Intrusion Tolerance: Concepts and Design Principles. A Tutorial

Paulo Veríssimo

DI–FCUL TR–02–6

July 2002

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Intrusion Tolerance: Concepts and Design Principles

Paulo Veríssimo
pjv@di.fc.ul.pt
University of Lisboa Faculty of Sciences

Objective of Tutorial

There is a significant body of research on distributed computing architectures, methodologies and algorithms, both in the fields of dependability and fault tolerance, and in security and information assurance. Whilst they have taken separate paths until recently, the problems to be solved are of similar nature: keeping systems working correctly, despite the occurrence of mishaps, which we could commonly call *faults* (accidental or malicious); ensure that, when systems do fail (again, on account of accidental or malicious faults), they do so in a non harmful/catastrophic way.

In classical dependability, and mainly in distributed settings, fault tolerance has been the workhorse of the many solutions published over the years. Classical security-related work has on the other hand privileged, with few exceptions, intrusion prevention, or intrusion detection without systematic forms of processing the intrusion symptoms.

A new approach has slowly emerged during the past decade, and gained impressive momentum recently: *intrusion tolerance*. That is, the notion of tolerance to a wide set of faults encompassing intentional and malicious faults (we may collectively call them intrusions), which may lead to *failure of the system security properties* if nothing is done to react, counteract, recover, mask, etc., the effect of intrusions on the system state. In short, instead of trying to prevent every single intrusion, the latter are allowed, but *tolerated*: the system has the means to trigger mechanisms that prevent the intrusion from generating a system failure.

The purpose of this tutorial is to expose the audience to these new concepts and design principles. The tutorial reviews previous results under the light of intrusion tolerance (IT), introduces the fundamental ideas behind IT, and presents recent advances of the state-of-the-art, coming from European and US research efforts devoted to IT. The program of the tutorial will address: a review of the dependability and security background; introduction of the fundamental concepts of intrusion tolerance (IT); intrusion-aware fault models; intrusion prevention; intrusion detection; IT strategies and mechanisms; design methodologies for IT systems; examples of IT systems and protocols.

Intrusion Tolerance: Concepts and Design Principles

a Tutorial

Paulo Veríssimo
Univ. of Lisboa Faculty of Sciences
Lisboa – Portugal

pjv@di.fc.ul.pt
http://www.navigators.di.fc.ul.pt

- ***A lot of the material presented here derives from past experience with fault tolerant and secure systems, and from new work and challenging discussions, during the past three years, within the European IST MAFTIA project. I wish to warmly acknowledge the contributions of all members of the team, several of whom contributed results presented here, and collectively have represented a fenomenal thinking tank.***
- ***Further reading on the concepts and design principles presented here can thus be found in:***
  - **Paulo Veríssimo and Luís Rodrigues, *Distributed Systems for System Architects*, Kluwer Academic Publishers, 2001. http://www.navigators.di.fc.ul.pt/dssa/**
  - **A. Avizienis, J.-C. Laprie and B. Randell, *Fundamental Concepts of Dependability*, Research Report N°01145, LAAS-CNRS, April 2001. http://www.laas.fr/**
  - **P. Veríssimo and N. F. Neves, eds., *Service and Protocol Architecture for the MAFTIA Middleware*. Deliv. D23, Project MAFTIA IST-1999-11583, Tech.Rep. DI/FCUL TR-01-1, Univ. Lisboa Jan. 2001. http://www.di.fc.ul.pt/tech-reports/abstract01-1.html**
  - **D. Powell and R. Stroud, eds., *MAFTIA Conceptual Model and Architecture,* Deliv. D2, Project MAFTIA IST-1999-11583, Tech.Rep. DI/FCUL TR-01-10, Univ. Lisboa Nov. 2001. http://www.di.fc.ul.pt/tech-reports/abstract01-10.html**

The case for Intrusion Tolerance

- **Distribution and fault tolerance go hand in hand:**
  - **You distribute to achieve resilience to common mode faults**
  - **You embed FT in a distributed system to resist higher fault probabil.**
- **Security and distribution go hand in hand:**
  - **You break, split, and separate your information**
  - **You make life harder to an attacker**
- **So it looks like we should be talking about (distributed) malicious fault tolerance, a.k.a.**

  (Distributed) Intrusion Tolerance
- **If this is so obvious, why hasn't it happened earlier?**
  - **Distributed systems present fundamental issues and limitations that took a long time to learn**
  - **Classical fault tolerance tolerance follows a framework that is not completely fit to the universe of intentional and/or malicious faults**

Dependability as a common framework

- **Dependability is defined as "that property of a computer system such that reliance can justifiably be placed on the service it delivers"**
- **The service delivered by a system is its behaviour as it is perceptible by its user(s); a user is another system (human or physical) which interacts with the former**
- **Dependability grew under the mental framework of *accidental faults***
- **But all that is defined w.r.t. dependability can be applied to *malicious faults***

Intrusion Tolerance

- **Traditionally, security has involved either:**
  - Trusting that certain attacks will not occur
  - Removing vulnerabilities from initially fragile software
  - Preventing attacks from leading to intrusions
- **In contrast, the tolerance paradigm in security:**
  - Assumes that systems remain to a certain extent vulnerable
  - Assumes that attacks on components or sub-systems can happen and some will be successful
  - Ensures that the overall system nevertheless remains secure and operational
- **Obviously, a complete approach combines tolerance with prevention, removal, forecasting, after all, the classic dependability fields of action!**

1

Introduction to security and information assurance

Security Properties

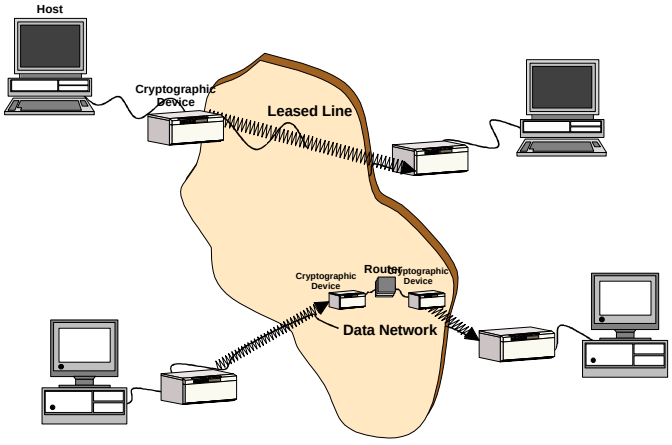
- **Confidentiality**
  - the measure in which a service or piece of information is protected from unauthorized disclosure
- **Authenticity**
  - the measure in which a service or piece of information is genuine, and thus protected from personification or forgery
- **Integrity**
  - the measure in which a service or piece of information is protected from illegitimate and/or undetected modification
- **Availability**
  - the measure in which a service or piece of information is protected from denial of authorized provision or access

Security Frameworks

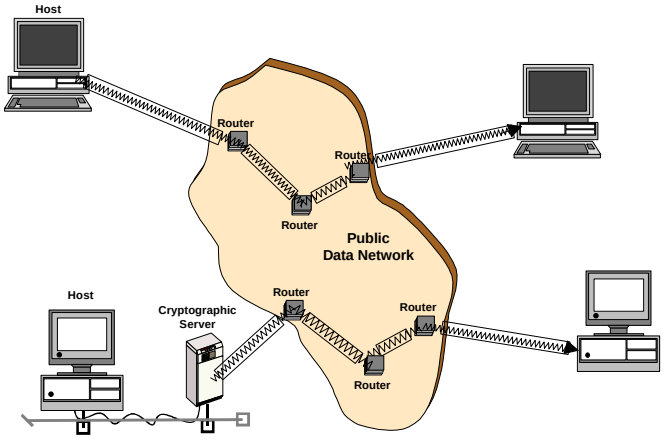
- **Secure Channels and Envelopes**
  - Communicating in a secure way
  - Dispatching information in a secure way
- **Authentication**
  - Ensuring what we deal with is genuine: end-users, data, servers, etc.
- **Protection and Authorization**
  - Protecting resources from unauthorized access
  - Ensuring the users are authorized to do just what they should
- **Auditing and Intrusion Detection**
  - Following the system execution for a posteriori analysis
  - Detecting anomalous usage in runtime

Example Secure Networks and Architectures

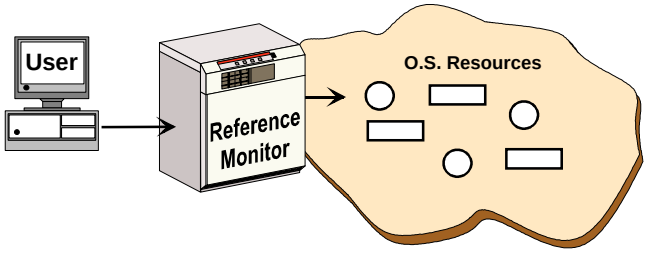
Secure Physical Circuits



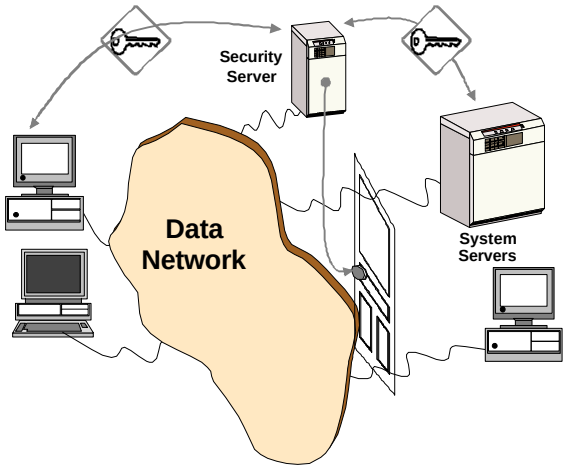
Secure Virtual Circuits



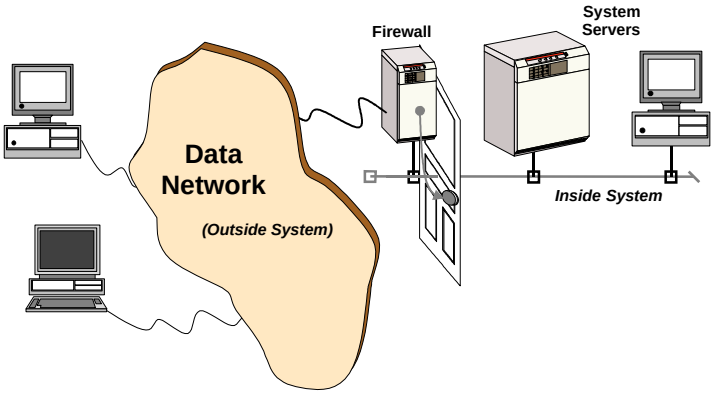
Reference Monitor



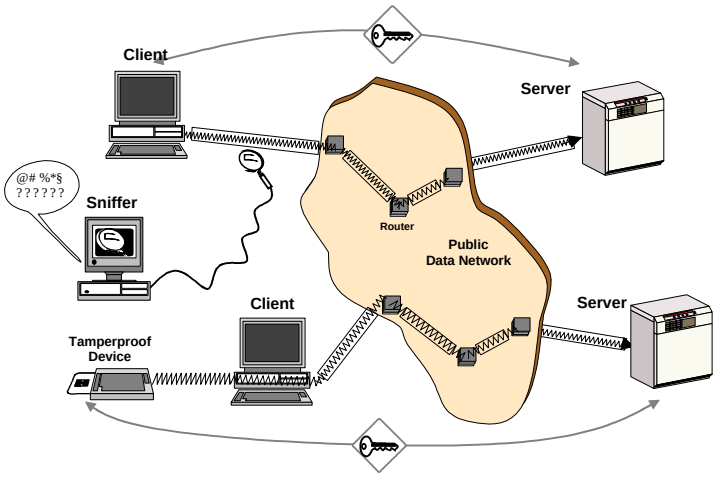
Security Server



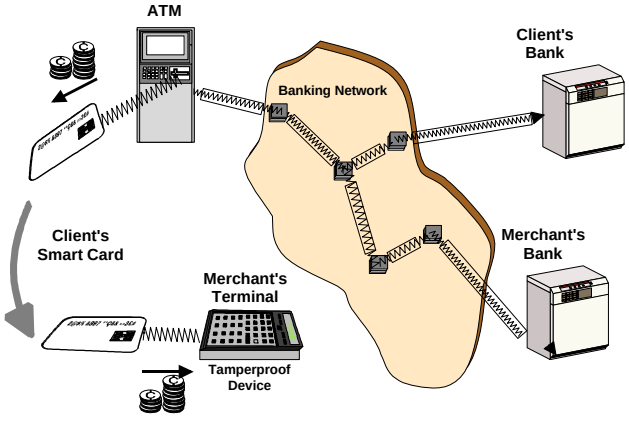
Firewall



Secure Remote Operations



Electronic Payment



Introduction
to
fault tolerance and dependability

The failure of computers

- ***Why do computers fail and what can we do about it?***

  [J. Gray]

- **Because:**
  - **All that works, fails**
  - **We tend to overestimate our HW e SW--- that's called faith☺**
- **So:**
  - **We had better prevent (failures) than remedy**
- **Dependability is ...**
  - **that property of a computer system such that reliance can justifiably be placed on the service it delivers**
- **Why?**
  - **Because (faith notwithstanding) it is the scientific way to quantify, predict, prevent, tolerate, the effect of disturbances that affect the operation of the system**

Does not get better with distribution

- *A distributed system is the one that prevents you from working because of the failure of a machine that you had never heard of.*

  [L. Lamport]

- **Since:**
  - **Machines fail independently, for a start**
  - **They influence each other,**
  - **They communicate through unreliable networks, with unpredictable delays**
- **...gathering machines renders the situation worse:**
  - **The reliability (<1) of a system is the product of the individual component reliabilities, for independent component failures**
  - **R(10 @ 0.99)= 0.9910= 0.90; R(10 @ 0.90)= 0.9010= 0.35**

Can get much worse with malicious failures

- **Failures are no longer independent**
  - **Human attackers are the "common-mode" link**
  - **Components may perform collusion through distributed protocols**
- **Failures become harder**
  - **Components producing malicious, e.g. inconsistent output, at wrong times, forged, etc.**
- **"Fault models"?**
  - **How do you model the mind of an attacker?**

Faults, Errors and Failures

- A system failure occurs when the delivered service deviates from fulfilling the system function
- An error is that part of the system state which is liable to lead to subsequent failure
- The adjudged cause of an error is a fault

- EXAMPLES:
  - Fault --- stuck-at '0' RAM memory register
  - Error --- what happens when the register is read after '1' is written
  - Failure --- the wrong reading ('0') is returned to the user buffer
- SOLUTIONS?
  - Remove the faulty memory chip
  - Detect the problem, e.g. using parity bits
  - Recover from the problem, e.g. using error correcting codes (ECC)
  - Mask the problem, replicating the memory and voting on the readings

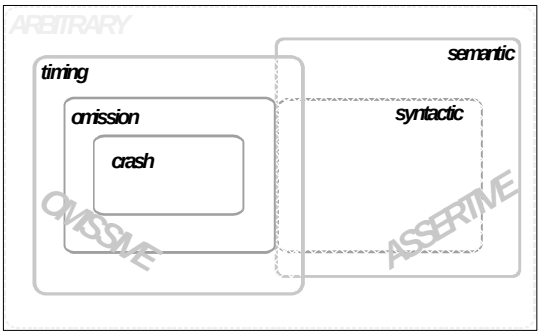
Types of Faults

- Physical
- Design
- Interaction (*)
- Accidental vs. Intentional vs. Malicious (*)
- Internal vs. External
- Permanent vs. Temporary
- Transient vs. Intermittent

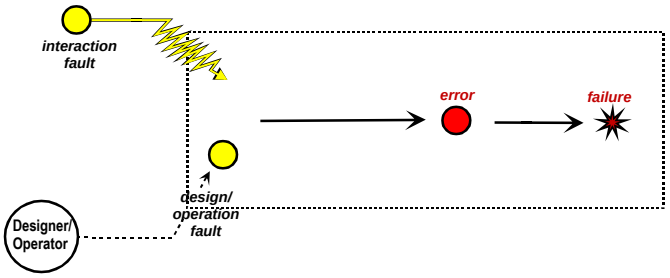
(*) Especially important in distributed systems and security

Interaction Fault classification

- **Omissive**
  - Crash
    - » host that goes down
  - Omission
    - » message that gets lost
  - Timing
    - » computation gets delayed
- **Assertive**
  - Syntactic
    - » sensor says air temperature is 100º
  - Semantic
    - » sensor says air temperature is 26º when it is 30º



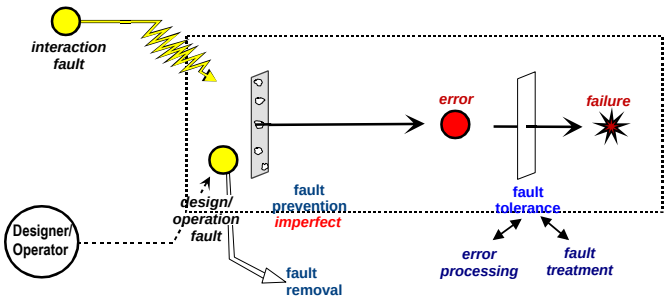
sequence fault® error® failure



Achieving dependability

- **Fault prevention**
  - how to prevent the occurrence or introduction of faults
- **Fault tolerance**
  - how to ensure continued correct service provision despite faults
- **Fault removal**
  - how to reduce the presence (number, severity) of faults
- **Fault forecasting**
  - how to estimate the presence, creation and consequences of faults

Dependability measures



Measuring dependability

- **Reliability**
  - the measure of the continuous delivery of correct service (ex. MTTF)
- **Maintainability**
  - the measure of the time to restoration of correct service (ex. MTTR)
- **Availability**
  - measure of delivery of correct service with respect to alternation between correct and incorrect service (ex. MTBF/(MTBF+MTTR))
- **Safety**
  - the degree to which a system, upon failing, does so in a non-catastrophic manner
- **Integrity**
  - the measure in which a service or piece of information is protected from illegitimate and/or undetected modification
- **Confidentiality**
- **Authenticity**

The "Dependability Tree"

IMPAIRMENTS	• FAULTS • ERRORS • FAILURES
MEANS	• FAULT PREVENTION • FAULT TOLERANCE • FAULT REMOVAL • FAULT FORECASTING
ATTRIBUTES	• RELIABILITY • MAINTAINABILITY • AVAILABILITY • SAFETY • CONFIDENTIALITY • INTEGRITY • AUTHENTICITY

Foundations of (Modular and Distributed) Fault-Tolerant Computing

Forms of redundancy

- **Space redundancy**
  - several copies of the same component
  - same information stored in several disks
  - different nodes compute same result in parallel
  - messages disseminated along different paths
- **Time redundancy**
  - doing the same thing more than once, in same or different ways
  - retransmission of lost messages
  - repeating computations that have aborted
- **Value redundancy**
  - adding extra information about the data being stored or sent
  - codes that allow the detection or correction of integrity errors
  - parity bit or ECC added to memory chips or disk structures
  - frame check sequences or cyclic redundancy in xmitted data
  - cryptographic message signatures

Error processing techniques

- **error detection**
  - detecting the error after it occurs aims at: confining it to avoid propagation; triggering error recovery mechanisms; triggering fault treatment mechanisms
- **error recovery**
  - recovering from the error aims at: providing correct service despite the error

  backward recovery:

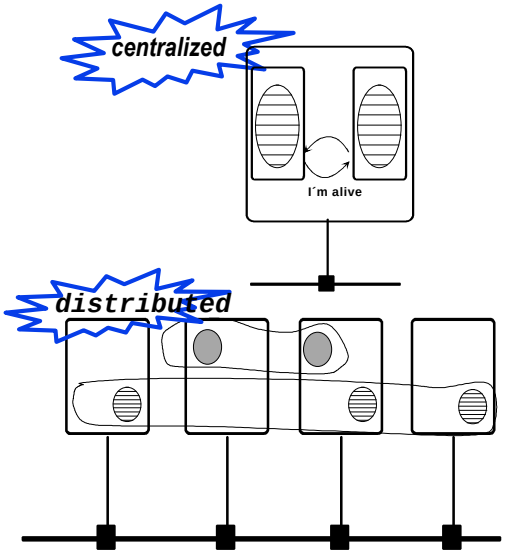
  the system goes back to a previous state known as correct and resumes

  forward recovery:

  the system proceeds forward to a state where correct provision of service can still be ensured
- **error masking**
  - the system state has enough redundancy that the correct service can be provided without any noticeable glitch

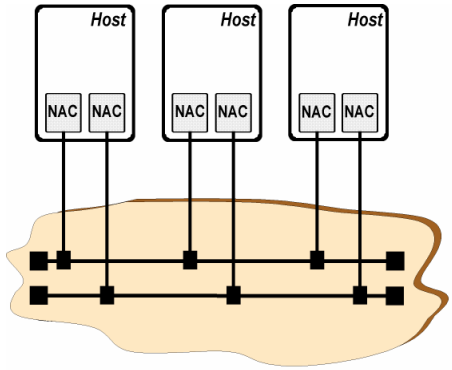
Foundations of modular and distributed fault tolerance

- **Topological separation**
  - failure independence
  - graceful degradation
- **Replication**
  - software vs. hardware
  - fine granularity
  - Resource optimization
- **incremental T/F by:**
  - class (omissive, semantic)
  - number of faults
  - number of replicas
    - » pairs, triples, etc.
  - Type of replica control
    - » active, passive
    - » round robin, voting

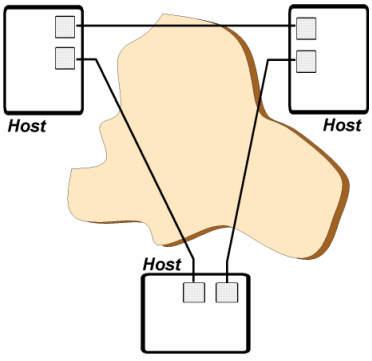


Example Fault Tolerant Networks and Architectures

Redundant Networks

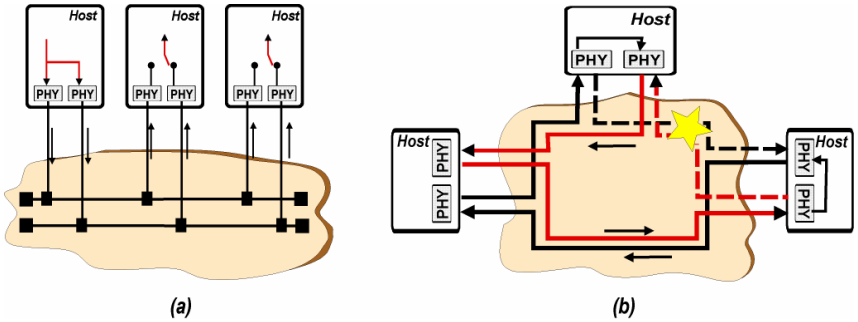


(a)

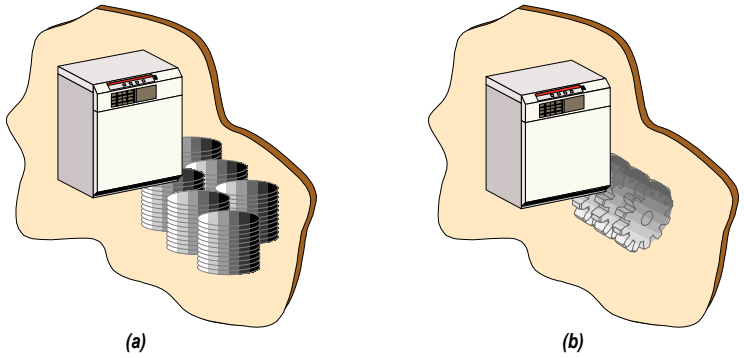


(b)

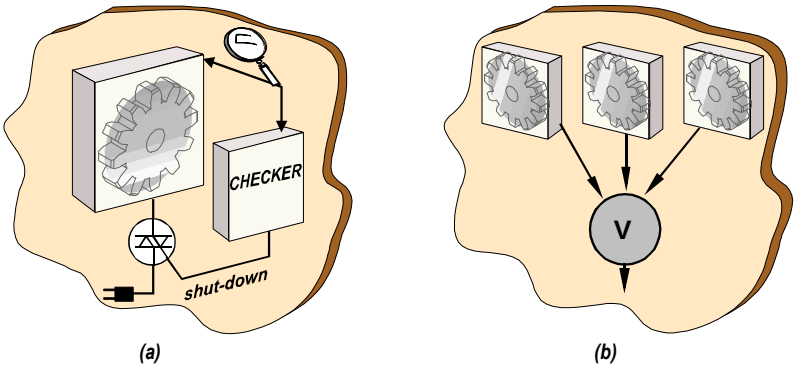
Redundant Media Networks



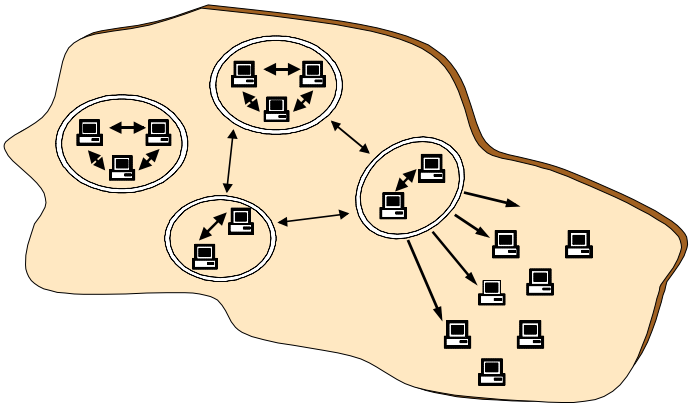
Redundant Storage and Processing



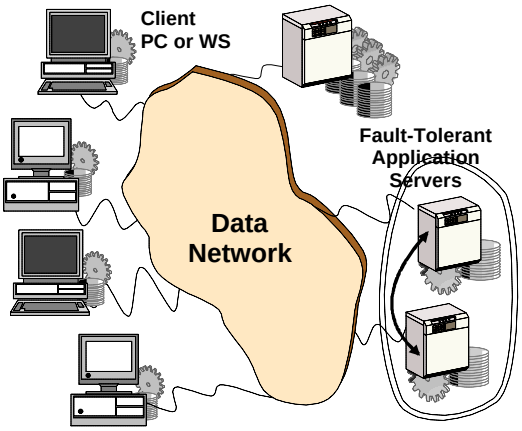
Error Detection and Masking



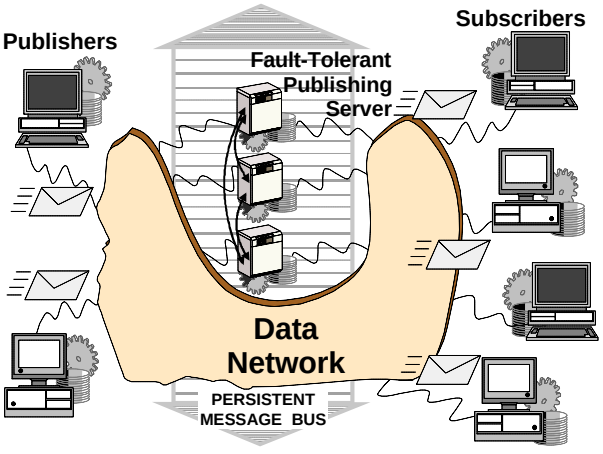
Modular Distributed FT with Replica Sets



Client-Server with FT Servers



FT Publisher-Subscriber



2

Intrusion Tolerance (IT) concepts and terminology

What measures the risk of intrusion?

- **RISK is a combined measure of the level of threat to which a computing or communication system is exposed, and the degree of vulnerability it possesses:**

  RISK = VULNERABILITY X THREAT

- ***The correct measure of how potentially insecure a system can be (in other words, of how hard it will be to make it secure) depends:***
  - ***on the number and severity of the flaws of the system (vulnerabilities)***
  - ***on the potential of the attacks it may be subjected to (threats)***

An example

- ***Consider the following two example systems:***
  - ***System Vault has a degree of vulnerability v_{vault}=0.1, and since it has such high resilience, its designers have put it to serve anonymous requests in the Internet, with no control whatsoever to whoever tries to access it, that is, subject to a high level of threat, t_{vault}=100.***
  - ***System Sieve, on the other hand, is a vulnerable system, v_{sieve}=10, and in consequence, its designers have safeguarded it, installing it behind a firewall, and controlling the accesses made to it, in what can be translated to a level of threat of t_{sieve}=1.***
- ***Which of them offers a lower operational risk?***
- ***Consider the product threat x vulnerability:***
  - ***with the imaginary values we attributed to each system, it equates to the same value in both systems (10), although system Vault is a hundred times less vulnerable than system Sieve.***

Should we bring the risk to zero? And... can we?

- **This is classical prevention/removal**
  - ***of the number and severity of the flaws of the system (vulnerabilities)***
  - ***of the potential of the attacks it may be subjected to (threats)***
- **We cannot make either arbitrarily low**
  - **too costly and infeasible**
  - **certain attacks come from the kind of service being deployed**
  - **certain vulnerabilities are attached to the design of the system proper**

- **...and the question is: should we?**
- **can't we talk about acceptable risk?**
- **doesn't the hacker also incur in a cost of intruding??!!**

Another example

- *Is SSL secure?*
  - *Secure Sockets Layer (SSL) reportedly ensures secure client-server interactions between browsers and WWW servers. Users have tended to accept that the interactions are secure (assumed low degree of vulnerability), without quantifying how secure.*
  - *Several companies have built their commerce servers around SSL, some of them to perform financially significant transactions on the Internet (high level of threat).*
  - *Netscape's SSL implementation broken because of a bug that allowed to replicate session keys and thus decrypt any communication. The corrected version was then broken at least twice through brute-force attacks on the 40-bit keys version.*
  - *This initial situation led to a high risk*

Another example (cont.)

- *Is SSL secure enough?*
  - *Netscape reported that it would cost at least USD10,000 to break an Internet session on the second version of SSL, in computing time.*
  - *The cost of intruding a system versus the value of the service being provided allows the architect to make a risk assessment. Someone who spends 10 000 EURO to break into a system and get 100 EURO worth of bounty, is doing a bad business.*
  - *This defined the acceptable risk. Unfortunately, these estimates may fail: shortly after Netscape's announcement, a student using a single but powerful desktop graphics workstation, broke it for just USD600.*
  - *However, what went wrong here was not the principle and the attitude of Netscape, just the risk assessment they made, which was too optimistic.*

What is Intrusion Tolerance?

- **The tolerance paradigm in security:**
  - Assumes that systems remain to a certain extent vulnerable
  - Assumes that attacks on components or sub-systems can happen and some will be successful
  - Ensures that the overall system nevertheless remains secure and operational, with a measurable probability
- **In other words:**
  - Faults-- malicious and other--- occur
  - They generate errors, i.e. component-level security compromises
  - Error processing mechanisms make sure that security failure is prevented
- **Obviously, a complete approach combines tolerance with prevention, removal, forecasting, after all, the classic dependability fields of action!**

Intrusion Tolerance

Fault Models

Classical methodologies

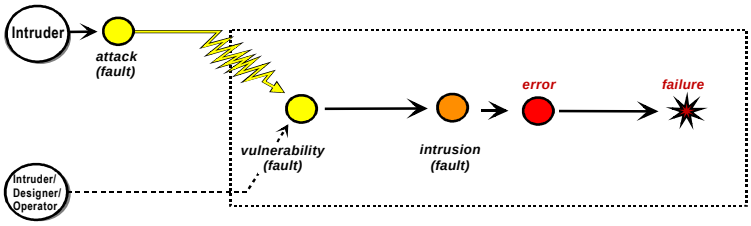
Error processing

Fault treatment

Attacks, Vulnerabilities, Intrusions

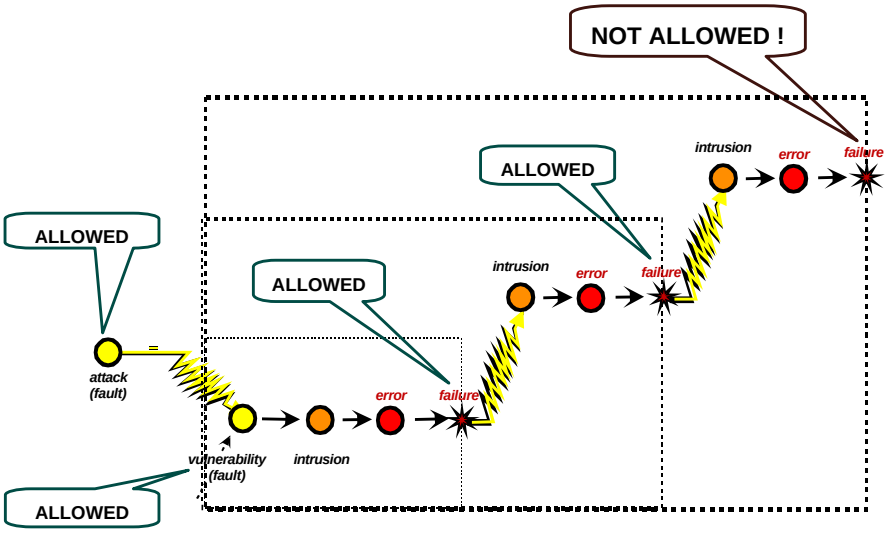
- **Intrusion**
  - an externally induced, intentionally malicious, operational fault, causing an erroneous state in the system
- **An intrusion has two underlying causes:**
- **Vulnerability**
  - malicious or non-malicious weakness in a computing or communication system that can be exploited with malicious intention
- **Attack**
  - malicious intentional fault introduced in a computing or comm's system, with the intent of exploiting a vulnerability in that system
  - without attacks, vulnerabilities are harmless
  - without vulnerabilities, there cannot be successful attacks
- **Hence: attack + vulnerability ® intrusion ® error ® failure**
  - A specialization of the generic *"fault,error,failure"* sequence

Attack-Vulnerability-Intrusion composite fault model



AVI sequence : attack + vulnerability ® intrusion ® error ® failure

Faults in Cascade



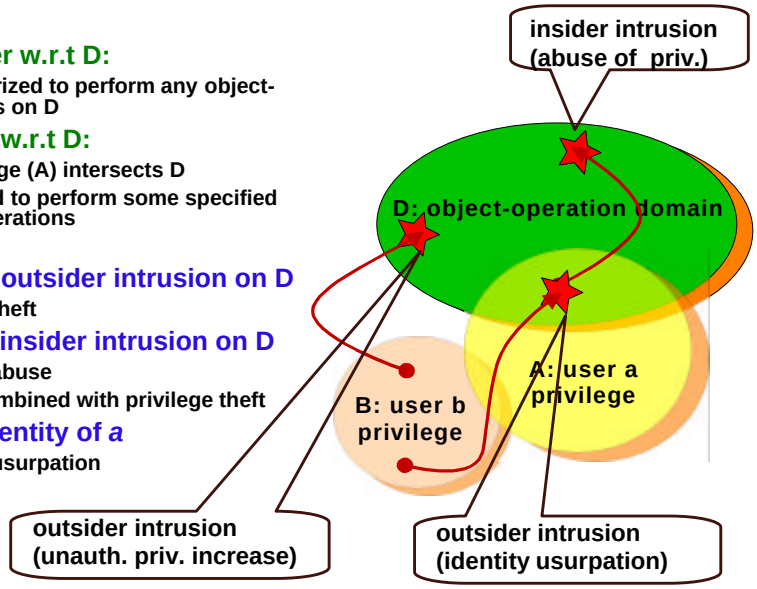
Who's fault was it?

- **Subjects undergo authentication, to prove an identity:**
  - **they are who they say**
- **Subjects receive authorization, in the measure of their privilege:**
  - **allowed ops are defined by access rights on objects: *<subject,object,rights>***
  - **privilege is set of rights over a sub-set in the universe of object-oper. pairs**
- **This allows finer definitions of subject status, although a good approximation is: Outsider Û Privilege = Ø**
- ***outsider vs. insider* derives from security perimeter assumption:**
  - **inadequate for open distr. syst.: insiders can/should have different privileges**

- **Theft of privilege:**
  - **an unauthorised increase in privilege, i.e., a change in the privilege of a user that is not permitted by the system's security policy**
- **Abuse of privilege:**
  - **a misfeasance, i.e., an improper use of authorised operations**
- **Usurpation of identity:**
  - **impersonation of a subject *a* by a subject *i*, who with that usurps *a*'s privilege**

Outsider vs Insider intrusions

- ***b* is outsider w.r.t D:**
  - **not authorized to perform any object-operations on D**
- ***a* is insider w.r.t D:**
  - **his privilege (A) intersects D**
  - **authorized to perform some specified object-operations**
- ***b* performs outsider intrusion on D**
  - **privilege theft**
- ***a* performs insider intrusion on D**
  - **privilege abuse**
  - **maybe combined with privilege theft**
- ***b* usurps identity of *a***
  - **privilege usurpation**



Intrusion Tolerance

Fault Models

Classical methodologies

Error processing

Fault treatment

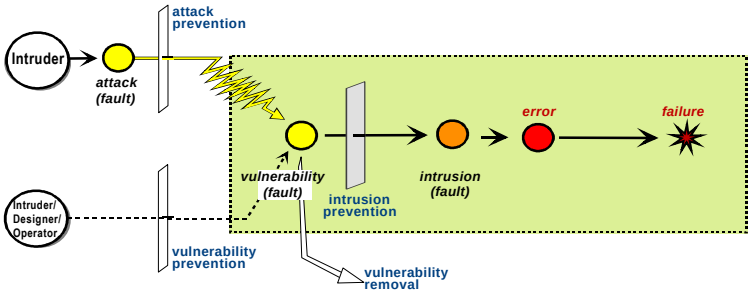
Achieving dependability w.r.t. malicious faults (the classical ways...)

- **Attack prevention**
  - Ensuring attacks do not take place against certain components
- **Attack removal**
  - Taking measures to discontinue attacks that took place
- **Vulnerability prevention**
  - Ensuring vulnerabilities do not develop in certain components
- **Vulnerability removal**
  - Eliminating vulnerabilities in certain components (e.g. bugs)

Examples

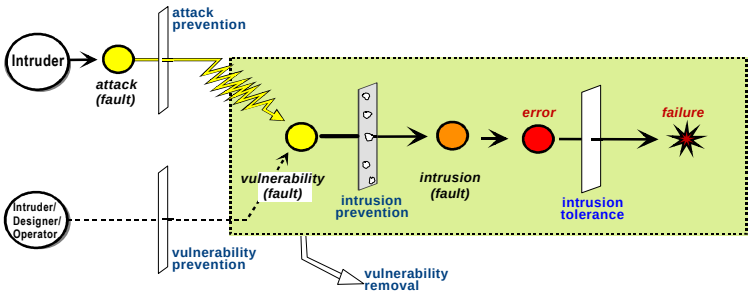
- **Attack prevention**
  - selectively filtering access to internal parts of the system (e.g., if a component is behind a firewall and cannot be accessed from the Internet, attack from there is prevented)
  - disabling JavaScript and/or Java prevents attacks by malicious scripts or applets
- **Attack removal**
  - identifying source of an external attack and taking measures to terminate it
- **Vulnerability prevention**
  - best practice in software development
  - measures preventing configuration and operation faults
- **Vulnerability removal**
  - of: coding faults allowing program stack overflow, files with root setuid in UNIX, naive passwords, unprotected TCP/IP ports

AVI Composite fault model



➢sequence : attack + vulnerability® intrusion® failure

AVI Composite fault model



➢sequence : attack + vulnerability® intrusion® failure

Intrusion Tolerance

Fault Models
Classical methodologies
Error processing
Fault treatment

Processing the errors deriving from intrusions

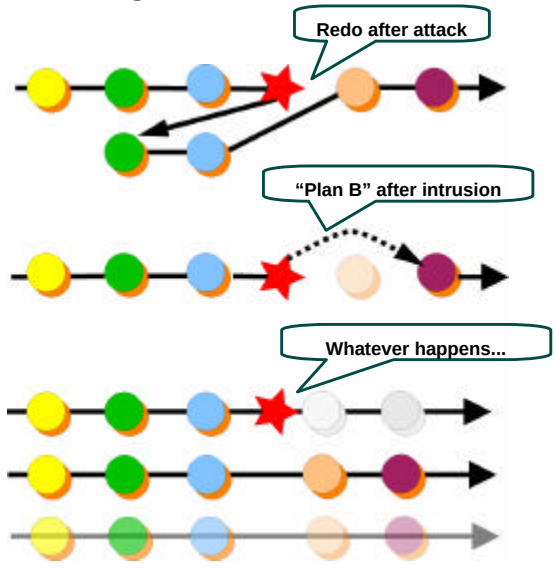
- **error detection**
  - detecting the error after it occurs aims at: confining it to avoid propagation; triggering error recovery mechanisms; triggering fault treatment mechanisms
  - **modified files or messages; phony OS account; sniffer in operation; host flaky or crashing on logic bomb**
- **error recovery**
  - recovering from the error aims at: providing correct service despite the error
  - **recovering from effects of intrusions**

Processing the errors deriving from intrusions

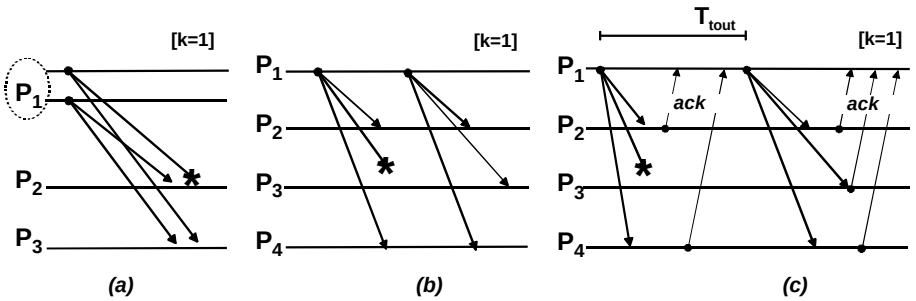
- **backward recovery:**
  - the system goes back to a previous state known as correct and resumes
  - system suffers DOS (denial of service) attack, and re-executes the corrupted operation
  - system detects corrupted files, pauses, reinstalls them, goes back
- **forward recovery:**
  - or proceeds forward to a state that ensures correct provision of service
  - system detects intrusion, considers corrupted operations lost and increases level of security (threshold/quorums increase, key renewal)
  - system detects intrusion, moves to degraded but safer op mode
- **error masking**
  - redundancy allows providing correct service without any noticeable glitch
  - systematic voting of operations; fragmentation-redundancy-scattering
  - sensor correlation (agreement on imprecise values)

Error processing at work

- **backward recovery**
- **forward recovery**
- **error masking**



Examples of error processing in communication



- **Communication Error Processing:**
  - **(a) Masking (Spatial);**
  - **(b) Masking (Temporal);**
  - **(c) Detection/Recovery**

Intrusion Tolerance

Fault Models
Classical methodologies
Error processing
Fault treatment

Fault Treatment facets

- **Diagnosis**
  - determine cause of error, i.e., the fault(s): location and nature
  - non-malicious or malicious syndrome (intrusion)?
  - attack? --- to allow removal/retaliation
  - vulnerability? --- to allow removal
- **Isolation**
  - prevent new activation
  - intrusion: prevent further penetration
  - attack: disable further attacks of this kind (block the origin)
  - vulnerability: passivate the cause of successful attack (e.g. patch)
- **Reconfiguration**
  - so that fault-free components provide adequate/degraded service
  - contingency plans to degrade/restore service

Intrusion Detection

Classical methodologies

ID as error detection

ID as fault diagnosis

Intrusion Detection

- Classically, ID encompasses all kinds of attempts to detect the presence or the likelyhood of an intrusion
- ID can be performed in real-time, or off-line
- It is directed at any or all of: attacks (e.g. port scan detection), vulnerabilities (e.g. scanning), and intrusions (e.g. correlation engines)
- Definition of ID given by NSA (1998):
  - "Pertaining to techniques which attempt to detect intrusion into a computer or network by observation of actions, security logs, or audit data. Detection of break-ins or attempts either manually or via software expert systems that operate on logs or other information available on the network."

ID system classes

- Behavior-based (or anomaly detection) systems
  - no knowledge of specific attacks
  - provided with knowledge of normal behavior of monitored system, acquired e.g. through extensive training of the system
  - advantages: they do not require a database of attack signatures that needs to be kept up-to-date
  - drawbacks: potential false alarms; no info on type of intrusion, just that something unusual happened
- Knowledge-based (or misuse detection) systems
  - rely on a database of previously known attack signatures
  - whenever an activity matches a signature, an alarm is generated
  - advantage: alarms contain diagnostic information about the cause
  - drawback: potential omitted or missed alarms, e.g. new attacks

ID: Error detection or fault diagnosis?

- classical IDS have two facets
- detecting errors as per the security policy specification
- diagnosing faults as per the system fault model
- consider the following example:
  - *Organization A has an intranet with an extranet connected to the public Internet. It is fit with an IDS*
  - the IDS detects a port scan against one of the extranet hosts, coming from the Internet
  - the IDS detects a port scan against an internal host, coming from the intranet
  - *what is the difference?*

Intrusion Detection

Classical methodologies

ID as error detection

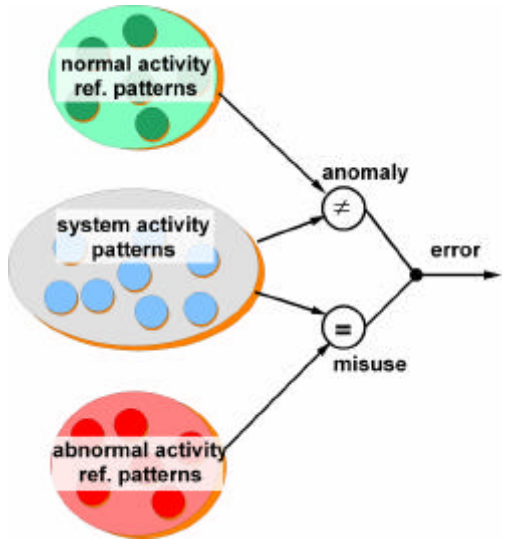
ID as fault diagnosis

ID as error detection

- addresses detection of erroneous states in a system's computation, deriving or not from malicious action
  - e.g. modified files or messages, OS penetration
- this puts emphasis on the result, rather than on the cause (AVI- attack, vulnerability, intrusion): the observable failure of some component to provide correct service
- the possible causes have been defined previously when devising the fault model
- any detector of errors caused by malicious faults should detect errors caused by non-malicious ones
  - ex. a byzantine (component) failure detector in a distributed system, detects an abnormal behavior of components: sending inconsistent info to different participants. Whether or not it is caused by malicious entities, is irrelevant

Detection mechanisms

- consider system activity specified by patterns
- anomaly detection
  - looks for deviation from NORMAL ACTIVITY PATTERNS
- misuse detection
  - looks for existence of ABNORMAL ACTIVITY PATTERNS
- we can have hybrids
- Quality of Service
  - false alarm rate
  - omitted alarm rate



Intrusion Detection

Classical methodologies

ID as error detection

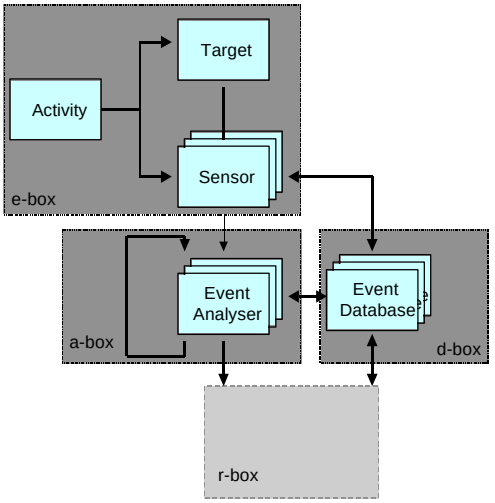
ID as fault diagnosis

ID as fault diagnosis

- **error detection's first impact is on automated fault tolerance mechanisms**
- **regardless of the error processing mechanism (recovery or masking), administration subsystems have a paramount action w.r.t. fault diagnosis**
- **this facet of classical ID fits into fault treatment**
- **Intrusion diagnosis**
  - **assessing the degree of success of the intruder in terms of corruption of components and subsystems**
- **Vulnerability diagnosis**
  - **determining the channels through which the intrusion took place**
- **Attack diagnosis**
  - **finding out who/what performed the attack**
- **diagnosis can be done proactively, before errors happen**
  - **by activating faults (e.g. vulnerability scanning) and post-processing (forecasting their effects) one can get a metrics of resilience (subject to the method coverage...)**

CIDF (Common Intrusion Detection Framework)

- **CIDF proposes a structure for intrusion-detection systems:**
- **e-box (event generator) gathers event information**
- **a-box (analysis box) analyses event information, detecting errors and diagnosing faults**
- **d-box (database) saves persistent information for the IDS**
- **r-box (response box) acts upon the results of analysis**
- ***in fact, r-box is not intrusion detection, but error recovery and fault treatment***



Intrusion Tolerance (IT) mechanisms and strategies

Classical security strategies

- **Openess of system security policies**
  - **The 4P policies: paranoid, prudent, permissive, promiscuous**
- **Zero-Vulnerabilities**
  - **e.g. Trusted Computing Bases (TCB) assumed to be tamperproof**
- **Attack Prevention**
  - **e.g. firewalls preventing direct access to inside resources**
- **Detection and Reaction to Intrusion**
  - **e.g. the intrusion detection process, and countermeasures**
- **Disruption avoidance**
  - **Maintaining availability and integrity against attacks**

Classical fault tolerance strategies

- **Fault Tolerance versus Fault Avoidance in HW-FT**
  - **tradeoff between reliable but expensive components and less performant and more complex mechanisms**
- **Tolerating Design Faults**
  - **going beyond HW-FT, helpless with common-mode faults (e.g. SW)**
- **Perfect Non-stop Operation?**
  - **when no perceived glitch is acceptable**
- **Reconfigurable Operation**
  - **less expensive, when a glitch is allowed**
- **Recoverable Operation**
  - **cheap, when a noticeable but acceptable service outage allowed**
- **Fail-Safe versus Fail-Operational**
  - **safety track--- when a fault cannot be tolerated, two hypothesis: shutdown, or contingency plan for degraded op. mode**

Modeling malicious failures

- **What are malicious failures?**
  - **how do we model the mind and power of the attacker?**
- **Basic types of failure assumptions:**
  - **Controlled failures : assume qualitative and quantitative restrictions on compon. failures, hard to specify for malicious faults**
  - **Arbitrary failures : unrestricted failures, limited only to the "possible" failures a component might exhibit, and the underlying model (e.g. synchronism)**
- **Fail-controlled vs. fail-arbitrary models in face of intrusions**
  - **FC have a coverage problem, but are simple and efficient**
  - **FA are normally inefficient, but safe**

The problem of time and timeliness

- **Why can't we have secure synchronous (real-time) protocols?**
- **Synchronous models (timed):**
  - **use time, a powerful construct to solve timed problems**
  - **yield simple algorithms**
  - **but susceptible to attacks on timing assumptions**
- **Solutions:**
  - **don't use time (asynchronous models) OR**
  - **make indulgent timing assumptions, ones that resist a certain level of threat (timed partially asynchronous models) OR**
  - **protect time under "good" subsystems, i.e. make sure that timing assumptions are never violated (real-time security kernels)**

The problem of time and timeliness

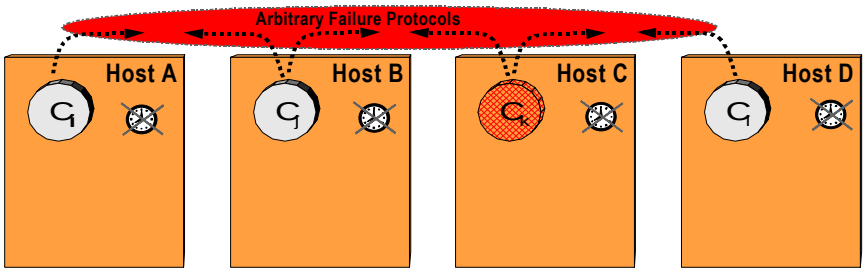
- **Asynchronous model (time-free):**
  - **resist attacks on timing assumptions**
  - **no deterministic solution of hard problems e.g. consensus, BA**
  - **efficient probabilistic approaches**
  - **does not solve timed problems (e.g., e-com, stocks)**
- **Partial Synchrony (timed):**
  - **exploit the power of intermediate models**
  - **accommodate several degrees of sync/async.**
  - **lives with indulgent timing assumptions**
  - ***Ex: a message is delivered within 100ms with 90% probability; a message may take a very long time to be delivered but I'll know accurately whether it is delayed or the sender crashed***
- **Real-Time security kernels**
  - **protect time from attackers and other faults**

Arbitrary failure assumptions

- **probl. of coverage of controlled failure assumptions:**
  - **all lies on the coverage of the fail-controlled subsystem assumptions**
- **operations of very high value and/or criticality:**
  - **financial transactions of very high value**
  - **contract signing; provision of long term credencials**
  - **risk of failure due to violation of assumptions cannot be incurred**
- **arbitrary-failure resilient building blocks (e.g. Byzantine agreement protocols):**
  - **no assumptions on existence of security kernels or other fail-controlled components**
  - **time-free approach, i.e. no assumptions about timeliness**

Fail-uncontrolled IT protocols

- Time-free
- Arbitrary failure environment
- Arbitrary failure protocols
- Used in: probabilistic Byzantine-agreement based set of protocols



Modeling malicious failures

- Intrusion-aware composite fault models
  - the competitive edge over the hacker
  - AVI: attack-vulnerability-intrusion fault model
- Combined use of prevention and tolerance
  - malicious failure universe reduction
  - attack prevention, vulnerability prevention, vulnerability removal, in system architecture subsets and/or functional domains subsets
- Hybrid failure assumptions
  - different failure modes for distinct components
  - reduce complexity and increase performance, maintaining coverage
- Quantifiable assumption coverage
  - fault forecasting (on AVI)

Did you say trusted?

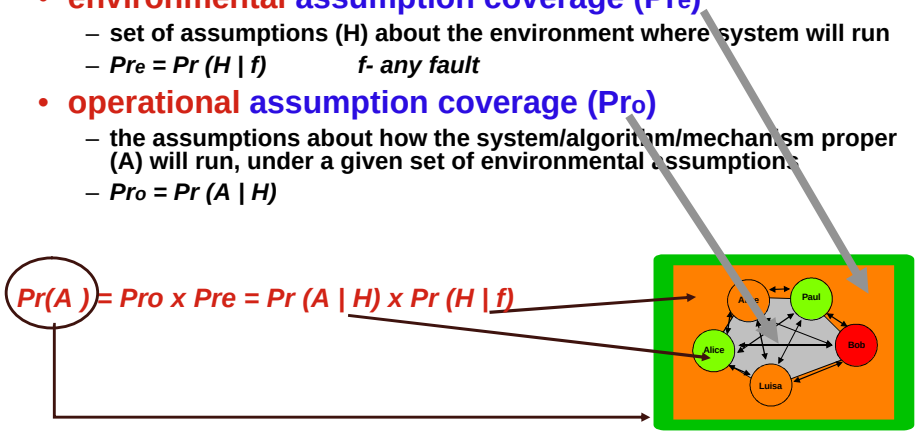
- **Sometimes components are tamper-proof, others tamper-resistant...**
  - **Watch-maker syndrome:**
    - » ***--- "Is this watch waterproof?"***
    - » ***--- "No, it's water-resistant"***
    - » ***--- "Anyway, I assume that I can swim with it!"***
    - » ***--- "Well…yes, you can… but i wouldn't trust that very much"***
- **How can something trusted be not trustworthy?**
  - ***Unjustified reliance syndrome:***
    - » ***--- "I trust Alice"***
    - » ***--- "Well Bob, you shouldn't, she's not trustworthy"***
- **What is the difference? If we separate specification from implementation, and provide notions of *justification* and of *coverage*, all becomes clearer**

Trust, Trustworthy, Tamperproof

- **Trust**
  - **Reliance. Belief that something is dependable**
- **Trustworthy**
  - **Dependable. Property of a (sub)system that makes us justifiably rely on it**
- **Tamperproof**
  - **Property of a system/component of being shielded, i.e. whose attack model is that attacks can only be made at the regular interface**
  - **Coverage of the "tamperproof" assumption may not be perfect**
- **Example:**
  - ***Implementation of an authorisation service using Java Cards to store private keys. We assume J.Cards are tamperproof, and so we argue that they are trustworthy (they will not reveal these keys to an unauthorised party). Hence we can justifiably argue that the service is trusted, with the coverage given by our assumptions, namely, the tamperproofness of JCards***

On coverage and separation of concerns

- **predicate P holds with a coverage Pr**
  - **we say that we are confident that P has a probability Pr of holding**
- **environmental assumption coverage (Pr_e)**
  - **set of assumptions (H) about the environment where system will run**
  - ***Pr_e = Pr (H | f)*** ***f- any fault***
- **operational assumption coverage (Pr_o)**
  - **the assumptions about how the system/algorithm/mechanism proper (A) will run, under a given set of environmental assumptions**
  - ***Pr_o = Pr (A | H)***



Hybrid failure assumptions considered useful

- **Classic hybrid fault models**
  - **flat, use stochastic foundation to explain different behavior from same type of components (i.e. *k* crash and *w* byzantine in vector of values)**
- **The problem of well-foundedness**
  - **an intentional player defrauds these assumptions**
- **Architectural hybridation**
  - **different assumptions for distinct component subsets**
  - **behavior enforced by construction: trustworthiness**

Intrusion tolerance with hybrid failure assumptions

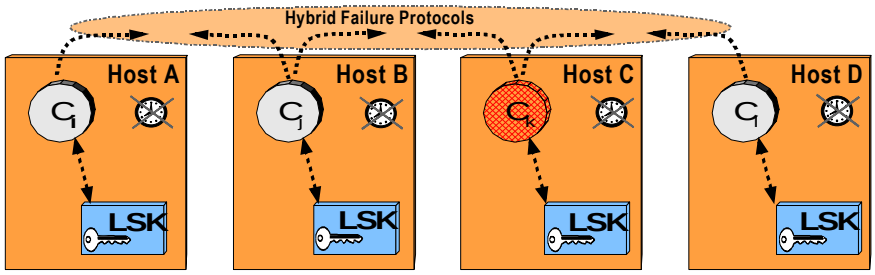
- **Composite fault model with hybrid failure assumptions:**
  - **the presence and severity of vulnerabilities, attacks and intrusions varies from component to component**
- **Trustworthiness:**
  - **how to achieve coverage of controlled failure assumptions, given unpredictability of attacks and elusiveness of vulnerabilities?**
- **Design approach:**
  - **modular architectures**
  - **combined use of vulnerability prevention and removal, attack prevention, and component-level intrusion tolerance, to justifiably impose a given behavior on some components/subsystems**
- **Trusted components:**
  - **fail-controlled components with justified coverage (trustworthy), used in the construction of fault-tolerant protocols under hybrid failure assumptions**

Intrusion tolerance with hybrid failure assumptions

- **Using trusted components:**
  - **black boxes with benign behavior, of omissive or weak fail-silent class**
  - **can have different capabilities (e.g. synchronous or not; local or distributed), can exist at different levels of abstraction**
- **Fault-tolerant protocols:**
  - **more efficient than truly arbitrary assumptions protocols**
  - **more robust than non-enforced controlled failure protocols**
- **Tolerance attitude in design:**
  - **unlike classical prevention-based approaches, trusted components do not mediate all accesses to resources and operations**
  - **assist only crucial steps of the execution of services and applications**
  - **protocols run in untrusted environment, local participants only trust trusted components, single components can be corrupted**
  - **correct service built on distributed fault tolerance mechanisms, e.g., agreement and replication amongst participants in several hosts**

Fail-controlled IT protocols with Local Security Kernels

- Trustworthy component - Local Security Kernel (LSK) (e.g. smart or Java card; appliance board)
- Time-free
- Arbitrary failure environment + LSK
- Hybrid failure protocols
- Example usage: FT distributed authentication and authorisation protocols



Intrusion tolerance with hybrid failure assumptions

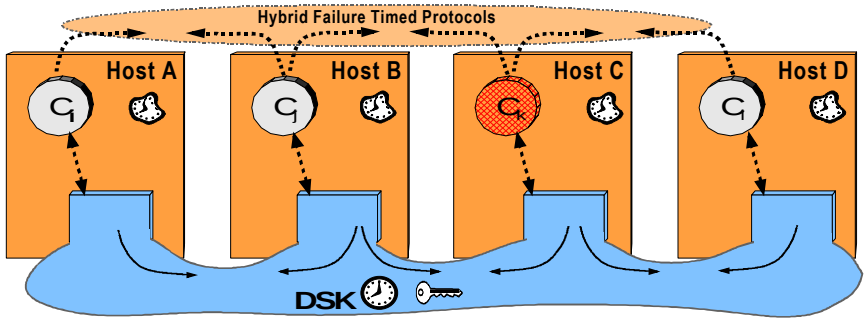
- distributed security kernels (DSK):
  - amplifying the notion of local security kernel, implementing distributed trust for low-level operations
  - based on appliance boards with a private control channel
  - can supply basic distributed security functions
- how DSK assists protocols:
  - protocol participants exchange messages in a world full of threats, some of them may even be malicious and cheat
  - there is an oracle that correct participants trust, and a channel that they can use to get in touch with each other, even for rare moments
  - acts as a checkpoint that malicious participants have to synchronise with, and this limits their potential for Byzantine actions

Intrusion tolerance under partial synchrony

- **real-time distributed security kernels (DSK):**
  - control channel might as well provide reliable clocks and timely (synchronous) inter-module communication
  - ensures implementation of strong paradigms (e.g. perfect failure detection, consensus)
- **protocols can now be timed**
  - timed despite the occurrence of malicious faults
- **how DSK assists protocols:**
  - determine useful facts about time (be sure it executed something on time; measure a duration; determine it was late doing something)

Fail-controlled IT protocols with a Distributed Security Kernel

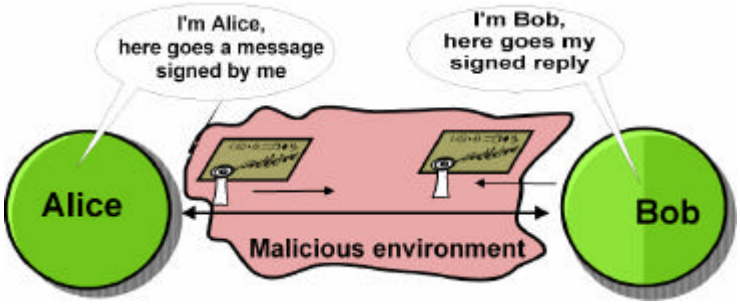
- Trustworthy subsystem – Distributed Security Kernel (DSK) (e.g. appliance boards interconnected by dedicated network)
- Time-free, or timed with partial synchrony
- Arbitrary failure environment + (synchronous) DSK
- Hybrid failure protocols
- Example usage: FT transactional protocols requiring timing constraints



3

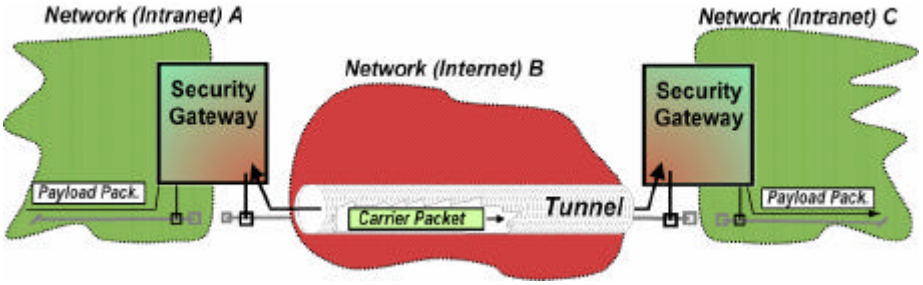
Some paradigms under an IT look

Authentication, signatures, MACs



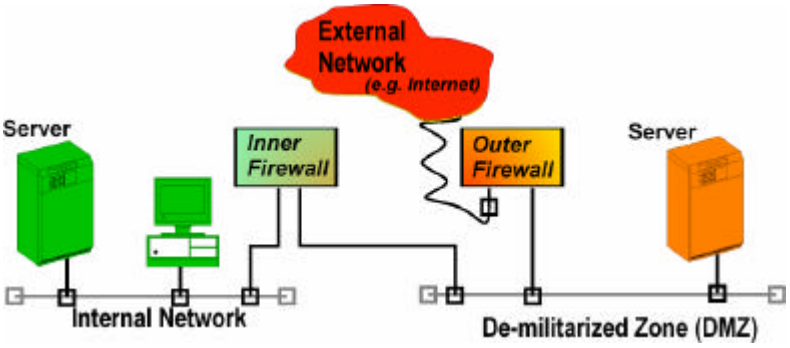
- **Intrusion prevention device: enforces authenticity, integrity**
- **Coverage: signature/authentication method**
- **End-to-end problem: who am I authenticating? me or my PC?**

Tunelling, secure channels



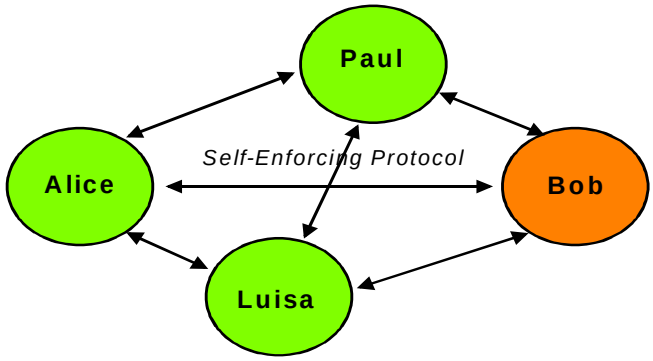
- Intrusion prevention device: enforces confidentiality, integrity (authenticity)
- Coverage: tunelling method, resilience of gateway
- End-to-end problem: are all intranet guys good?

Firewalling



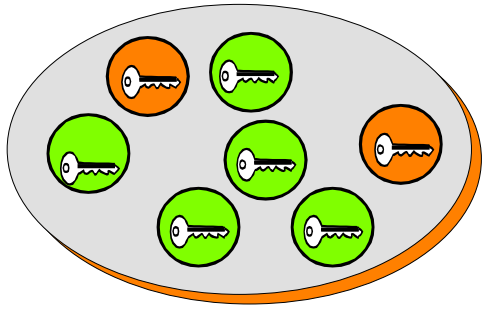
- Intrusion prevention device: prevents attacks on inside machines
- Coverage: semantics of firewall functions, resilience of bastions
- End-to-end problem: are all internal network guys good?

Communication and agreement protocols



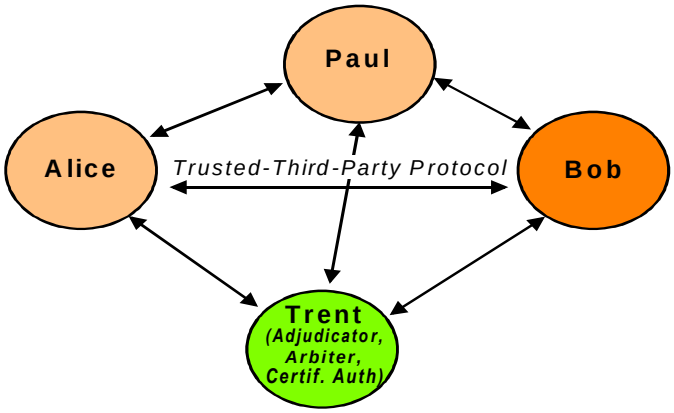
- Intrusion tolerance device: error processing or masking (3f+1, 2f+1, f+2)
- Coverage: semantics of protocol functions, underlying model assumptions

Threshold cryptography



- Intrusion tolerance device: error processing/masking (f+1 out of n)
- Coverage: crypto semantics, brute force resilience, underlying model assumptions

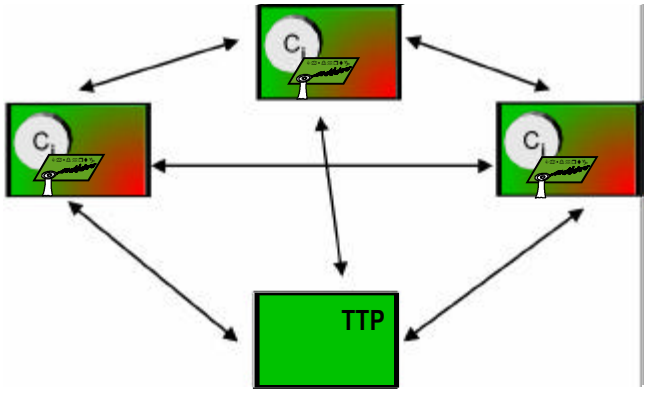
Trusted Third Party (TTP) protocols



- **Intrusion tolerance device: error processing/masking**
- **Coverage: semantics of protocol functions, underlying model assumptions, resilience of TTP**

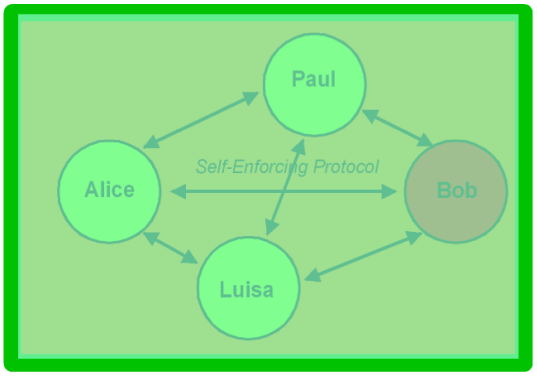
Strategies for construction of IT subsystems

Recursive use of F. Prevention and F.Tolerance



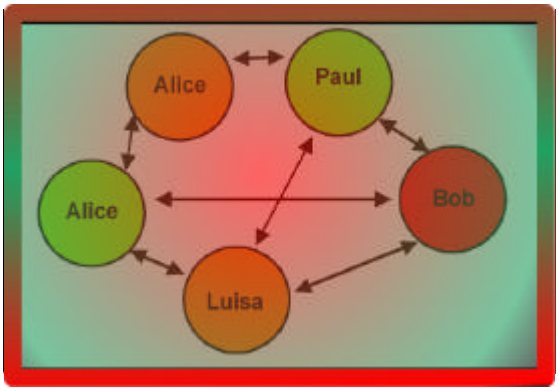
- The TTP protocol revisited
- Work at subsystem level to achieve justifiable behaviour
- Architectural hybridation w.r.t. failure assumptions

Strategies for construction of IT subsystems



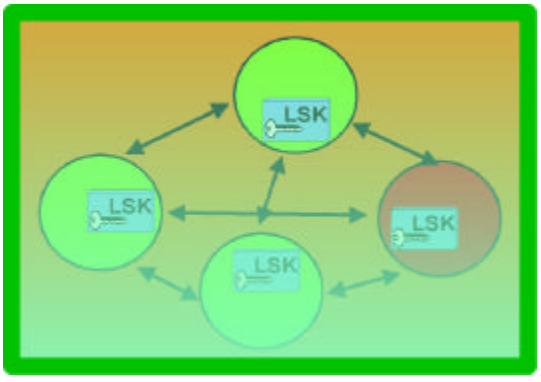
- Arbitrary model – no assumptions
- High coverage – very little to “cover”

Strategies for construction of IT subsystems



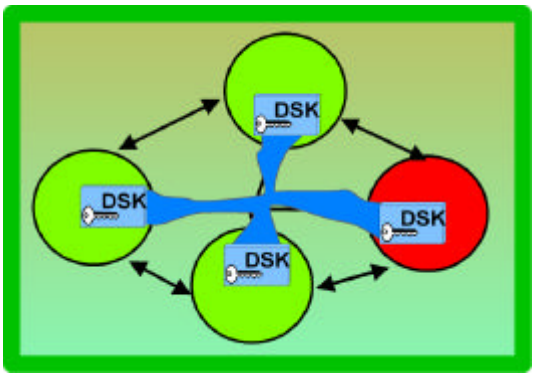
- **Fail-controlled model -- unjustified environment assumptions**
- **Fair coverage – no enforcement**

Strategies for construction of IT subsystems



- **Fail-controlled model – little environment assumptions; justified component assumptions**
- **High coverage – enforcement by Local Security Kernel**

Strategies for construction of IT subsystems

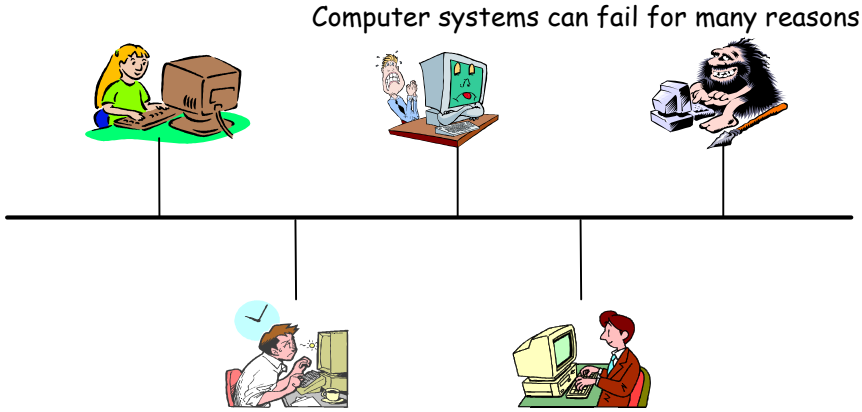


- **Fail-controlled model – little environment assumptions; justified component assumptions**
- **High coverage – enforcement by Distr. Security Kernel**

Example IT projects and systems

(by no means exhaustive, but focusing on works with a strong “IT” brand)

MAFTIA - Malicious and Accidental Fault Tolerance for Internet Applications



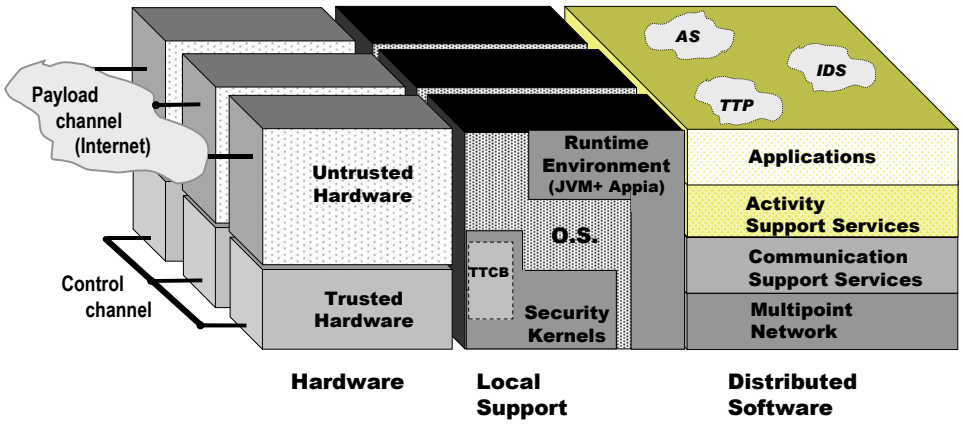
MAFTIA is investigating ways of making computer systems more dependable in the presence of both accidental and malicious faults

Objectives

- **Architectural framework and conceptual model**
- **Mechanisms and protocols:**
  - **dependable middleware**
  - **large scale intrusion detection systems**
  - **dependable trusted third parties**
  - **distributed authorisation mechanisms**
- **Validation and assessment techniques**
- **Partners**
  - **DERA/Qinetiq, Malvern (UK) – Tom McCutcheon / Sadie Creese**
  - **IBM, Zurich (CH) – Marc Dacier / Michael Waidner**
  - **LAAS-CNRS, Toulouse (F) – Y. Deswarte / D. Powell**
  - **Newcastle University (UK)(Coord.) R. Stroud / Brian Randell**
  - **Universität des Saarlandes (D) – Michael Steiner**
  - **Universidade de Lisboa (P) - Paulo Veríssimo / Nuno F. Neves**
- **EU coordinator – Andrea Servida**

http://www.research.ec.org/maftia

Architecture Overview
Host architecture



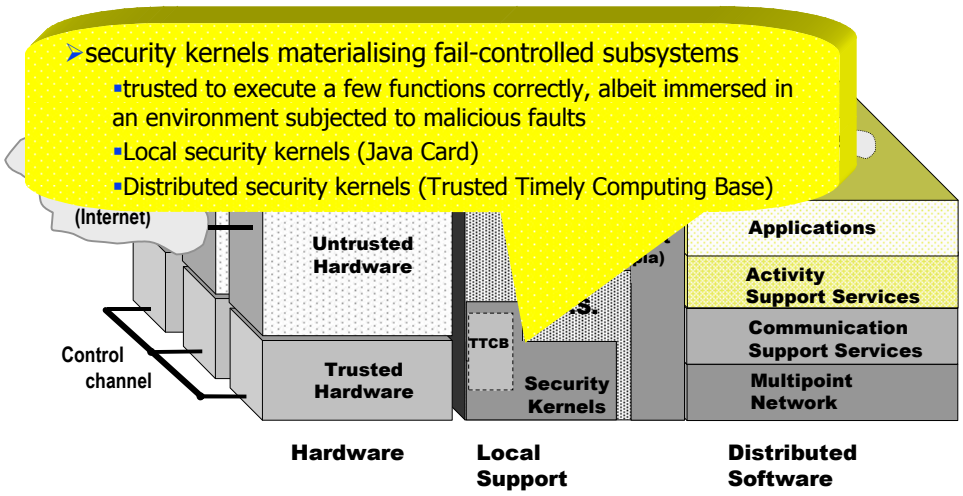
AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service

Architecture Overview
Host architecture

- trusted— vs. untrusted— hardware
  - most of MAFTIA's hardware is untrusted, but small parts considered trusted in the sense of tamperproof by construction

AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service

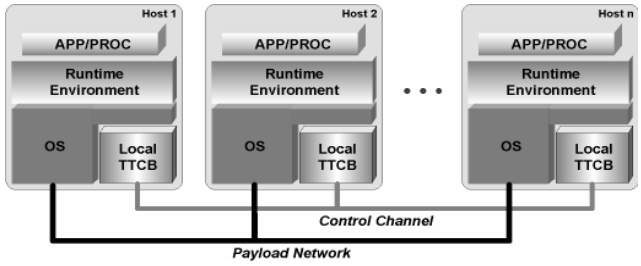
Architecture Overview
Host architecture



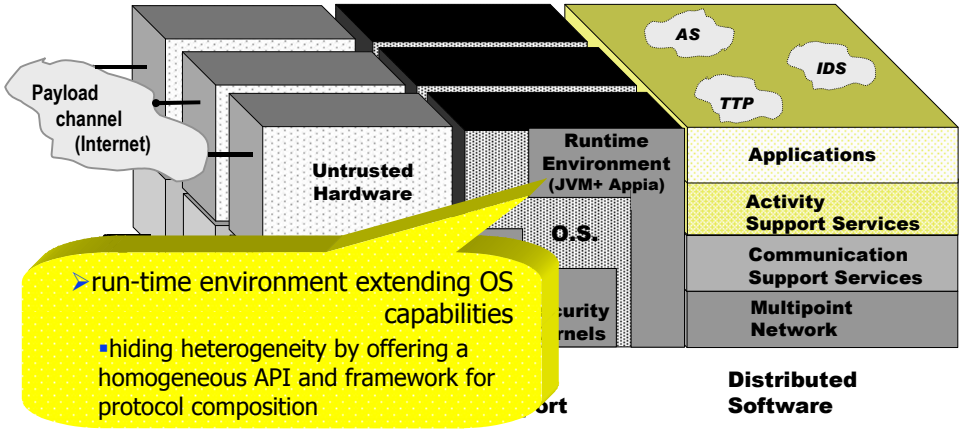
AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service

Trusted Timely Computing Base DSK

- **TTCB is a distributed security kernel that provides a minimal set of trusted and timely services**
- **Construction principles: interposition, shielding, validation**
- **Classic Trusted Computing Base aims at *fault prevention*, while the TTCB aims at *fault tolerance***
- **TTCB can be a: *special hardware module* (e.g. tamperproof device); *secure real-time microkernel* running on a workstation or PC underneath the OS**
- **TTCB control channel has to be both timely and secure: *virtual network* with predictable characteristics coexisting with the payload channel; *separate physical network***

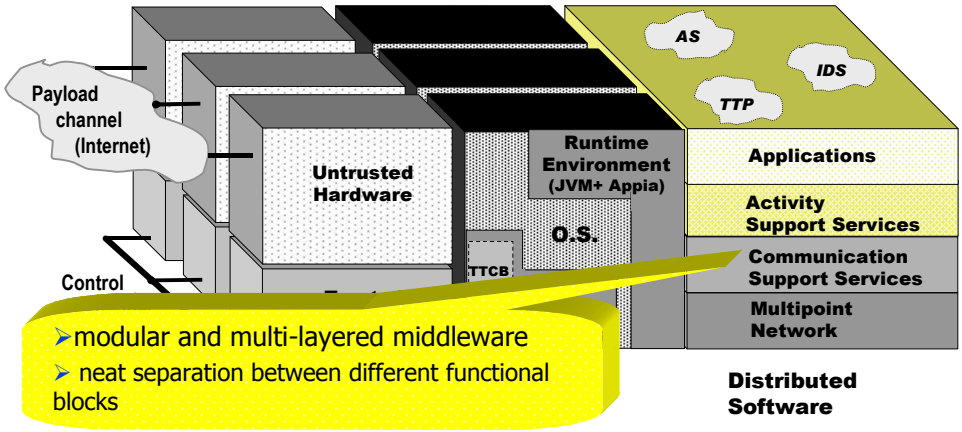


Architecture Overview
Host architecture



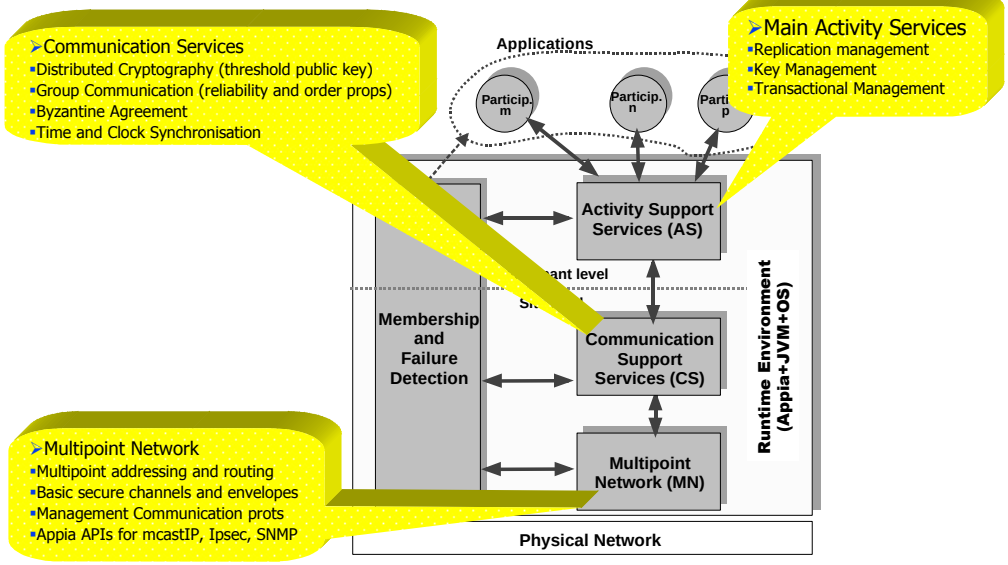
AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service

Architecture Overview
Host architecture



AS - Authorisation Service, IDS - Intrusion Detection Service, TTP - Trusted Third Party Service

Modular Group Architecture

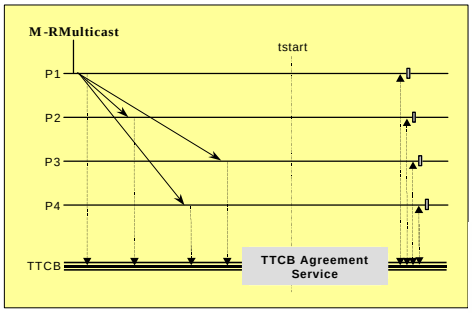


Group Communication on Asynchronous model

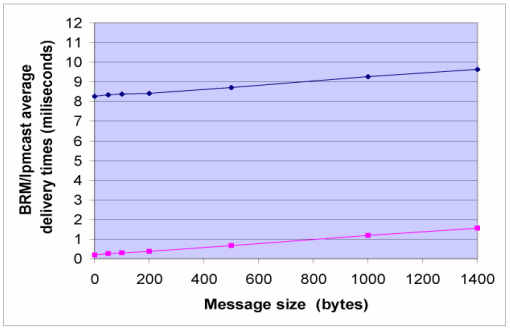
- **Stack of protocols for (among other applications) intrusion-tolerant replicated servers on an asynchronous wide-area setting**
- **Main characteristics of the model: asynchronous; static and open groups; up to n/3 corrupted processes (f < n/3); threshold crypto; manual and trusted key distribution**

Secure Causal Atomic Broadcast	*provides confidentiality*
Atomic Broadcast	*ordering of client requests*
Multi-value Agreement	*agreement on request values*
Broadcast Primitives / Binary Agreement	*reliable and consistent broadcasts, agreement on Y/N questions*
Threshold Cryptography	*threshold cryptosystem and digital signature scheme*
Reliable Point-to-Point Links	

Byzantine-Reliable Multicast on Timed Model with TTCB



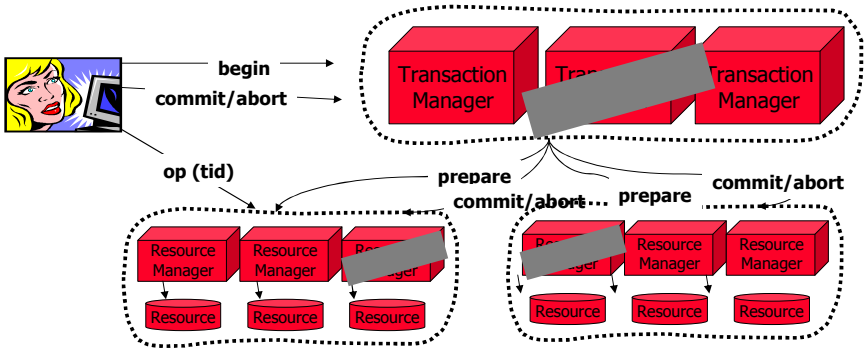
Byzantine Reliable Multicast Protocol (1 Phase)



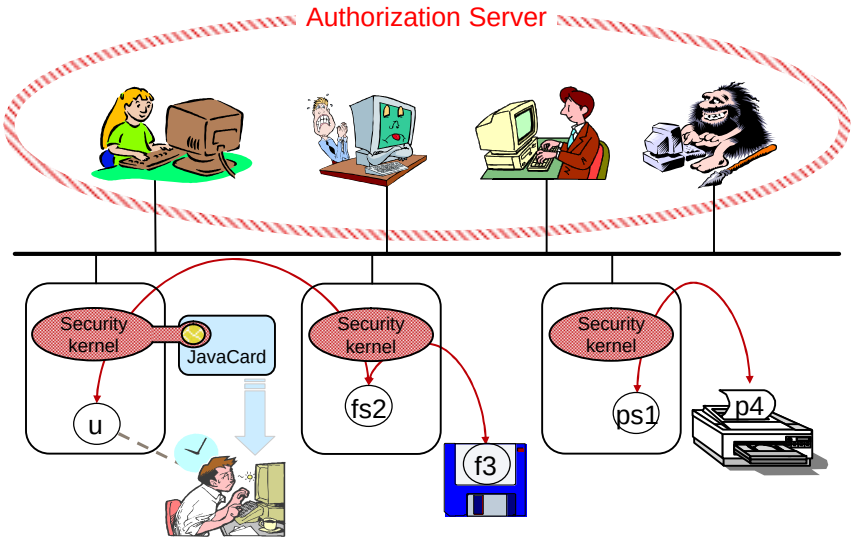
5-Node Delivery Times

IT Transactions with Error Masking

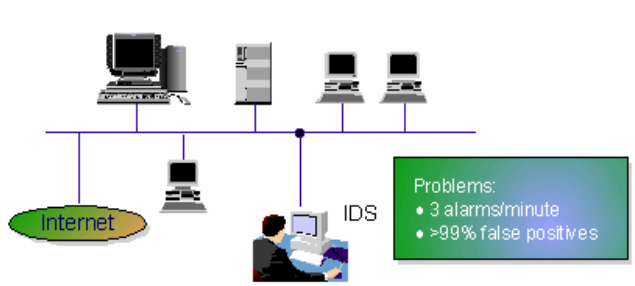
- **A CORBA-style transaction service, standard ACID properties**
- **Support for multiparty transactions**
- **Uses error masking to tolerate intrusions**
- **Application of hybrid failure assumptions**



IT Authorisation Service

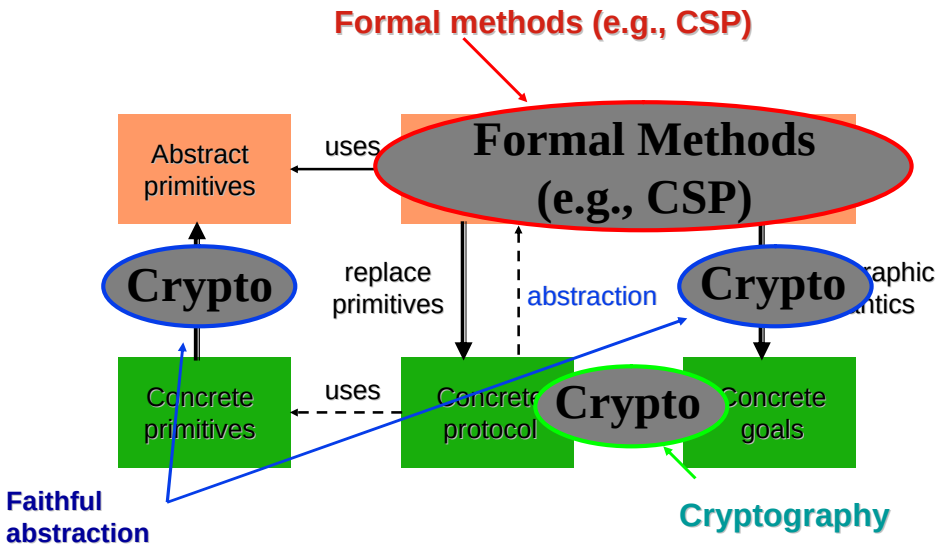


IT Intrusion Detection Service



- **finding solutions to the problems of the high rate of false positive and false negative alarms generated by existing solutions**
- **these false alarms can also be due to attacks against the IDS itself, therefore the need to design an IDS which is itself tolerant to intrusions**
- **study and evaluate how notions such as fault injection, diversity and distributed reasoning can address the weaknesses of existing solutions**

Verifying MAFTIA protocols

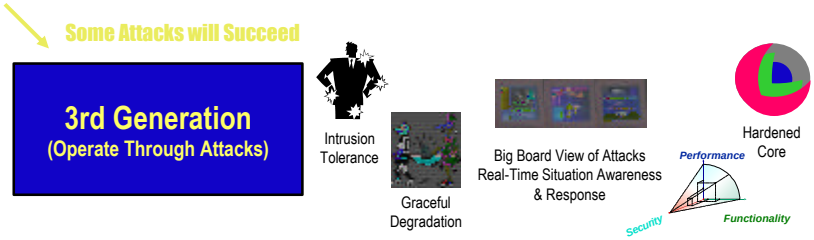


DARPA

OASIS

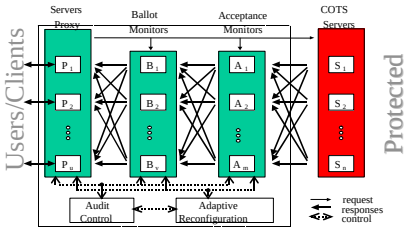
ORGANICALLY ASSURED & SURVIVABLE INFORMATION SYSTEMS

Dr. Jaynarayan Lala – jlala@darpa.mil, 703-696-7441
Organically Assured Survivable Information Systems,
OASIS Demonstration and Validation Program



OASIS

Intrusion Tolerant Architecture



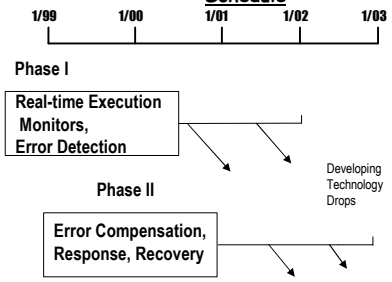
Objectives

- Construct intrusion-tolerant architectures from potentially vulnerable components
- Characterize cost-benefits of intrusion tolerance mechanisms
- Develop assessment and validation methodologies to evaluate intrusion tolerance mechanisms

Technical Approach

- Real-Time Execution Monitors: In-line reference monitors, wrappers, sandboxing, binary insertion in legacy code, proof carrying code, secure mobile protocols
- Error Detection & Tolerance Triggers: Time and Value Domain Checks, Comparison and Voting, Rear Guards
- Error Compensation, Response and Recovery: Hardware and Software Redundancy, Rollback and Roll-Forward Recovery
- Intrusion Tolerant Architectures: Design Diversity, Randomness, Uncertainty, Agility
- Assessment & Validation: Peer Review Teams, Red Team, Assurance Case (Fault Tree, Hazard Analysis, Formal Proofs, Analytical Models, Empirical Evidence)

Schedule



OASIS Projects

Project Title	Organization	PI
Scaling Proof-Carrying Code to Production Compilers and Security Policies	Princeton University	Andrew Appel
Sandboxing Mobile Code Execution Environments	Cigital	Anup Ghosh
Containment and Integrity for Mobile Code	Cornell University	Fred Schneider
Integrity Through Mediated Interfaces	Teknowledge	Bob Balzer
Agile Objects: Component-based Inherent Survivability	UC, San Diego	Andrew Chien
A Distributed Framework for Perpetually Available and Secure Information Systems	CMU	Pradeep Khosla
New Approaches to Mobile Code: Reconciling Execution Efficiency with Provable Security	UC, Irvine	Michael Franz
A Binary Agent Technology for COTS Software Integrity	InCert Software Corp.	Anant Agarwal
Self-Protecting Mobile Agents	NAI Labs	Lee Badger
Intrusion Tolerant Software Architectures	SRI International	Victoria Stavridou
Computational Resiliency	Syracuse University	Steve Chapin
Intrusion Tolerance Using Masking, Redundancy and Dispersion	Draper Laboratory	Janet Lepanto
Dependable Intrusion Tolerance	SRI International	Alfonso Valdes
Intrusion Tolerant Distributed Object Systems	NAI Labs	Gregg Tally
Hierarchical Adaptive Control for QoS	Teknowledge	Jim Just
Intrusion Tolerant Server Infrastructure	Secure Computing Corp	Dick O'Brien
Randomized Failover Intrusion Tolerant Systems	Architecture Technology Corp	Ranga Ramanujan
A Comprehensive Approach for Intrusion Tolerance Based on Intelligent Compensating Middleware	Telcordia	Amjad Umar

OASIS Projects

Project Title	Organization	PI
Engineering a Distributed Intrusion Tolerant Database System Using COTS Components	University of Maryland, Baltimore County	Peng Liu
Intrusion Tolerance by Unpredictable Adaptation	BBN Technologies	Partha Pal, Bill Sanders
SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services	MCNC, Duke University	Fengmin Gong, Kishor Trivedi
Tolerating Intrusions Through Secure System Reconfiguration	University of Colorado	Alexander Wolf
Active Trust Management for Automonous Adaptive Survivable Systems	MIT	Howie Shrobe
Enterprise Wrappers (NT)	Teknowledge	Bob Balzer
Enterprise Wrappers (Unix)	NAI Labs	Mark Feldman
Information Assurance Science and Engineering Project	CMU, SEI	Tom Longstaff
Autonomix: Component, Network and System Autonomy	WireX Communications, Inc.	Crispin Cowan
An Aspect-Oriented Security Assurance Solution	Cigital	Tim Hollebeek
Dependence Graphs for Information Assurance of Systems	Grammatech, Inc	Tim Teitelbaum
Cyberscience	SRI International	Victoria Stavridou

OASIS technology groups

- **Possible grouping by primary focus, for example**

How to develop programs/systems With fewer vulnerabilities?

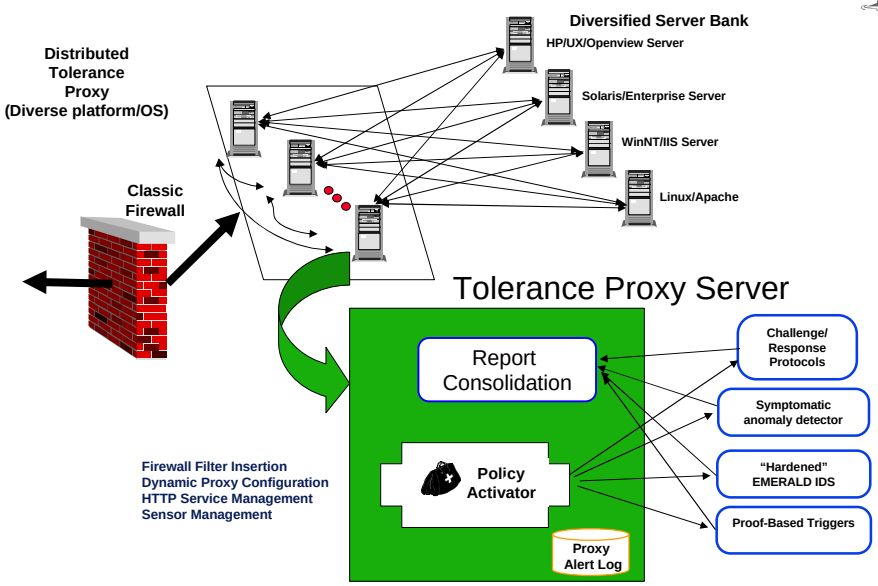
How to organize intrusion tolerant subsystems?

How to detect/resist/tolerate component vulnerabilities?

How to understand behavior of intrusion tolerant systems?

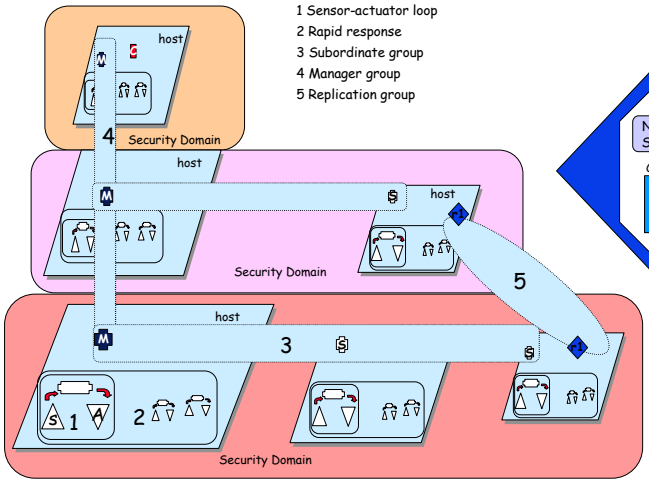
ALL THESE LISTS ARE INCOMPLETE

Intrusion Tolerance by Dependable Proxy Server

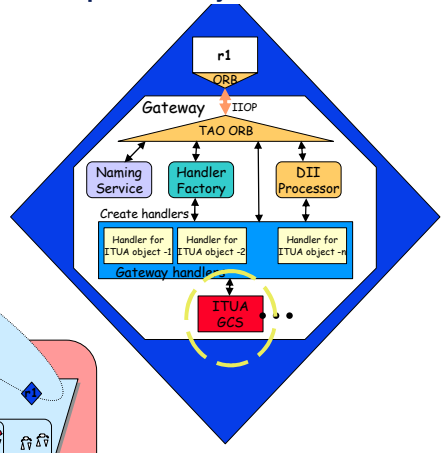


ITUA: Intrusion Tolerance by Unpredictability and Adaptation

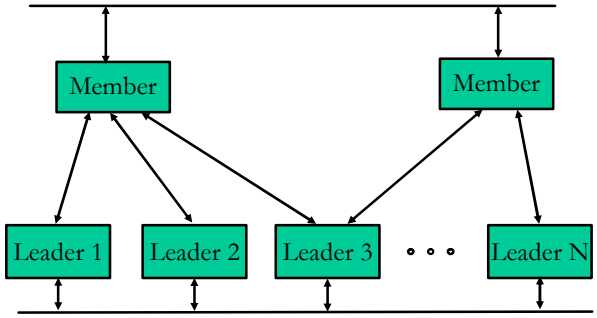
General Architecture



Replicated Object Architecture

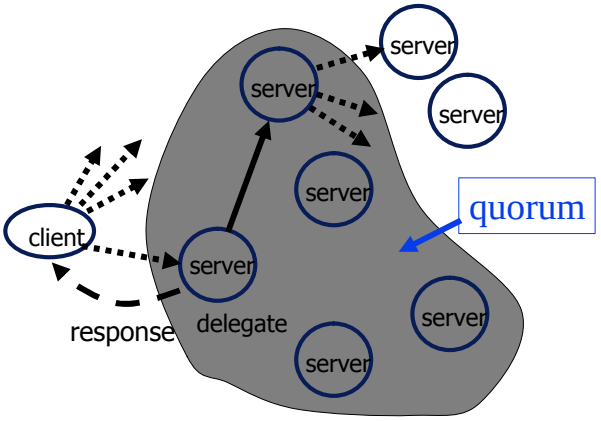


Enclaves: IT support for group collaboration



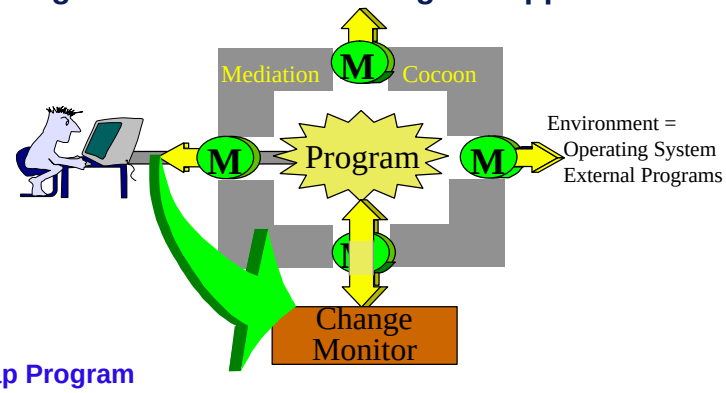
- **Middleware for secure group applications in insecure nets (Internet)**
- **Lightweight, software-only implementation (currently Java)**
- **Services provided:**
  - **Secure group multicast (confidentiality and integrity: encryption with common group key)**
  - **Group management: user authentication; join and leave protocols; group-key generation, distribution, refresh**

COCA C.A. System Architecture



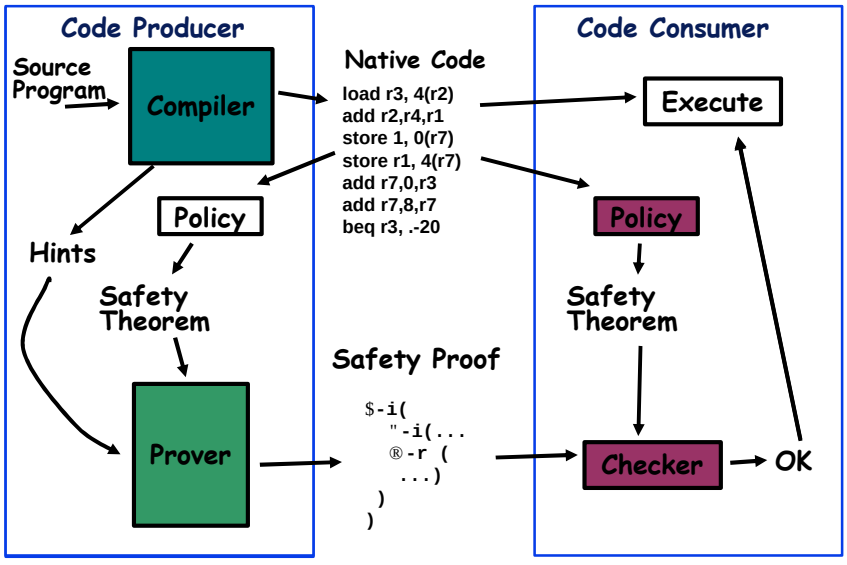
server failure
↓ dissem. Byzantine Quorum
server compromise
↓ threshold signature protocol
mobile attack
↓ proactive secret sharing (PSS)
asynchrony
↓ asynchronous PSS

Hardening COTS Products Through Wrapper Mediation

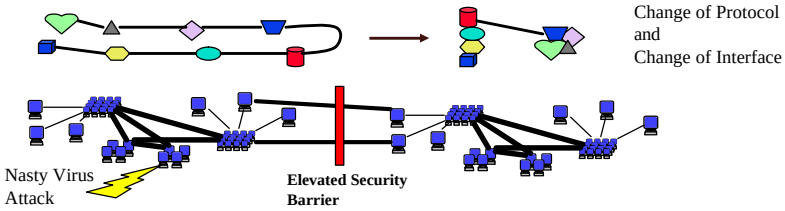


- **Wrap Program**
  - Detect access of integrity marked data & decode it
  - Monitor User Interface to detect change actions
    - » Translate GUI actions into application specific modifications
  - Detect update of integrity marked data
    - » Re-encode & re-integrity mark the updated data
- **Repair any subsequent Corruption from History**
- **Build on existing research infrastructure**

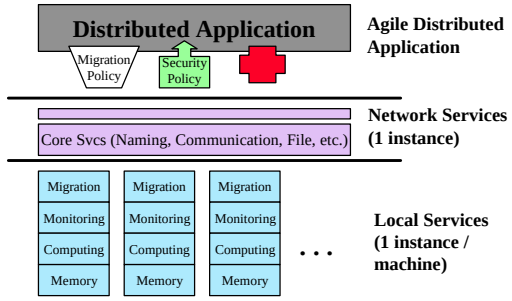
Scaling Proof-Carrying Code



Agile Objects and Elusive Interfaces

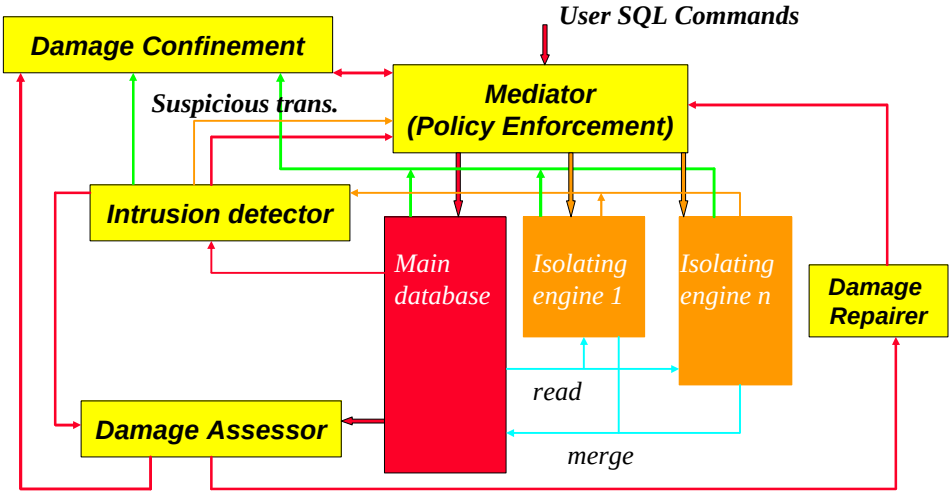


- **Integrated security mechanisms with high performance RPC/distributed objects (Elusive Interfaces)**
  - Exploit computer manipulable interfaces and data reorganization
- **Adaptive security management for Agile, highly decentralized applications**
  - Rapidly and continuously changing environment and configurations



Distributed IT Database System Using COTS Components

Simple intrusion tolerance, multi-phase confinement, isolation



Further Reading

- L. Alvisi, D. Malkhi, E. Pierce, M. K. Reiter, and R. N. Wright, Dynamic Byzantine quorum systems," in Proc. Int'l Conference on Dependable Sys and Networks (FTCS-30/DCCA-8), pp. 283-292, 2000.
- Y. Amir et al. Secure group communication in asynchronous networks with failures: Integration and experiments. In Proc. The 20th IEEE International Conference on Distributed Computing Systems (ICDCS 2000), pages 330-343, Taipei, Taiwan, April 2000.
- G. Ateniese, M. Steiner, G. Tsudik: Authenticated group key agreement and friends. In Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS-98), pages 17-26, New York, November 3-5 1998. ACM Press.
- Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. New multi-party authentication services and key agreement protocols. IEEE Journal of Selected Areas on Communications, 18, March 2000.
- Christian Cachin. Distributing Trust on the Internet. In Procs. of the Int'l Conf. on Depend. Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.
- C. Cachin, K. Kursawe and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography", in Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC), pp.123-32, 2000b.
- C. Cachin and J. A. Poritz, Hydra: Secure replication on the Internet," In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- M. Castro and B. Liskov, Practical Byzantine fault tolerance," in Proc. Third Symp. Operating Systems Design and Implementation (OSDI), 1999.
- T. D. Chandra and S. Toueg, Unreliable failure detectors for reliable distributed systems," Journal of the ACM, vol. 43, no. 2, pp. 225-267, 1996.
- Nick Cook, Santosh Shrivastava, Stuart Wheater. Distributed Object Middleware to Support Dependable Information Sharing between Organisations. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.

Further Reading

- M. Correia, Lau Cheuk Lung, Nuno Ferreira Neves, and P. Veríssimo. Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model. In Proc. of Symp. of Reliable Distributed Systems, October 2002, Japan.
- Miguel Correia, Paulo Veríssimo, and Nuno~Ferreira Neves. The architecture of a secure group communication system based on intrusion tolerance. In International Workshop on Applied Reliable Group Communication, Phoenix, Arizona, USA, April 2001.
- M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In proceedings of the EDCC-4, Fourth European Dependable Computing Conference, Toulouse, France - October 23-25, 2002.
- H. Debar, M. Dacier, A. Wespi: Towards a taxonomy of intrusion detection systems. Computer Networks, 31:805-822, 1999.
- Y. Desmedt: Society and group oriented cryptography: a new concept; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 120-127.
- Y. Desmedt, Threshold cryptography," European Transactions on Telecommunications, vol. 5, no. 4, pp. 449-457, 1994.
- Y. Deswarte, N. Abghour, V. Nicomette and D. Powell, "An internet authorization scheme using smart card-based security kernels", in Int'l Conf. on Research in Smart Cards (E-smart 2001), (Cannes, France), Lecture Notes in Computer Science, pp.71-82, Springer-Verlag, 2001.
- Y. Deswarte, L. Blain, J.-C. Fabre: Intrusion tolerance in distributed systems. In Proc. Symp. on Research in Security and Privacy, pages 110-121, Oakland, CA, USA, 1991. IEEE CompSoc Press.
- Durward McDonell, Brian Niebuhr, Brian Matt, David L. Sames, Gregg Tally, Szu-Chien Wang, Brent Whitmore. Developing a Heterogeneous Intrusion Tolerant CORBA System. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.

Further Reading

- Bruno Dutertre, Hassen Saïdi and Victoria Stavridou. Intrusion-Tolerant Group Management in Enclaves. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2001), Gotteborg, Sweden, 2001.
- J. Fraga and D. Powell, "A Fault and Intrusion-Tolerant File System", in IFIP 3rd Int. Conf. on Computer Security, (J. B. Grimson and H.-J. Kugler, Eds.), (Dublin, Ireland), Computer Security, pp.203-18, Elsevier Science Publishers B.V. (North-Holland), 1985.
- R. Guerraoui, M. Hurn, A. Mostefaoui, R. Oliveira, M. Raynal, and A. Schiper, Consensus in asynchronous distributed systems: A concise guided tour," in Advances in Distributed Systems (S. Krakowiak and S. Shrivastava, eds.), vol. 1752 of LNCS, pp. 33-47, Springer, 2000.
- V. Hadzilacos and S. Toueg, Fault-tolerant broadcasts and related problems," in Distributed Systems (S. J. Mullender, ed.), New York: ACM Press & Addison-Wesley, 1993. An expanded version as Technical Report TR94-1425, Department of Computer Science, Cornell University, Ithaca NY, 1994.
- HariGovind V Ramasamy, Prashant Pandey, James Lyons, Michel Cukier, William H. Sanders. Quantifying the Cost of Providing Intrusion Tolerance in Group Communication Systems, In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- Matti A. Hiltunen, Richard D. Schlichting and Carlos A. Ugarte. Enhancing Survivability of Security Services Using Redundancy. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.
- K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, The SecureRing protocolsfor securing group communication," in Proc. 31st Hawaii Int'l Conf. on System Sciences, pp. 317-326, IEEE, Jan. 1998.
- J. H. Lala, "A Byzantine Resilient Fault-Tolerant Computer for Nuclear Power Plant Applications", in 16th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-16), (Vienna, Austria), pp.338-43, IEEE Computer Society Press, 1986.

Further Reading

- B. Madan, K. Goseva-Popstojanova, K. Vaidyanathan, K. Trivedi. Modeling and Quantification of Security Attributes of Software Systems. In Procs. of the Int'l Conf. on Dep. Syst. and Networks (DSN-2002), Washigton, USA, 2002.
- D. Malkhi and M. K. Reiter, An architecture for survivable coordination in large distributed systems," IEEE Transactions on Knowledge and Data Engineering, vol. 12, no. 2, pp. 187-202, 2000.
- Jean-Philippe Martin, Lorenzo Alvisi, Michael Dahlin. Small Byzantine Quorums. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Washigton, USA, 2002.
- Roy A. Maxion and Tahlia N. Townsen, Masquerade Detection Using Truncated Command Lines. In Procs. of the Int'l Conf. on Dep. Syst. and Networks (DSN-2002), Washigton, USA, 2002.
- F. Meyer and D. Pradhan, "Consensus with Dual Failure Modes," presented at The 17th International Symposium on Fault-Tolerant Computing Systems, Pittsburgh, PA, 1987, pp. 214--22.
- L. E. Moser, P. M. Melliar-Smith, and N. Narasimhan. The SecureGroup communication system. In Proceedings of the IEEE Information Survivability Conference, pages 507–516, January 2000.
- Peter G. Neumann, "Practical Architectures for Survivable Systems and Networks," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report http://www.csl.sri.com/~neumann/private/arldraft.{pdf|ps}, October 1998.
- Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. 7th ACM Conference on Computer and Communications Security, Athens, November 2000, ACM Press, New York 2000, 245-254.
- P. Porras, D. Schnackenberg, S. Staniford-Chen and M. Stillman, "The Common Intrusion Detection Framework Architecture", CIDF working group, http://www.gidos.org/drafts/architecture.txt, (accessed: 5 September, 2001).

Further Reading

- D. Powell, G. Bonn, D. Seaton, P. Veríssimo and F. Waeselynck, "The Delta-4 Approach to Dependability in Open Distributed Computing Systems", in 18th IEEE Int. Symp. on Fault-Tolerant Computing Systems (FTCS-18), (Tokyo, Japan), pp.246-51, IEEE Computer Society Press, 1988.
- M. K. Reiter: Distributing trust with the Rampart toolkit; Communications of the ACM, 39/4 (1996).
- F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial", ACM Computing Surveys, 22 (4), pp.299-319, 1990.
- P. Verissimo, A. Casimiro and C. Fetzer, "The Timely Computing Base: Timely Actions in the Presence of Uncertain Timeliness", in Proc. of DSN 2000, the Int. Conf. on Dependable Systems and Networks, pp.533-52, IEEE/IFIP, 2000.
- Paulo Veríssimo, Nuno~Ferreira Neves, and Miguel Correia. The middleware architecture of MAFTIA: A blueprint. In Proceedings of the IEEE Third Information Survivability Workshop (ISW-2000), Boston, Massachusetts, USA, October 2000.
- Chenxi Wang, Jack Davidson, Jonathan Hill and John Knight. Protection of Software-Based Survivability Mechanisms. In Procs. of the Int'l Conf. on Dependable Systems and Networks (DSN-2002), Gotteborg, Sweden, 2001.
- J.~Xu, A.~Romanovsky, and B.~Randell. Concurrent exception handling and resolution in distributed object systems. IEEE Trans. on Parallel and Distributed Systems, 10(11):1019--1032, 2000.
- L. Zhou, F. B. Schneider, and R. van Renesse, COCA: A secure distributed online certification authority,Tech. Rep. 2000-1828, CS Dpt, Cornell University, Dec. 2000. Also ACM TOCS to appear.

Where to find us

- ***Navigators Group***

  http://www.navigators.di.fc.ul.pt

- **IT related research in the site:**
- *Look-up our research in the Fault and Intrusion Tolerance in Open Distributed Systems research line*
- *Look-up recent papers authored by Veríssimo and Correia, and/or referencing the MAFTIA project*

- **Feel free to email**
- **Paulo:** *pjv@di.fc.ul.pt*