise in the first place. Therefore, it is not enough to simply restore replicas to known states, since this would allow an attacker to exploit the same vulnerabilities as before. To address this issue, we argue that the recovery procedure should itself introduce some degree of diversity to restored replicas, so that attackers will have to find other vulnerabilities in order to compromise a replica.

3. THE FOREVER SERVICE

In this section we explain how FOREVER tackles the challenges presented in Section 2.

3.1 Recovery and Evolution

We envisage FOREVER as a service that can be used to enhance the resilience of intrusion-tolerant replicated systems by allowing these systems to tolerate an arbitrary number of faults. Such an ambitious goal is achieved through the combination of two important and complementary characteristics:

Recovery. FOREVER allows an intrusion-tolerant system to recover from past malicious actions/faults, by cleaning the effects of such actions through (i.) time-triggered periodic recoveries, and (ii.) event-triggered reactive recoveries when malicious behavior is detected. When more than f replicas *detect* that a given replica R_i is inequivocally faulty (e.g., R_i sends contradictory signed messages in a distributed protocol), this replica is immediately recovered (this is an event-triggered recovery). If, on the other hand, the replicas only *suspect* that R_i is faulty (but they cannot be certain), recovery of R_i is scheduled in such a way that overall system availability is not affected. Also, time-triggered recoveries are performed periodically in order to neutralize the effects of any undetected faults and intrusions.

Evolution. When FOREVER triggers a recovery of a certain replica, it modifies the vulnerabilities that may be exploited by a malicious adversary (e.g., OS access passwords, open ports, authentication methods). Moreover, FOREVER coordinates online upgrades (e.g., installation of security patches, installation of a new version of the intrusion-tolerant application being protected) with a minimum impact on availability.

These characteristics make FOREVER different from any other recovery service proposed in the past [6, 30, 25].

3.2 Hybrid Model and Architecture

In order to avoid that FOREVER itself is not victim of malicious attacks, an intrusion-tolerant system enhanced with FOREVER should be built under a hybrid system model and architecture [28] in which the system is composed of *two parts*, with distinct properties and assumptions. These two parts are typically called *payload* and *wormhole*. The intrusion-tolerant application (and replication library) runs in the payload part exposed to arbitrary faults and asynchrony. The FOREVER service runs in the wormhole part that is guaranteed to be secure and timely by construction. Figure 1 depicts an intrusion-tolerant application and FOREVER deployed under a hybrid architecture.

In more practical terms, FOREVER should be deployed as a secure real-time subsystem separated from the intrusion-tolerant application being protected. The separation can be achieved through either hardware (e.g., a different PC or a PC appliance board [15, 27]) or software (e.g., using virtualization [2]). Moreover, as it will be explained in Section 3.3, the local FOREVER subsystems running in each replica need to be connected by a synchronous and secure control network.

Although such an hybrid architecture is sufficient to guarantee the security of the FOREVER service, it is not sufficient to guarantee the security and timeliness of the entire recovery process. In fact, FOREVER needs to perform actions over the intrusion-tolerant application, namely it needs to shutdown, recover, and reboot the operating system running in the payload. This means that none of these actions can be allowed to be affected by a malicious attacker. In Section 4 we explain how virtualization can be used to achieve this goal.

3.3 Coordination

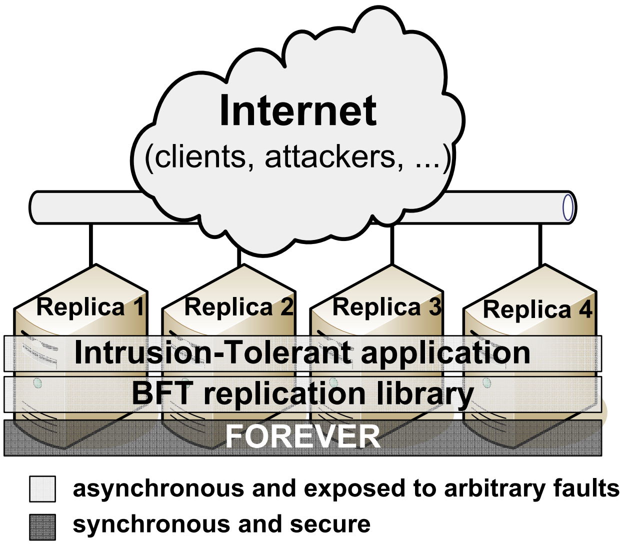
Internet
(clients, attackers, ...)
Replica 1
Replica 2
Replica 3
Replica 4
Intrusion-Tolerant application
BFT replication library
FOREVER
asynchronous and exposed to arbitrary faults
synchronous and secure

Figure 1: FOREVER under a hybrid architecture.

In order to guarantee availability, FOREVER needs to coordinate the recoveries of different replicas. This coordination requires three services:

- **Perfect Failure Detection:** This service is employed to detect if application replicas are alive. In fact each FOREVER monitor monitors each other and, when one is detected to be faulty, the whole replica is considered to be faulty[1].
- **Clock Synchronization**: All local wormholes have synchronized clocks to ensure that the recovery time schedule for each replica is precise.
- **Total Order Multicast**: The algorithm employed to schedule reactive recoveries due to detections [25] uses total order multicast to update the same recovery schedule on every FOREVER monitor without affecting the availability of the intrusion-tolerant application.

Notice that the implementation of these services requires strong timing assumptions, so the FOREVER wormhole subsystem needs to be synchronous.

3.4 Introducing Diversity

One of the most innovative features of FOREVER is the automatic introduction of diversity after a recovery. The idea is that a recovered system should be different from its previous incarnation. The easiest way to do this is to have a pool of operating system images and to start a different one each time the system is recovered. Contrary to common belief, some recent studies reported that different COTS software does not have the same vulnerabilities [10], at least for database management systems. We believe this kind of independence also holds for operating systems.

[1]Since the FOREVER subsystem is faulty on this replica, it cannot be recovered without an administrator manual intervention.

Besides the use of different operating systems and different versions of the same operating system, we intend to use two other features to generate diversity on recovered replicas:

- **Address Obfuscation**: Attacks which exploit memory programming errors (e.g., buffer overflows) are one of today's most serious security threats. These attacks require an attacker to have an in-depth understanding of the internal details of the system being attacked, including the locations of critical data and/or code. Address obfuscation techniques randomize the location of program data and code each time a program is executed. It has been shown that address obfuscation can greatly reduce the probability of successful attacks [4, 3, 8, 22, 29, 23]. Given that recoveries force the restart of every program in the system, they are a perfect way of introducing address obfuscation both at the OS and application level.
- **System Transformations**: A system transformation $\tau : I, F, P \rightarrow I', U$ is a function that takes a file system image I and generates a new file system image I' updating a set of features F according to a set of parameters P. The idea of these transformations is to generate new images changing critical parameters such as access ports, disk layouts, usernames, passwords, and permissions; the set of updates performed by the transformation (e.g., the new passwords) is stored in U, to ensure that the new image can be accessed by its authorized users.

We are still beginning to investigate the use of diversity on the FOREVER architecture. A fundamental aspect that needs further investigation is how to generate sufficient diversity while maintaining the compatibility with the intrusion-tolerant application.

4. PROTOTYPE

We have already developed a prototype in which the different parts of the system (intrusion-tolerant application and FOREVER) are separated through the use of virtualization.

Virtual machine (VM) kernels, such as the Xen virtual machine monitor [2], may host multiple guest operating systems, every one executed within an isolated VM. In the case of Xen, VMs are called *domains*. Xen domains do not have all the same privileges, namely there is a special domain with more privileges, called *dom0*, which is created automatically when the physical machine boots. Domain *dom0* builds other domains (called *dom1*, *dom2*, etc). It also performs administrative tasks such as suspending and resuming other VMs, and it can be configured to execute with higher priority than the remaining VMs. These characteristics make Xen an appropriate deployment environment for an intrusion-tolerant application enhanced with FOREVER: the latter would be deployed in domain *dom0*, while the former would run in a non-privileged domain.

Figure 2 depicts how our current prototype is deployed in each replica. The FOREVER local monitor has access to a file-system repository (FSR) that contains a set of different file-system images (FSI) composed of different operating systems configured with different passwords. Each recovery boots a different FSI and consequently the set of vulnerabilities is changed between recoveries.

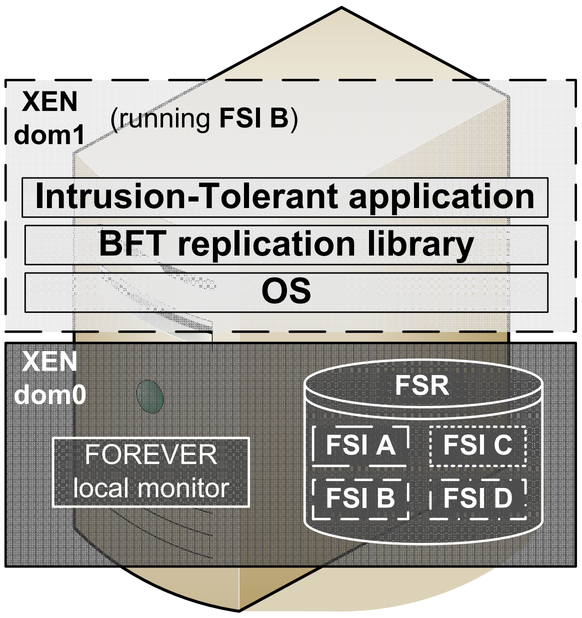

Figure 2: FOREVER prototype deployed in each replica.

Another benefit of using virtual machines is that they allow the recovery process to be more efficient [24]. In our context, FOREVER may instantiate a new payload image (i.e., boot a new virtual replica) before shutting down the old image. If the intrusion-tolerant application is stateless, the transition from the old image to the new recovered image can be almost instantaneous.

5. RELATED WORK

FOREVER is an extension of previous work on proactive and reactive recovery [25]. It intends to be a generic recovery and evolution service supporting not only proactive and reactive recoveries but also application and OS online upgrades.

Rejuvenation [13, 9] has been proposed in the 90s as a proactive technique to deal with transient failures due to software aging. The idea is to periodically rejuvenate some software components to eliminate and prevent failures. Some works on this field, beginning with [9], developed techniques to choose the optimal rejuvenation period in order to minimize the downtime of the system. None of the works on software rejuvenation assume faults caused by malicious adversaries, so the problem that we deal in this work is much more complicated than software aging.

Several works advocate the use of proactive recovery to make the system tolerate an unbounded number of malicious faults during its lifetime as long as no more than f faults occur during a bounded time period [6, 30]. Recently, some of the authors showed that all these works have some hidden problems that could be exploited by a smart adversary [26]. The main problem is that these works assume the asynchronous distributed system model and, under this model it is not possible to guarantee that recoveries are triggered and executed within known time bounds: an adversary can delay the execution of the recoveries and be able to corrupt more than f nodes of a system. To make this problem worse, it is impossible to detect such timing attacks under the asynchronous model. The architecture presented in the present paper is based on a hybrid distributed system model [28] and thus uses some trusted and timely components to ensure that replicas are always rejuvenated in accordance to the predefined time bounds.

There is a large body of research that aims to reactively recover systems as fast and efficiently as possible (e.g., [12, 21, 14]). Early work on this field (e.g., [12]) advocates the execution of some kind of recovery action (in general, restart the monitored process) after a fault is detected. More recently, the recovery oriented computing project proposed a set of methods to make recovery actions more efficient and less costly in terms of time [21]. Several techniques were developed in this project, either to detect failures, restart the minimum set of system components to put the system back to correct operation and to undo some operator configuration errors. These works do not consider Byzantine faults or security-compromised components and also do not rely on redundancy to ensure that the system stays available during recoveries, the main problems addressed by FOREVER.

6. CONCLUSIONS AND FUTURE WORK

Intrusion-tolerant protocols are sufficient to ensure the resilience of replicated systems as long as no more than a certain finite number of replicas is compromised. Unfortunately, this guarantee has limited utility in long-lived systems exposed to malicious attacks. This paper introduced FOREVER, a novel service that can be used to enhance the resilience of intrusion-tolerant replicated systems operating in malicious environments. FOREVER removes faults and intrusions that may happen during system execution, and such removal is done by combining both evolution and recovery techniques.

As future work, we intend to develop novel heuristics to generate diversity after recoveries. Moreover, we intend to assess the benefits and the tradeoff of enhancing a replicated system exposed to malicious attacks with FOREVER. Our ultimate goal is to develop a dependability case of FOREVER and of a system enhanced with this service [1].

7. ACKNOWLEDGMENTS

This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL) and NoE IST-4-026764-NOE (RESIST/FOREVER), and by the FCT, through the Multiannual and the CMU-Portugal Programmes.

8. REFERENCES

[1] M. Banatre, A. Pataricza, A. Moorsel, P. Palanque, and L. Strigini. From resilience-building to resilience-scaling technologies: Directions – ReSIST NoE Deliverable D13. DI/FCUL TR 07–28, Dep. of Informatics, Univ. of Lisbon, Nov. 2007.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebaurer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP'03)*, pages 164–177, Oct. 2003.

[3] S. Bhatkar, D. C. DuVarney, and R. Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *Proc. of the 12th USENIX Security Symp.*, pages 105–120, Aug. 2003.

[4] S. Bhatkar, R. Sekar, and D. C. DuVarney. Efficient techniques for comprehensive protection from memory error exploits. In *Proc. of the 14th USENIX Security Symp.*, pages 271–286, Aug. 2005.

[5] E. Byres and J. Lowe. The myths and facts behind cyber security risks for industrial control systems. In *Proc. of VDE Kongress*, 2004.

[6] M. Castro and B. Liskov. Practical Byzantine fault-tolerance and proactive recovery. *ACM TOCS*, 20(4):398–461, 2002.

[7] A. Doudou, B. Garbinato, R. Guerraoui, and A. Schiper. Muteness failure detectors: Specification and implementation. In *Proc. of the 3rd European Dependable Computing Conf.*, pages 71–87, Sept. 1999.

[8] S. Forrest, A. Somayaji, and D. H. Ackley. Building diverse computer systems. In *Proc. of the 6th Workshop on Hot Topics in Operating Systems*, pages 67–72, May 1997.

[9] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of software rejuvenation using Markov regenerative stochastic Petri nets. In *Proc. of Int. Symp. on Software Reliability Engineering (ISSRE'95)*, Oct. 1995.

[10] I. Gashi, P. Popov, and L. Strigini. Fault tolerance via diversity for off-the-shelf products: A study with SQL database servers. *IEEE Transactions on Dependable and Secure Computing*, 4(4):280–294, Oct-Dec 2007.

[11] A. Haeberlen, P. Kouznetsov, and P. Druschel. The case for Byzantine fault detection. In *Proc. of the 2nd Workshop on Hot Topics in System Dependability*, 2006.

[12] Y. Huang and C. M. R. Kintala. Software implemented fault tolerance: Technologies and experience. In *Proc. of 23rd Int. Symp. on Fault Tolerant Computing (FTCS-23)*, pages 2–9, June 1993.

[13] Y. Huang, C. M. R. Kintala, N. Kolettis, and N. D. Fulton. Software rejuvenation: Analysis, module and applications. In *Proc. of 25th Int. Symp. on Fault Tolerant Computing (FTCS-25)*, pages 381–390, June 1995.

[14] K. R. Joshi, M. Hiltunen, W. H. Sanders, and R. Schlichting. Automatic model-driven recovery in distributed systems. In *Proc. of the 24th IEEE Symp. on Reliable Distributed Systems (SRDS 2005)*, pages 26–38, Oct. 2005.

[15] S. Kent. *Protecting Externally Supplied Software in Small Computers.* PhD thesis, Laboratory of Computer Science, Massachusetts Institute of Technology, 1980.

[16] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *Proc. of the 21st ACM Symp. on Operating Systems Principles (SOSP'07)*, Oct. 2007.

[17] D. Maynor and R. Graham. Scada security and terrorism: We're not crying wolf! In *Black Hat*, Jan. 2006.

[18] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo. Randomized intrusion-tolerant asynchronous services. In *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN'06)*, pages 568–577, jun 2006.

[19] R. R. Obelheiro, A. N. Bessani, L. C. Lung, and M. Correia. How practical are intrusion-tolerant distributed systems? DI-FCUL TR 06–15, Dep. of Informatics, Univ. of Lisbon, Sept. 2006.

[20] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (ext. abstract). In *Proc. of the 10th ACM Symp. on Principles of Distributed Computing (PODC'91)*, pages 51–59, 1991.

[21] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery oriented computing (ROC): Motivation, definition, techniques and case studies. UCB/CSD TR 02–1175, Computer Science Dep., Univ. of California at Berkeley, Mar. 2002.

[22] PaX. http://pax.grsecurity.net/, 2001.

[23] R. Pucella and F. B. Schneider. Independence from obfuscation: A semantic framework for diversity. In *Proc. of the 19th IEEE Workshop on Computer Security Foundations*, pages 230–241, 2006.

[24] H. P. Reiser and R. Kapitza. Hypervisor-based efficient proactive recovery. In *Proc. of the 26th IEEE Symp. on Reliable Distributed Systems (SRDS'07)*, pages 83–92, Oct. 2007.

[25] P. Sousa, A. N. Bessani, M. Correia, N. F. Neves, and P. Verissimo. Resilient intrusion tolerance through proactive and reactive recovery. In *Proc. of the 13th IEEE Pacific Rim Int. Symp. on Dependable Computing (PRDC'07)*, pages 373–380, Dec. 2007.

[26] P. Sousa, N. F. Neves, and P. Verissimo. Hidden problems of asynchronous proactive recovery. In *Proc. of the Workshop on Hot Topics in System Dependability (HotDep'07)*, June 2007.

[27] Trusted Computing Group. TCG Specification Architecture Overview, revision 1.2. https://www.trustedcomputinggroup.org/groups/tpm/, 2004.

[28] P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1), 2006.

[29] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Transparent runtime randomization for security. In *Proc. of the 22nd Int. Symp. on Reliable Distributed Systems (SRDS)*, pages 260–269, Oct. 2003.

[30] L. Zhou, F. Schneider, and R. Van Rennesse. COCA: A secure distributed online certification authority. *ACM TOCS*, 20(4):329–368, Nov. 2002.