Hidden Problems of Asynchronous Proactive Recovery

Paulo Sousa
pjsousa@di.fc.ul.pt
Univ. of Lisboa[*]

Nuno Ferreira Neves
nuno@di.fc.ul.pt
Univ. of Lisboa

Paulo Verissimo
pjv@di.fc.ul.pt
Univ. of Lisboa

Abstract

A node-exhaustion-safe intrusion-tolerant distributed system is a system that assuredly does not suffer more than the assumed number of node failures. In a recent work, we showed that it is not possible to build any *type of node-exhaustion-safe distributed f intrusion-tolerant system under the asynchronous model. Some years ago, an intuition about this problem motivated the research around proactive recovery, which made possible the appearance of asynchronous systems that allegedly can tolerate any number of faults over the lifetime of the system. In this paper, each of these works is analyzed in detail and is explained why they fail to achieve their goal. Afterwards, we summarize the four problems that may affect systems employing proactive recovery.*

1 Introduction

In order for fault or intrusion tolerant systems to be useful, they should be guaranteed not to exceed the usually predefined f quorum of faults throughout their mission. Let us think about node failures in fault-tolerant distributed systems. Depending on the system, nodes may fail by crash, or start behaving in a Byzantine way, or disclose some secret information, and all these types of failures may be caused by accidents (e.g., a system bug), or may be provoked by malicious actions (e.g., an intrusion carried out by a hacker). From now on, we talk about intrusion-tolerant systems in the sense of subsuming all kinds of failures, accidental or malicious.

The system should be guaranteed not to suffer more than f node failures. We called this predicate *exhaustion-safety*, and showed that achieving it is a hard problem, given the inexorable production of faults during the system's lifetime. The problem becomes much harder in the presence of malicious faults (attacks, intrusions). An intuition about this problem motivated some groundbreaking research around proactive recovery, which made possible the appearance of asynchronous protocols and systems that can tolerate any number of faults over the lifetime of the system, provided that fewer than a subset of the nodes become faulty within a supposedly bounded small window of vulnerability. This is achieved through protocols that periodically rejuvenate the system.

However, we showed in theory that it is not possible to build *any* type of exhaustion-safe distributed f intrusion-tolerant system under the asynchronous model [8, 7]. Moreover, we proved the need for the proactive recovery mechanisms to be synchronous. In the course of our findings, we analyzed several significant works in asynchronous proactive recovery [2, 11, 1, 5], to confirm whether or not they fell under this impossibility result, and if so, in what conditions.

This paper concisely reports what we learned. These systems, which are otherwise correct according to their stated assumptions under the asynchronous model used, end-up making a few synchrony assumptions (some of which implicit) that in normal (accidental-fault) cases would have acceptable coverage. However, in the malicious fault scenario, which they all address, these assumptions will certainly be attacked and hence cannot be reasonably taken for granted.

The explanation of why such well-designed and relevant systems might come to have problems is intuitively the following: these compromises of synchrony or timing assumptions are in fact nothing else than *unexpected timing faults*. However, an asynchronous model is by nature blind to timing faults so it is natural that these systems will find it hard dealing with the latter, let alone recognize (detect) them, as we found out. Note that some assumptions may be implicit and so their violation, harmful as it may be, does not even entail a violation of the system's explicit assumptions. In this case, the system is completely oblivious to (and thus unprotected from) this kind of attacks.

This analysis allowed us to categorize the four generic concrete problems that may affect any asynchronous system employing proactive recovery. Recall that the root of these problems is the need for the recovery mechanisms to be synchronous. In other works [9, 7] we have shown that by resorting to hybrid distributed systems models (a.k.a. Wormhole models) these problems can be fixed in an elegant way, by preserving the asynchrony of most of the system, but encapsulating the necessary timing assumptions in a safe manner. In fact, the excellent algorithms and mechanisms of the works surveyed can all be provided in an exhaustion-safe manner should they eventually be re-designed under a hybrid model.

[*] This work was partially supported by the EC through project IST-2004-27513 (CRUTIAL) and NoE IST-4-026764-NOE (RESIST), and by the FCT through project POSI/EIA/60334/2004 (RITAS) and the Large-Scale Informatic Systems Laboratory (LaSIGE).

2 Hidden Problems

2.1 CODEX

CODEX (COrnell Data EXchange) is a recent distributed service for storage and dissemination of secrets [5]. It binds secrets to unique names and allows subsequent access to these secrets by authorized clients.

The service makes relatively weak assumptions about the environment and the adversaries. It assumes an asynchronous model where operations and messages can suffer unbounded delays. Moreover, messages while in transit may be modified, deleted or disclosed. Nevertheless, it is assumed fair links, which means that if a message is transmitted sufficiently often, then it will eventually be delivered.

CODEX enforces three security properties. Availability is provided by replicating the values in a set of n servers. It is assumed that at most f servers can (maliciously) fail at the same time, and that $n \geq 3f+1$. Cryptographic operations such as digital signatures and encryption/decryption are employed to achieve confidentiality and integrity of the communication. These operations are based on public key and threshold cryptography. The private key of CODEX is shared by the n CODEX servers using an $(n, f+1)$ secret sharing scheme[1], which means that no server is trusted with that private key. Therefore, even if an adversary controls a subset of f or less replicas, he will be unable to sign as CODEX or to decrypt data encrypted with the CODEX public key.

An adversary must know at least $f+1$ shares in order to construct the CODEX private key and, knowing it, violate the confidentiality property. CODEX assumes that a maximum of f nodes running CODEX servers are compromised at any time, with $f = \frac{n-1}{3}$. This assumption excludes the possibility of an adversary controlling $f+1$ servers, but as the CODEX paper points out, "it does not rule out the adversary compromising one server and learning the CODEX private key share stored there, being evicted, compromising another, and ultimately learning $f+1$ shares". To defend against these so called *mobile virus attacks* [6], CODEX employs the APSS proactive secret sharing protocol [12], where some implicit time-related assumptions appear: "This protocol is *periodically executed*, each time generating a new sharing of the private key but without ever materializing the private key at any server". Because older secret shares cannot be combined with new shares, the CODEX paper concludes that "a mobile virus attack would succeed only if it is completed in the *interval between successive executions* of APSS". This scenario can be prevented from occurring by running APSS regularly, in intervals that *"can be as short as a few minutes"*.

Note that by exploring the asynchrony of APSS, one can deploy an attack to increase its execution interval without it being noticed. However, lengthening the window's real time duration would collide with the implicit assumptions made above, and would allow the retrieval of $f+1$ shares and the disclosure of the CODEX private key. Once this key is obtained, it is trivial to breach the confidentiality of the service.

Briefly, attacks slowing down some nodes and postponing the delivery of messages between two parts of the system would exhibit a behavior that could have occurred in any fault-free asynchronous system, not violating any assumption. Secondly, an attack lowering, one after the other, the rate of each local clock, would resemble a mobile virus attack. Both (adjustment and mobile attack) are allowed by the system's assumptions, where clocks are not obliged to have a bounded drift rate.

So, in a nutshell, combinations of slowing down the clock of at least $f+1$ up to all of the n nodes, and of slowing down or temporarily cutting-off the links between these nodes and the rest of the system, might grant the attacker a situation where the system would be very slow and/or oblivious to recovery-triggering messages. Slow enough to allow it to deploy a final mobile virus attack, allowed in the CODEX paper, learning, one by one, $f+1$ CODEX private key shares.

2.2 COCA

COCA (Cornell Online Certification Authority) [11] is the predecessor of CODEX. COCA makes the same type of assumptions, namely it builds on the asynchronous model and uses APSS. However, there are two main differences between COCA and CODEX: (i.) COCA assumes a bound on the maximum number of nodes (f) that can be compromised during each window of vulnerability (defined as the interval between two consecutive rejuvenations); and (ii.) in COCA, not only the private key shares are refreshed through APSS, but also the server code and state are periodically rejuvenated.

Despite these two differences, attacks such as described in the previous section may also affect COCA. Certainly, if the operating system is not rejuvenated (the paper is not clear about the definition of server code). However, let us consider that clocks are rejuvenated as part of the recovery process.

COCA assumes that "at most t of the n COCA servers are ever compromised *during each window of vulnerability*, where $3t+1 \leq n$ holds". But the COCA definition of *"compromised server"* is: 1) stop executing; or 2) deviate arbitrarily from its specified protocols (i.e., Byzantine failure); or 3) disclose information stored locally.

Note that there is an implicit assumption that local clocks have bounded drift rate, but in fact according to COCA's stated system model, clocks cannot be anything else than increasing counters. In consequence, a mobile clock-rate changing attack leaves the clocks correct, i.e., in accordance with COCA's assumptions, and as such *does not* count for the number of compromised servers. To make it clearer, "very slow" servers would still be correct according to COCA's model, be it because they are too loaded or because their clocks tick slowly. So, such an attack could legally slow

[1] In a $(n, f+1)$ secrete sharing scheme, there are n shares and any subset of size $f+1$ of these shares is sufficient to recover the secret. However, nothing is learnt about the secret if the subset is smaller than $f+1$.

down the clocks of $f+1$ servers during a window of vulnerability, i.e., between consecutive rejuvenations, realizing the conditions for the final attack described earlier for CODEX.

2.3 BFT and BFT-PR

In [2], it is proposed the first Byzantine-fault-tolerant (BFT) state machine replication algorithm not relying on any synchrony assumption to provide safety. Actually, BFT is proposed in two flavors. Firstly, authors present non-proactive BFT, which works under the asynchronous model and assumes that the number f of faulty replicas is bounded by $\lfloor \frac{n-1}{3} \rfloor$ during the entire lifetime of the system. Then, authors propose BFT with proactive recovery (BFT-PR), which can tolerate any number of faults provided fewer than 1/3 of the replicas become faulty within a supposedly small window of vulnerability. However, contrarily to COCA and CODEX, the BFT proactive recovery mechanism makes extra assumptions: secure cryptography, read-only memory and watchdog timers. Secure cryptography means that a replica can sign and decrypt messages without exposing its private key. Read-only memory is used both to store the public keys for other replicas and to store the code of the recovery monitor that executes the (proactive) recovery procedure.

Watchdog timers are used to periodically interrupt processing and hand control to the recovery monitor, and prefigure a timing assumption. It is also assumed that there is some unknown point in the execution after which *"either all messages are delivered within some constant time* Δ *or all non-faulty clients have received replies to their requests"*. If these assumptions are satisfied, then BFT-PR works correctly. Namely, authors point out that an appropriate choice of Δ allows recoveries at a fixed rate. This suggests that the length T_v of the window of vulnerability can have a *known bounded value* in normal conditions.

Unlike COCA and CODEX, BFT-PR explicitly makes an assumption about the need for a more synchronous component – the watchdog timer – that is able to guarantee the timely triggering of proactive recovery protocols. This is a good measure on the one hand, with the coverage of a hardware component, which makes BFT-PR immune to attacks such as described in Section 2.1. On the other hand, we argue that these timing assumptions alone may not be sufficient to guarantee the correct execution of the proactive recovery mechanism, in a malicious environment.

The watchdog timer guarantees a timely periodic interrupt and can therefore be used to timely *trigger* the periodic activity of the recovery monitor responsible for the recovery procedures. Note that there is an implicit assumption that once started, the recovery procedure finishes within a known time bound. However, the watchdog timer is a mere 'click': it only guarantees that the recovery monitor is timely started. The recovery monitor, although immune to modification faults, is executed as a task of the asynchronous system. So, in theory, the recovery monitor, even if timely started, may take an unknown and very long interval to finish its execution. In practice, in a malicious environment, it will be slowed down, if that fits the purpose of the attacker. For instance, it is straightforward for a malicious adversary to increase the delivery time of recovery network messages.

Other attacks may be made before recovery, attempting at postponing it, along the lines discussed in the previous sections. BFT-PR authors admit the impossibility of ensuring a bound on T_v, but they only consider the scenario of a DoS attack and assume that one is detectable. We call this kind of attacks *conspicuous time attacks* [10] and indeed they are easier to detect and alert an administrator, as the authors propose. However, an adversary able to compromise the clock rate (which again, would not invalidate the asynchronous assumptions of BFT-PR) may simultaneously increase the value of T_v and prevent detection, performing a *stealth time attack* [10].

Note that another implicit assumption is made that the interface between the asynchronous Byzantine part and the synchronous crash-failure watchdog is tamperproof. But in reality there is nothing in the design that enforces that. An attack breaking this interface might for example reprogram the watchdog, preventing it from triggering or delaying it.

So, in summary, albeit periodically triggered, the BFT-PR proactive recovery mechanism may have an unbounded execution time and a stealth adversary may render impossible the detection of this anomalous behavior from within BFT nodes.

2.4 Asynchronous Proactive Cryptosystems

The process of building a fault-tolerant cryptosystem typically includes the use of threshold cryptography [3]. The idea is that a cryptographic operation is performed by a group of n servers, such that an adversary who corrupts up to f ($f<n$) servers and observes their secret key shares can neither break the cryptosystem nor prevent the system as a whole from correctly performing the operation.

However, when a threshold cryptosystem operates over a longer time period, it may not be realistic (or safe) to assume that an adversary corrupts only f servers during the entire lifetime of the system. Proactive cryptosystems address this problem by operating in phases; they can tolerate the corruption of up to f different servers during every phase [4]. They rely on the assumption that servers may have a special reboot procedure that erases data and removes the adversary from a corrupted server. The idea is to proactively reboot all servers at the beginning of every phase, and to subsequently refresh the secret key shares such that in any phase, knowledge of shares from previous phases does not give the adversary any type of advantage. Therefore, proactive cryptosystems tolerate a mobile adversary [6], which may move from server to server and eventually corrupt every server in the system.

In [1], the neat idea of asynchronous proactive cryptosystems is introduced, i.e., proactive cryptosystems in asynchronous networks. The formal model proposed for asynchronous proactive networks extends an asynchronous network by an *abstract timer* that is accessible to every server. The timer is scheduled by the adversary and *defines the phase of a server* locally. It is assumed that the adversary corrupts at most f servers who are in the same local phase. When the adversary corrupts a server, it may provoke arbitrary changes to the server state. However, when a local phase begins, the server is rebooted from a correct state.

The *abstract timer* is formally modelled by a *trivial protocol timer* that works as follows: "Every honest server continuously runs one instance of this protocol, which starts when the server is initialized. Upon initialization, the protocol sends a timer message called a clock tick to itself. Whenever the server receives a clock tick, the server resends the message to itself over the network. The local phase of an uncorrupted server P_i is defined as the number of clock ticks that it has received so far." Note that in the way the problem was specified it would in fact not be possible to put an upper bound on the maximum duration of a local phase, given that network messages do not have bounded delivery time in an asynchronous environment, namely with malicious adversaries. This would render proactive recovery impossible.

The authors remove the impossibility by making a rather strong assumption in the implementation section, that *neither asynchrony nor malicious actions may affect phase changes*. Authors say that, in practice, the start of every local phase may be done through an *impulse from an external clock*, and that an intruder *must not be able to influence it*. In other words, there is an implicit assumption that there is an autonomous watchdog that periodically triggers phase changes, the same type of assumption done in BFT-PR, and thus possessing the same limitations.

3 Conclusions

This paper concisely reports an analysis of a set of existing asynchronous proactive recovery approaches, trying to extract conclusions about the generic types of problems that can be met by the former, as a consequence of the recently discovered impossibility of building asynchronous proactive recovery systems. This analysis allowed us to categorize the four generic concrete problems that may affect any asynchronous system employing proactive recovery [10]: (i.) a malicious adversary may deploy more power than originally assumed, and corrupt nodes at a pace faster than recovery; (ii.) it may attempt to slow down the pace of recovery by any means possible, in order to leverage the chances of intruding the system with the available power; (iii.) it may perform stealth attacks on the system timing, which in asynchronous or partially synchronous systems may not even be perceived by the essentially time-free logic of the system, leaving it defenseless; (iv.) recovery procedures may make it necessary to bring individual nodes to a temporarily inactive state, lowering the redundancy quorum and thus system resilience.

The first problem (violation of attacker power assumptions) is the basic and fundamentally unsolvable problem, since those assumptions are at the heart of the intrusion-tolerance body of research. It must be addressed with techniques that mitigate any leverage an attacker may unexpectedly try to get, such as diversity, obfuscation, or trusted components. However, the other problems must and can be addressed. There is no insurmountable problem in the works surveyed that cannot be addressed by resorting, for example, to hybrid distributed systems models, which allow circumventing the impossibility of building asynchronous proactive recovery systems. In a recent work we have shown how an architecture and generic algorithmic approach to proactive recovery, globally designated *proactive resilience*, addresses each of the remaining problems [9, 7].

References

[1] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *CCS '02: Proc. of the 9th ACM Conf. on Computer and Communications Security*, pages 88–97, 2002.

[2] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, Nov. 2002.

[3] Y. Desmedt. Some recent research aspects of threshold cryptography. In *ISW '97: Proc. of the First Int. Workshop on Information Security*, pages 158–173, 1998.

[4] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Proc. of the 15th Annual International Cryptology Conference on Advances in Cryptology*, pages 339–352. Springer, 1995.

[5] M. A. Marsh and F. B. Schneider. CODEX: A robust and secure secret distribution system. *IEEE Trans. on Dependable and Secure Computing*, 1(1):34–47, January–March 2004.

[6] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 51–59. ACM Press, 1991.

[7] P. Sousa, N. F. Neves, A. Lopes, and P. Verissimo. On the resilience of intrusion-tolerant distributed systems. DI/FCUL TR 06–14, Dep. of Informatics, Univ. of Lisbon, Sept 2006.

[8] P. Sousa, N. F. Neves, and P. Verissimo. How resilient are distributed f fault/intrusion-tolerant systems? In *Proc. of the Int. Conf. on Dependable Systems and Networks*, pages 98–107, June 2005.

[9] P. Sousa, N. F. Neves, and P. Verissimo. Proactive resilience through architectural hybridization. In *Proc. of the ACM Symp. on Applied Computing*, pages 686–690, Apr. 2006.

[10] P. Sousa, N. F. Neves, P. Verissimo, and W. H. Sanders. Proactive resilience revisited: The delicate balance between resisting intrusions and remaining available. In *Proc. of the Symp. on Reliable Distributed Systems*, pages 71–80, Oct. 2006.

[11] L. Zhou, F. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, Nov. 2002.

[12] L. Zhou, F. B. Schneider, and R. V. Renesse. APSS: proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security*, 8(3):259–286, 2005.