Assumptions: The Trojan Horses of Secure Protocols

Paulo Verissimo

Univ. Lisboa
pjv@di.fc.ul.pt

Abstract. Secure protocols rely on a number of assumptions about the environment which, once made, free the designer from thinking about the complexity of what surrounds the execution context.

Henceforth, the designer forgets about the environment and moves on proving her algorithm correct, given the assumptions. When assumptions do not represent with sufficient accuracy the environment they are supposed to depict, they may become the door to successful attacks on an otherwise mathematically correct algorithm. Moreover, this can happen as unwitting to systems as a Trojan Horse's action.

We wish to discuss the theoretical underpinnings of those problems and evaluate some recent research results that demonstrate a few of those limitations in actual secure protocols.

1 Introduction

Secure protocols rely on a number of assumptions about the environment which, once made, free the designer from thinking about the complexity of what surrounds the execution context. Henceforth, the designer forgets about the environment and moves on proving her algorithm correct, given the assumptions. When assumptions do not represent with sufficient accuracy the environment they are supposed to depict, they may become the door to successful attacks on an otherwise mathematically correct algorithm. Moreover, this can happen as unwitting to systems as a Trojan Horse's action.

Intrusion Tolerance has become a reference paradigm for dealing with faults and intrusions, achieving security (and dependability) in an automatic way. However, there are issues specific to the severity of malicious faults (attacks and intrusions) that made the problems and limitations introduced above very visible and, what is more, very plausible. Intrusion-tolerant protocols that deal with intrusions much along the lines of classical fault tolerance, like for example Byzantine agreement, atomic broadcast, state machine replication, or threshold secret sharing, have become a reference in this field.

Using them as example, we wish to discuss the theoretical underpinnings of those problems and evaluate some recent research results that demonstrate a few of those limitations in actual secure protocols.

V. Gorodetsky, I. Kotenko, and V.A. Skormin (Eds.): MMM-ACNS 2007, CCIS 1, pp. 34–41, 2007.

2 Classical Distributed Algorithms Design

The design of distributed algorithms follows a well-determined path and fault/intrusion-tolerant (FIT) algorithms are no exception. The basic proposition underlying the design of FIT algorithms is, informally:

FIT - Given n processes and f a function of n, and a set H of assumptions on the environment, then for at least $n - f$ correct processes, algorithm $\mathcal{A}$ satisfies a pre-defined set of properties $\mathcal{P}$, i.e. executes correctly.

Classical distributed algorithms design has focused its attention on *"algorithm A satisfies a pre-defined set of properties $\mathcal{P}$, i.e. executes correctly"*, consider it the mathematics viewpoint: assumptions are accepted as a given, and it is proved that the algorithm satisfies the properties.

There is nothing wrong with this in principle, but how about looking critically at other parts of the proposition? For example, *"a set H of assumptions on the environment"*. In fact, the usual path is to start with the weakest possible assumptions, normally, in distributed systems security, one talks about arbitrary failure modes, over asynchronous models. The unfortunate fact is that such weak assumptions restrain the amount of useful things that can be done. Namely:

- algorithms are safe, but normally inefficient in time and message complexity;
- deterministic solutions of hard problems such as consensus, Byzantine Agreement or State Machine Replication with Atomic Broadcast are impossible, a result known as FLP impossibility [1];
- furthermore, timed problems (e.g., e-commerce, online stock-exchange, web applications with SLAs, SCADA, etc.) are by definition impossible in a time-free world.

If efficient/performant FIT algorithms are sought, one has to assume controlled failure modes (omissive, fail-silent, etc.). Moreover, for solving the above problems of determinism or for building any timely services (even soft real-time), one must relax the asynchrony assumption and assume at least partially synchronous models. However, this brings a problem: these algorithms will only work to the coverage of those assumptions. Unfortunately, relaxing these assumptions amounts to creating attack space for the hacker, unless something is done to substantiate them:

- controlled failures are hard to enforce in the presence of malicious faults;
- partial synchrony is susceptible to attacks on the timing assumptions;
- even in benign but open settings (e.g., Internet), synchrony is difficult to implement.

As such, there is a significant body of research continuing to assume arbitrary failure modes over asynchronous models, instead turning itself to weakening the semantics of what can be achieved in those conditions.

Some results look very promising and useful if one is to solve non-timed problems with the highest possible coverage:

- toning down determinism, for example, through randomised versions of the above-mentioned algorithms, e.g. consensus;
- tone down liveness expectations, for example, through indulgence, which means that the algorithms may not be guaranteed to terminate, but they will always keep safety;
- use other (sometimes weaker) semantics which do not fall under the FLP impossibility result, for example, through reliable broadcast, secret sharing, quorums, etc.

Another alternative is toning down the allowed fault or intrusion severity, for example, through hybrid fault models [2], which imply that the quorum of f faults that is accepted from a set of processes is divided amongst different fault types, for example Byzantine and crash, $f = f_B + f_c$.

When even the simplest timing is needed, the system can no longer be time-free, for example if a periodical operation or the timely trigger of an execution are needed. In that case, one has to tone down asynchrony, for example, through time-free or timed partially synchronous algorithms [3,4]. Some solutions to circumvent FLP, even non-timed, rely on a form of partial synchrony as well, eventual synchrony, such as the failure detectors [5]. However, these only fare well under benign failure modes.

Unlike the former, these two alternatives pull the boundary of arbitrary failures and of asynchrony, respectively, a bit back, with the corresponding risks in coverage.

3 Assumptions as Vulnerabilities

It is usually said that one should make the weakest assumptions possible. Let us question why. For example, people assume that large-scale (i.e., Internet) systems are asynchronous not for the sake of it, but just because it is hard to substantiate that they behave synchronously. So, confidence (coverage) on the former assumption is higher than on the latter one. Likewise with assuming Byzantine vs. benign behaviour, if the system is open or not very well known. In other words, the asynchrony/Byzantine assumptions would lead to safer designs in this case, though probably not the most effective and efficient.

"Every assumption is a vulnerability"

is a frequently heard quote, which of course leads us right onto the above-mentioned path of *arbitrary failure modes over asynchronous models*. Why? Because it looks like we are not making any assumption: we do not assume anything about the behaviour of processes; we do not assume anything about time.

The caveat is that asynchrony/Byzantine yield so weak a model that it prevents us from doing some important things, as shown earlier. When problems

offer significant hardness, algorithm designers often insert some synchrony in the underlying model, or relax the arbitrary behaviour just a bit, trying to circumvent these difficulties.

According to the quote above, these extra assumptions are vulnerabilities. Furthermore, whilst some are explicit and can deserve attention from the designer, such as the above-mentioned hybrid faults or timed partial synchrony, others are implicit, like assuming that the periodical triggering of a task, a timeout, or the reading of a clock, are meaningful w.r.t. real time in an asynchronous model.

Observation 1 - These subtle feathers of synchrony introduce vulnerabilities which often go undetected, exactly because they are not clearly assumed. In fact, they are bugs in the model, as there are bugs in software: the algorithm designer relies that the system will perform as the assumptions say, but the latter conceal a different behaviour, just as buggy software performs differently than assumed. The consequence is that one may no longer have safe designs with regard to time, despite using asynchronous system models.

4 On Resource Exhaustion

Back to the "pure" Byzantine/asynchronous model, under this no-assumptions model we tend to rely on the fact that we are not making assumptions on time or behaviour, and consider the system extremely resilient. We assume that the system lives long enough to do its job. Can we, really?

In fact, we are still making important assumptions: on the maximum number of allowed faults/intrusions f; on the harshness of the environment or the power of the attacker, respectively for accidental or malicious faults, giving the speed at which the latter are performed; about fault independence or on the expectation that faults/attacks do not occur simultaneously in several nodes or resources. That is, accepting these assumptions as vulnerabilities, those systems, despite following an asynchronous and Byzantine model, are in fact subject to several threats:

- unexpected resource exhaustion (e.g. the number of replicas getting below threshold);
- attacks on the physical time plane (faults and attacks produced at too fast a rate versus the internal pace of the system);
- common-mode faults and attacks (simultaneous attacks against more than one replica).

These problems have been recognized by researchers, who have devised some techniques to keep systems working long enough to fulfil their mission, such as ensuring a large-enough number of replicas at the start and/or using diversity (e.g., different hardware and software, n-version programming, obfuscation) to delay resource exhaustion. However, with static or stationary f-intrusion-tolerant algorithms, even in asynchronous Byzantine environments, it is a matter of time until more than f intrusions occur.

This prompts us for looking critically at other parts of the proposition FIT presented in section 2, like for example *"for at least $n - f$ correct processes"*. In this way, we accept that the proposition is conditional to there being $n - f$ or more correct processes. What if we end-up with less than $n - f$?

Two things may have happened here. That fact may come from an inadequate implementation or design decision, and there is really nothing the algorithm designer can do about it: an adequate algorithm over an inadequate implementation. However, the problem may have a more fundamental cause, something that might be determined at algorithm design time. Were it true, and we would have an inadequate algorithm to start with, and no design or implementation to save it.

How should theory incorporate this fact? For example, by enriching proposition FIT with a safety predicate that would assert or deny something like *"Algorithm $\mathcal{A}$ always terminates execution with at least $n-f$ correct processes."*.

This predicate was called Exhaustion Safety, which informally means that the system maintains the required resources, e.g., the amount of processes or nodes, to guarantee correct performance in the presence of intrusions [6]. As a corollary, an f-intrusion-tolerant distributed system is exhaustion-safe if it terminates before $f + 1$ intrusions being produced. In consequence, a result that would establish at design time that the above predicate would almost never be true or not be guaranteed to be true throughout execution of algorithm $\mathcal{A}$, for whatever real setting, would imply that the algorithm, under the assumed model, would be inadequate to achieve FIT, because it could not be exhaustion-safe.

There has been some research trying to solve the "$n - f$" issue, that is, trying to keep systems working in a perpetual manner or, in other words, achieving exhaustion-safety. The techniques used have been called reactive or proactive recovery (e.g., rejuvenating, refreshing) [7].

Some of these works have assumed an asynchronous model with arbitrary failures in order to make the least assumptions possible. However, given the hardness of the problems to solve, which include for example being able to trigger rejuvenations periodically or reboot a machine within a given delay, these systems end-up making a few 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, giving room for another set of "Trojan-horse"-like pitfalls. What leads to the pitfall is that in normal conditions, clocks and timeouts in asynchronous systems seem to have a real time meaning, but the truth is that a clock is a sequence counter, a timeout does not have a bounded value. System execution and relevant assumptions follow the internal timeline, but attacks can follow the physical timeline. In consequence, there is no formal relation between both, the two timebases are said to be *free-running*. In this case, attacks are produced by hackers in real time (external, physical time), which can in that way delay or stall recovery until more that f intrusions are produced. These problems would have been unveiled if the predicate resource-exhaustion had been used and evaluated at algorithm design time [8].

Observation 2 - The explanation of why systems that are otherwise correct according to the stated assumptions under the asynchronous model used, may fail, is simple. Under attack, the internal timeline can be made to slide w.r.t. the physical timeline at the will of the attacker. For example, for a given speed of an attack along the physical timeline, the internal timeline can be made to progress slowly enough to enable the intrusion. However, since the system is asynchronous, it is completely oblivious to and thus unprotected from, this kind of intrusions: the slow execution could be a legitimate execution. In consequence, harmful as they may be, these intrusions do not even entail a violation of the system's explicit assumptions and resource exhaustion comes unwittingly.

5 On the Substance of Assumptions

Why do the things discussed in the previous sections happen? Let us introduce a postulate:

Postulate 1: Assumptions and models should represent the execution environment of a computation in a substantive and accurate way.

In Computer Science (like in Physics), assumptions should be substantive and models accurate, that is, they should represent the world where the computations are supposed to take place, faithfully enough.

In this scenario, an assumption need not necessarily be a vulnerability: if the assumption depicts the world "as is", there is nothing special that the adversary can attack there. Interestingly, in the formal framework after this postulate, the initial motto might be re-written as:

"Every non-substantiated assumption is a vulnerability"

One should not avoid making assumptions, but instead: *make the least set of assumptions needed to solve the problem at hand, making sure that they have satisfactory coverage*, i.e. that they represent the environment faithfully enough. However, it is not always the case that such a postulate is followed.

Take the synchrony dimension and consider an observation made about it: *"synchronism is not an invariant property of systems"* [9]. In fact, systems are not necessarily either synchronous or asynchronous, the degree of synchronism varies in the time dimension: during the timeline of their execution, systems become faster or slower, actions have greater or smaller bounds. However, and very importantly, it also varies with the part of the system being considered, that is, in the space dimension: some components are more predictable and/or faster than others, actions performed in or amongst the former have better defined and/or smaller bounds.

This opened the door to some research on *hybrid distributed systems models* [10] which helped address and solve some of the contradictions expressed in

the previous sections. Suppose that we endorsed such a hybrid model, where different subsystems can follow different assumptions with regard to time or failure modes.

Assuming, (1) a synchronous channel on a synchronous subsystem coexisting with its asynchronous counterparts under hybrid distributed systems model, is totally different from assuming, (2) that the former is achieved over an environment following the asynchronous model. Under (2), the hypothesis (synchrony) deviates from reality (asynchrony), that is, it has limited substance and as such is more prone to failure, for example, under attack by an adversary. This fragility does not exist at the start under (1), since the hypothesis (synchrony) matches reality (synchrony). Besides, coverage can be made as high as needed by construction, by using architectural hybridisation, a system design technique that matches hybrid distributed systems models [11].

In fact, the same comments apply to a set of constructs proposed recently by several researchers, whose common distinctive feature is the assumption of 'stronger' components in otherwise 'weak' systems: watchdog triggers, real time clocks and timeouts, periodic task dispatchers, crypto chips, trusted component units, synchronous or faster communication channels. Some of these systems contain an unspoken hybridisation, in the sense that they propose hybrid components, but work with a homogenous model and architecture. Those constructs would perhaps be better deployed under a computational model that understands hybridisation, and an architecture that substantiates the 'stronger' assumptions, to avoid the risk of failures, either accidental or deriving from successful attacks to the vulnerabilities caused by the model mismatches introduced [8].

Hybrid distributed systems models, versus homogeneous models, have the advantage of letting different parts of the system follow different assumptions and models. As such, they accurately represent the limit situations causing the problems discussed in Sections 3 and 4.

6 Conclusion

In this paper, we alerted to the fact that, in computer science, assumptions go well beyond simple mathematical abstractions. This certainly applies to the design of secure systems and algorithms. Assumptions should represent with sufficient accuracy the environment they are supposed to depict, else they amplify the attack space of the adversary and may become the door to security failures of an otherwise mathematically correct algorithm. We exemplified situations where this occurs in two sets of scenarios.

For example when arbitrary failure modes over asynchronous models are assumed but some constraints to behaviour and asynchrony are inserted, without necessarily being made explicitly, let alone enforced. These subtle feathers of synchrony introduce vulnerabilities which often go undetected, exactly because they are not clearly assumed. They become natural targets to the adversary.

The second scenario concerned the problem of resource exhaustion, in the same asynchronous models. The speed of attack is fundamental to determine

the pace at which replicas can fail in a FIT algorithm, and in consequence help determine the speed of the defence mechanisms implemented by the same algorithm. However, we showed that no relation between both can be formally defined, because they develop along different timebases which are free-running. However, such a relation is sometimes forced, with the nasty consequence that the system becomes vulnerable in a way that subsequent intrusions may even not entail a violation of the system's explicit assumptions and resource exhaustion may come unwittingly.

Both scenarios define what we may metaphorically call "Trojan-horse"-like pitfalls: the algorithm designer relies that the system will perform as the assumptions say, but in both cases the latter conceal a different behaviour.

References

1. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382 (1985)
2. Meyer, F., Pradhan, D.: Consensus with dual failure modes. In: Proceedings of the 17th IEEE International Symposium on Fault-Tolerant Computing, pp. 214–222. IEEE Computer Society Press, Los Alamitos (1987)
3. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM 35(2), 288–323 (1988)
4. Christian, F., Fetzer, C.: The timed asynchronous system model. In: Proceedings of the 28th IEEE International Symposium on Fault-Tolerant Computing, pp. 140–149. IEEE Computer Society Press, Los Alamitos (1998)
5. Chandra, T., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM 43(2), 225–267 (1996)
6. Sousa, P., Neves, N.F., Verissimo, P.: How resilient are distributed f fault/intrusion-tolerant systems? In: Proceedings of the Int. Conference on Dependable Systems and Networks, pp. 98–107 (2005)
7. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing, pp. 51–59. ACM Press, New York (1991)
8. Sousa, P., Neves, N.F., Verissimo, P.: Hidden problems of asynchronous proactive recovery. In: Third Workshop on Hot Topics in System Dependability (HotDep'07) (2007)
9. Verissimo, P., Casimiro, A.: The Timely Computing Base model and architecture. Transactions on Computers - Special Issue on Asynchronous Real-Time Systems 51(8) (August 2002) A preliminary version of this document appeared as Technical Report DI/FCUL TR 99-2, Department of Computer Science, University of Lisboa (April 1999)
10. Verissimo, P.: Travelling through wormholes: a new look at distributed systems models. SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory) 37(1) (Whole Number 138) (2006)
11. Verissimo, P.: Uncertainty and predictability: Can they be reconciled? In: Schiper, A., Shvartsman, A.A., Weatherspoon, H., Zhao, B.Y. (eds.) Future Directions in Distributed Computing. LNCS, vol. 2584, pp. 108–113. Springer, Heidelberg (2003)