DARPA ITS Workshop, Williamsburg VA 5–6 October 1999

Security and Fault Tolerance Perspectives

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, CA

History

- For most of its history, computer security has focussed on elimination of vulnerabilities
  - Dogma written in to some of the criteria
- Dobson and Randell (1986) pointed out that the real world is not like that
  - Recognizes that even the best defenses may be breached
  - So must try to contain breaches and recover from them
  - Cf. Medieval and Japanese castles
- Intrusion detection is a first step on this path
- This workshop may be the next

Withstanding Attack

- If the bad guys can corrupt your basic environment (operating system, network services) it's hard protect your data
- So must first give thought to fault-tolerant infrastructure
- The dependability perspective: security is fault tolerance wrt. malicious faults
- Requires redundancy to detect faults, and more redundancy to mask them
- Extent of required redundancy depends on the fault model

Fault Models

- Fault-tolerant systems are built to withstand specific kinds and numbers (or rates) of faults—these constitute the chosen fault model
- Kinds of fault range from easy (crash) to difficult (Byzantine)
- Motivation for considering Byzantine faults is different in security than in classical fault tolerance

  Fault Tolerance: making no assumptions about the faults (Byzantine faults are arbitrary) simplifies assurance arguments for truly life-critical systems

  Security: Since the faults are created by an intelligent adversary, they may well exhibit asymmetric symptoms (the hallmark of Byzantine behavior)

Hybrid Fault Models

- Costs more redundancy to withstand large numbers of faults, or difficult kinds of faults
- However, a system designed to withstand, say 2 Byzantine faults (7-fold redundancy) may be overwhelmed by lots of simple faults that a different system with the same redundancy could withstand successfully
- Conversely a system designed to withstand many simple faults may fail against one difficult one
- Uniform effectiveness is the goal of hybrid fault models
- Can often tolerate more faults if you use digital signatures—but will lose if the crypto is broken
- There are hybrid algorithms that use signatures, but still work (not as well) if the crypto is broken

System Synchrony Models

Distributed systems are built to specific synchrony assumptions

Synchronous: there is a fixed, known upper bound on the time it takes nonfaulty participants to do things (e.g., reply to a message)

- Very difficult to achieve (requires dedicated systems, usually those for embedded control)
- If it's merely an assumption, and is violated, synchronous algorithms may fail even to maintain safety properties
- Adding timeouts does not make a system synchronous

System Synchrony Models (continued)

Asynchronous: there is no such upper bound

- Provably impossible to achieve basic services such as consensus, group membership in the presence of even trivial faults

Need intermediate points

Timed Asynchronous: mostly synchronous, safe and live when synchronous, safe when not

Failure detectors: axiomatize how good the timeouts need to be (“eventually weak,” “eventually perfect”)

Research Topics

- The bad guys will attack your fault model
  - So algorithms and architectures for hybrid fault models are a good topic
- The bad guys will attack your synchrony assumptions
  - So algorithms and architectures for weak synchrony assumptions are a good topic
- Combine these to achieve maximum resilience
- I'd go for hybrid fault tolerance under the timed asynchronous model

Research Topics

- You also need to get things back together after you've blown it (e.g., recovery from massive transients)
- Self-stabilizing algorithms are those that eventually converge to a good state after faults stop
- But classical self stabilization assumes no permanent faults
- So combining self stabilization with traditional fault tolerance is a good topic
- Also need to relate these to other algorithms for recovering from transient faults
- Dealing gracefully with partitioning is also an important topic (cf. Totem, Transis)

State of the art

- Algorithms for hybrid fault models are known (and have been formally verified) for synchronous systems
  - There's so much case analysis, it's infeasible without mechanized verification
- Algorithms for asynchronous systems are much more complex and usually consider only simple fault models
- Reiter's Rampart system combined Byzantine fault tolerance with asynchrony (which is impossible, so the exact assumptions need to be clarified)
- There is a strong (and formally verified) foundation for self-stabilization in terms of detectors and correctors (Arora and Kulkarni)

But What About The Data?

- The topics discussed protect the integrity of the system
- Bad guys who destroy or subvert one node cannot break the system, but they may be able to mess with data on the machine they've broken
- Can also use fault tolerance to protect data, provided infrastructure can withstand malicious attack
- Should be transparent to applications, work with COTS
  - E.g., Transparent TMR (hypervisor) layers developed by Randell (80s), Schneider (90s)
  - Vulnerable to denial of service attacks
- Can use scavenging programs to repair mangled data
  - Used successfully (against unintentional faults) by Lucent
  - Has some relationship to transient recovery

And To Stop Them Reading Your Data?

- Encrypt it
- Don't keep it all in one place
- High-value data (e.g., crypto keys): use k of n secret sharing schemes (employed by Reiter in Omega, built on Rampart)
- Lesser material: spread it out in RAID-like fragmentation (as at LAAS)