Intrusion tolerant systems

Anders Fongen

Forsvarets forskningsinstitutt/Norwegian Defence Research Establishment (FFI)

11 December 2007

FFI-rapport 2007/02611

1070

ISBN 978-82-464-1298-6

Keywords

Inntrengingstolerante systemer

Feiltolerans

Innbruddsdeteksjon

Datasikkerhet

Approved by

Eli Winjum	Project manager
Vidar S. Andersen	Director

English summary

Security intrusions and successful attacks on computer systems will occur regardless of the quality of the control and protection systems in use. It is therefore necessary to build computer systems that offer essential services even in the presence of a successful attack. Such systems are called *intrusion tolerant.*

Intrusion tolerant systems differ from *fault tolerant systems* by their threat models. Fault tolerant systems are designed to survive spontaneous errors (due to natural physical processes), whereas intrusion tolerant systems should withstand attacks from skilled, well informed and resourceful adversaries who would launch multi-stage attacks on the system, where also the detection and recovery mechanisms are targeted. Spontaneous errors may be statistically modeled, whereas a targeted attack cannot.

The research on intrusion tolerant systems draws on knowledge and experience from several other research fields, i.a. computing security, distributed systems and fault tolerant systems. These fields bring with them slightly different perspectives into the research, which will be presented in the report.

The construction of intrusion tolerant systems builds on top of a range of well known technologies from related research: Intrusion detection, cryptography, distributed recovery, system diversity etc. These contributions will be presented in the report and their contribution to intrusion tolerant systems will be identified.

There are no intrusion tolerant systems in the sense that they defend themselves against attacks and misuse under any circumstances. What can be found are attempts to combine existing techniques for intrusion detection, cryptography, crash recovery and damage mitigation into frameworks which create a stronger defence than if these techniques were applied separately. The report presents a few of these frameworks.

The report also points to the reasons why mobile, tactical systems are more difficult to turn into intrusion tolerant systems. A few research questions on this matter are suggested.

The report does not make a distinction between *intrusion* and *attack.* Someone may argue that misuse by disloyal employee should not be called intrusion, but this distinction is not made.

Sammendrag

Sikkerhetsinnbrudd og angrep på datamaskiner vil skje, uansett hvor gode kontroll- og beskyttelsessystemer som brukes. Derfor er det nødvendig å bygge datasystemene slik at de leverer essensielle tjenester også i tilfelle et vellykket angrep. Slike systemer kalles *inntrengningstolerante*.

Inntrengingstolerante systemer skiller seg fra *feiltolerante* systemer ved deres trusselbilder. Mens feiltrolerante systemer skal motstå spontane feil (f.eks. knyttet til naturlige fysiske prosesser), skal inntrengningstolerante systemer motstå målrettede angrep fra kunnskapsrike og ressurssterke aktører som kan skape flertrinns angrep rettet også mot deteksjons- og gjenopprettings-mekanismene. Mens spontane feil kan modelleres statistisk, kan målrettede angrep ikke det.

Forskningen på inntrengningstolerante systemer henter kunnskap og erfaringer fra flere andre felt, bl.a. datasikkerhet, distribuerte systemer og feiltolerante systemer. Disse feltene tar med seg litt ulike perspektiver inn i forskningen, og disse perspektivene blir presentert i rapporten.

Konstruksjonen av inntrengningstolerante systemer bygger på en rekke velkjente teknologier fra relatert forskning: Inntrengingsdeteksjon, kryptografi, distribuert gjenoppretting, diversifiserte systemer m.m.. Disse bidragene blir presentert i rapporten og deres bidrag til inntrengningstolerante systemer identifisert.

Det finnes ikke inntrengingstolerante systemer i den forstand at de forsvarer seg mot inntrengning og misbruk i alle situasjoner. Det som finnes er forsøk på å kombinere eksisterende teknikker for inntrengningsdeteksjon, kryptografi, gjenoppretting og skadebegrensning i rammeverk som skaper et sterkere vern enn om teknikkene ble anvendt separat. Rapporten beskriver kort noen slike rammeverkprosjekter.

Rapporten peker også på de særlige egenskaper ved mobile, taktiske nettverk som gjør det vanskeligere å gjøre dem inntrengningstolerante. Noen utkast til forskningsspørsmål på dette feltet blir presentert.

Rapporten gjør ikke forskjell på begrepene *inntrengning* og *angrep*. Noen vil kanskje hevde at angrep utført av betrodde innsidere ikke bør kalles inntrengning, men en slik nyansering er altså ikke gjort.

Contents

1 Introduction

The recent interest in the research field of *Intrusion Tolerance* is founded on the assumptions that a security perimeter will ultimately fail. The number of computers connected to a network, the range of tools available to an intruder and the hard pressed time-to-market for commercial software are factors that increase the probability for successful intrusions.
Realizing this fact, it is the goal of computing security scientists to provide systems that persist to offer correct and essential services even in the presence of successful compromise, or offer controlled recovery mechanisms and log data for forensic investigation. This is a formidable task that needs to draw on expertise and experience from several research fields; intrusion detection, fault tolerance, distributed systems, operating systems and programming languages.
It is the intention of this report to present the foundations and the current state of research on intrusion tolerance. A number of approaches will be presented, together with the associated frameworks, methodologies and experimental results.

The rest of the report is organized as follows: *Chapter 2* provides the motivational background for the research on intrusion tolerance. *Chapter 3* explains a list of terms important to the report. *Chapter 4* introduces a threat model, which is intended as a framework for analyzing threats and risks. *Chapter 5* views the field of intrusion tolerance from the perspective of other related research fields: Fault tolerance, distributed systems and computing security. *Chapter 6* identifies a selection of technologies which can be used when building intrusion tolerant systems, and discusses their contributions. *Chapter 7* gives a short presentation of intrusion tolerant research projects. *Chapter 8* gives a brief discussion on the issue of intrusion tolerance in mobile systems. *Chapter 9* gives a summary of the report and suggests some research questions which could be pursued by FFI.

2 The motivation for Intrusion Tolerance

Despite the focus on computer security and the available technology for protection, the threats to computer systems appear to be a growing problem. Some of the reasons for this are:

- A growing number of computers is connected to the Internet
- A growing number of computers is using wireless network technology
- A growing audience on the internet is likely to contain a larger number of potential intruders
- An increased availability of ready-to-use software tools for intrusion and other computer crimes allows anyone to attempt attacks, not only experts
- The time-to-market pressure of commercial software products leaves less time to quality control and security testing. Discovered security problems are fixed in regular updates, leaving a range of uncorrected security holes in many software products.
- Users are increasingly often involved in E-business and E-finance operations on open networks, and are exposed to a growing number of authentication mechanisms (involving

secret passwords). Sensible information (in the sense that it can be stolen and used for criminal purposes) is increasingly often sent over an open network.

A security perimeter, in the form of virus control, network firewall, authentication and authorization framework, will anticipate attack and prevent them based on the observable difference between regular and irregular computer activity:

- A firewall typically allows any outgoing TCP connection (from the ”safe” local network to the ”hostile” Internet) based on the assumption that these connections are initiated and controlled by benign activities.
- A virus protection program scans computer files, received e-mail and web traffic for bit patterns known to characterize known viruses. Bit patterns not recognized are assumed to be ”friendly”.
- An authorization framework will assume the identity of a user based on information not possessed by others (e.g. a password) and restrict the privileges granted to the user based on an authorization scheme.
- An intrusion detection system scans network activities and monitors internal computer processes with the purpose of detecting patterns of hostile activities.

All these protection systems assume that hostile activities have distinct characteristics which make them detectable in a complete and precise manner. This is not the case today, however, and is unlikely to be the case in the future. The stronger and more successful these security precautions are, the more annoying and counterproductive will they be, as they block or delay legal and productive activities. In order not to obstruct normal production, security protection mechanisms must allow a calculated risk for a successful attack.

The term ”security perimeter” indicates that once it has been compromised, it offers little resistance to further exploitation of the system. Some examples are:

- Once a virus has started its execution on a computer with Microsoft Windows, it can take full control over the computer, violating the integrity of communication, computation and storage, and install camouflage to remain undetected.[1]
- A trojan that has been started on a Unix computer may do anything the compromised user is allowed to do: Altering user files, sending E-mail or Web requests, starting background jobs and network servers etc.
- An incorrectly installed program on a Unix computer (e.g. leaving a ”set UID” root program in a writable state) leaves the computer vulnerable to total compromise.
- A compromised computer may be a useful launchpad for attacks on other computers on the same local network, since the security perimeter between these computers may be weak or even absent.

[1] Requires that the Windows user has administrative privileges, which is often observed in practice

Based on these observations, there appears to be a need for security measures that deal with successful attacks and compromised computers. These measures would deal with such issues as:

- Intrusion detection
- Eradication of the attack
- Damage mitigation
- Damage assessment
- Recovery to valid and consistent state
- Log analysis for forensic purposes
- Re-training of security mechanisms based on gained experience

The term *Intrusion Tolerance* is founded on the observation that successful attacks occur despite the presence of protection mechanisms. Intrusion tolerance refers to research which aims at *maintaining essential services at acceptable level in the presence of hostile system compromise.* We propose four keywords to describe the phases/activities in an intrusion tolerant system:

Detect – Limit – Recover – Learn

We will use these four keywords in the following discussions on elements and techniques within intrusion tolerance.

3 Definition of terms

ACID properties	The traditional requirements of a transaction is that it appears to be indivisible (**A**tomic), that it leaves the system in a **C**onsistent state, that it exposes no intermediary results (**I**solated) and leaves all result on permanent storage (**D**urable). Together, these properties make the acronym **ACID**.
Byzantine error	An error condition where a process can behave arbitrarily (including respond with apparently correct answers). It is difficult to detect nodes with this error condition (often called ”Byzantine nodes”)
COTS	”Commercial Off The Shelf”, describes products available on the general civil market
DNS	”Domain Name Services” the service where an Internet name (e.g. www.ffi.no) is mapped to an IP address.
DoS	”Denial of Service”, an attack on a system with the purpose of making it unavailable.
HIDS and NIDS	”Host-based/Network-based Intrusion Detection Systems”, describes system for automatic detection of intrusion and attacks. Host based systems are software installed in a computer and monitors activities in that computer, whereas network based systems are monitoring network activity on the outside.
Middleware	A layer of software between the operating system and the application. It is supposed to be ”business-unaware” and solve problems common to several

	applications, and offer services which are not commonly offered by the operating system. Example on middleware services are databases, clock synchronization, transactions, replication etc.
PDU	”Protocol Data Unit”, the data exchanged through a communication protocol. The data is formatted according to the rules of the protocol. An IP packet is a PDU of the IP protocol.
SQL	”Structured Query Language”, a standard command language for retrieval and modification of data in a database. SQL is not considered to be a programming language.
SYN flood	A famous type of DoS attack that repeatedly sets up TCP-connections, but fails to comply with the protocol to complete the setup phase. The target computer may allocate so many resources to each ongoing connection that it eventually cannot operate normally.
TCP	”Transport Control Protocol”, a protocol which allows two processes to communicate (i.e. its endpoint is a process, not a computer). TCP builds on the IP protocol.
URL	”Uniform Resource Locator”, a text string on a format which describes the location and access protocol of a resource in an IP network. Typically used by a web browser for finding web pages, e.g. ”http://www.ffi.no/” is a URL.
POTS, ISDN, 3G	Describes different telephone technologies, which can also be used for circuit-switched data communication. POTS (Plain Old Telephone Service) is the old analog telephone system, ISDN is a digital telephone system more modern and advanced than POTS and offers higher data rates. 3G is a digital mobile communication system which offers medium data rates (< 1 Mb/s).

4 Threat model

4.1 Difference from Risk model

In order to construct a framework for intrusion tolerance a *threat model* is required. This is different from *risk models* found in the field of Fault Tolerance, which makes some assumptions about the occurrence of errors and failures:

- Errors and faults occur without any purpose or intent
- The frequency of errors can be statistically modeled, e.g. with a Poisson model or numbers like ”Mean Time Between Failure” (MTBF)
- Errors and the respective countermeasures are independent and isolated
- Countermeasures have a stable effect, errors do not develop ”penicillin-resistance”

On the other hand, modeling the threats from a hostile attacker must consider the nature of the attacker:

- Attacks are purposeful, targeted at a kind of revenue (money, revenge, status etc.)
- The mind of the attacker cannot be modeled, and there is no probabilistic function to describe the frequency of successful attacks
- The attack tactics will probably adapt to the countermeasures taken, so the protection mechanisms need to be continually improved.

4.2 Purpose and position of the attacker

The success of an attack will rely on different properties of the attacker: The purpose, the skills/knowledge, the tools and the physical position. We will examine these properties in more detail:

4.2.1 The purpose

An attacker may be an economic criminal with the purpose to steal money or merchandise, and not get caught. On the other hand, the attacker may be a vandal or a hacker who is satisfied if the services of the system are temporarily broken. The sophistication of the former attack will be much higher than the second. A more sophisticated attack will also leave more traces which can be used for forensic purposes.

4.2.2 The positions

The attacker will be physically present in one or more places on the network, represented by the network nodes that he/she exploits for the attack. Having taken control of a computer inside a security perimeter more or less invalidates the security offered by that perimeter.
The positions of the attacker is a matter of protection, traceability and exposed data; There is probably more than one security perimeter around a node under attack, and a compromised node closer to this node will likely contain more useful information (for the attacker) and be more trusted[2] by the attacked node. When a node is under attack, other nodes in its neighborhood is therefore also at risk for attack.

The matter of traceability is that an attack conducted from a close node leaves fewer traces than if the attack must use a path of intermediate nodes with logging capabilities. Although, one should keep in mind that a path of *compromised* intermediate nodes could also be exploited in order to disguise the origin of the attack and the relation between events on separate nodes.

Since computer nodes on a local network are likely to cooperate, they will contain information about each other. This information could be exploited during the attack in order for the attacker to map the network and the roles of the nodes. An attack conducted from a nearby node should therefore be considered as a more serious threat than if the attack took place from a distant computer node.

[2] Trust in the sense that a process on one computer may be given automatic privileges on another computer without process authentication (possibly with authentication of the computer node)

4.2.3 Skills and tools

A part of the threat model should consider how the attacker may exploit the position of the compromised nodes, i.e. the ability of the attacker to threat the integrity of the system. The *skills and tools* possessed by the attacker are therefore necessary to consider.

The required skill in order to conduct a successful attack is becoming smaller. The necessary information, how-to's and software tools are found on the Internet. This aspect suggests that the population of potential attackers is growing (Tenet 1998).

4.2.4 Timing and sequence of events

Important for the severity of an attack is the timing and sequence of events which occurs during the attack. A normal sequence of events is shown on Figure 4.1 and consists of the four phases:

Intrusion → Detection → Termination → Recovered

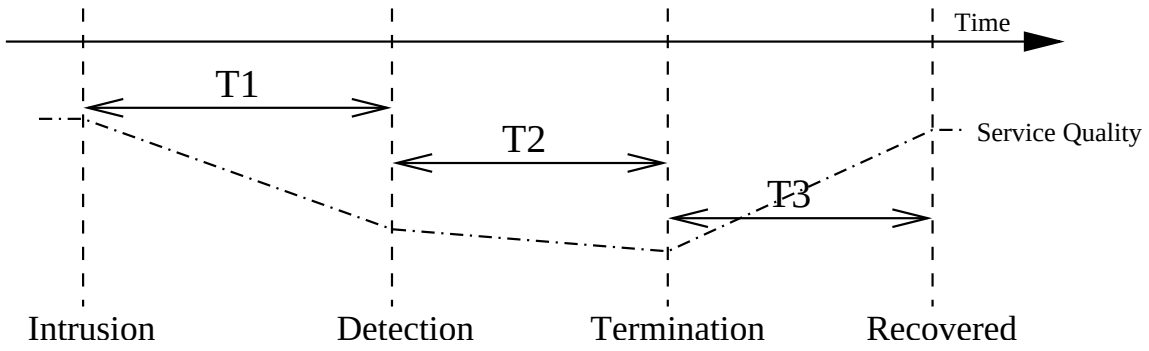

Figure 4.1 - Sequence of events during an attack

4.2.4.1 T1: Intrusion->Detection

The time it takes from a successful intrusion occurs until it is detected is critical, since no countermeasures can be taken before the intrusion is detected.

4.2.4.2 T2: Detection->Termination

After a successful attack has been detected, countermeasures will be taken in order to terminate the attack. After termination, the attack does not longer inflict damage on the system, but the system is still in a corrupted state. The time it takes to terminate an attack may be affected by the attack itself, e.g. if the countermeasure mechanisms are damaged by the attack.

4.2.4.3 T3: Termination-Recovered

The next period of the attack scenario is the recovery phase. It starts when the the attack has terminated and ends when the system has recovered into a safe and consistent state. This period includes the investigation necessary to ensure a successful recovery.

4.2.4.4 Sequence of events and state of service quality

It is possible to consider a ”service quality” as the number of unaffected services offered by the system. The system quality is likely to decrease during T1 and (to a lesser degree) T2, since services may be compromised or interrupted during an attack. The service quality is expected to increase during T3 as the recovery phase is progressing and bringing the services back. It is also reasonable to assume that the quality of system services will be lower with longer duration of T1 (and partly T2). Service quality is shown as the dotted line on Figure 4.1.

4.2.5 The disloyal insider

Given this brief analysis of the importance of position, skills and tools, the threat posed by a disloyal insider seems to be particularly serious. A disloyal insider is a person that has knowledge about the internal network, the configuration of computer nodes and software, business processes, security mechanisms etc., and is also authorized at some level through user accounts. A disloyal insider may know in advance the requirements for a successful and undetected attack which leaves little traces for investigation. It is therefore very difficult to protect the system effectively from insider attacks. Besides, a significant fraction of computer crimes originates from the inside of the affected organization (CSI/FBI, 2006).

4.3 Human factors – human errors

Although not a willful attack, the lack of skills, experience and motivation by personnel inside an organization may create vulnerabilities in the computer system. Poor management of firewalls, poor password practice, lack of e-mail screening etc. may open up well known security holes which may later be exploited by an outside attacker. During investigation of computer crime incidents, it is often seen that security holes either planted by virus/trojan programs or resulting from sloppy conduct are being exploited.

4.4 Program bugs

Program bugs are likely to be the most commonly found factor in computer crime incidents. There are two reasons for this:

1. The program development cycle in the industry is hard pressed on time. The competition for bringing new features on the market reduces the development and debugging time to an extent that reduces the quality of the software. It is probably correct to state that increased competition increases the number of offered features, but also increases the number of bugs that reach the end users.
2. The operating system market is a monoculture, with one family of operating systems (the Microsoft Windows family) installed in more than 90 % of desktop computers. When software vulnerabilities are discovered, they represent a risk for a large number of computers, and the knowledge about how to exploit the vulnerability is suspected to spread fast (cfr. section 6.4).

4.5 Denial of Service

One very common type of attacks does not affect the integrity of the system, only its availability. They are called *denial of service* (DoS) attacks, and are presented separately since these attacks are very common and hard to deal with.

A server or the connecting network may be overloaded by requests sent by a powerful adversary, leaving the service apparently unavailable. A so-called *bandwidth attack* may not even be noticed by the target node if the saturation blocks the traffic farther upstream. A bandwidth attack may also be indistinguishable from a *flash crowd*, a term which describes a sudden rise in the request rate due to some ”latest news” which everyone want at the same time.

A different form of attack exploits bugs or weaknesses in the communication protocol implementations, like the *SYN flood* attack (described in section 3). These attacks may cause the computer, the server processes *or other processes they rely on* to stop or crash. An attack on e.g. a DNS server may affect a lot of other nodes which relies on it.

Many attacking nodes can cooperate in a *Distributed Denial of Service* (DDos) attack. A DDoS attack can simply create an overwhelming amount of requests which can keep a service unavailable for as long as the attack endures, but does not necessarily create permanent damage. Protection against DDoS attacks also requires a distributed solution. Peng et al. (2007) offers an analysis and a survey on DDoS protection mechanisms.

4.6 Summary

The risk for computer intrusions and attacks can be analyzed with a simple model, whereby the different factors and properties of an attack are being analyzed. The actual conduct of an attacker cannot be modeled, but the factors enabling an attack may be. This chapter has presented a short analysis of these factors.

5 Research approaches

Researchers of intrusion tolerance apparently have their background from either *fault tolerance* research, *computer security* research or *distributed systems* research. All areas have valuable tools for prediction and analysis of errors, mitigation and recovery. All areas also share the same shortcoming in modeling of attacks.

5.1 Fault tolerance approach

Fault tolerance research (e.g. Verissimo 2002) deals with analysis and prediction of ”random” errors, i.e. errors that may be modeled by some probability density function (e.g. the Poisson distribution). Fault tolerance research distinguishes between *faults*, *errors* and *failures* in the following manner:

Faults — A possible cause of an error, e.g. a crack in a solder point, a broken transistor inside an integrated circuit or a stopped cooling fan. A fault will not necessary generate an error, but have the potential to do so.

Error — A malfunctioning system component, e.g. a crashed disk or a stopped server node. An error will affect the service and performance of the system, but not necessarily cause the entire system to fail.

Failure — The malfunction of a system, e.g. a netshop unable to take orders. A failure is the consequence of one or several errors. Error in a component which is *a single point of failure* will cause a system failure.

Faults are unavoidable, since they are the result of normal physical processes. Faults may propagate into errors unless they are *tolerated.* Errors may propagate into failures unless they are *masked*. The terms *fault-tolerance* and *error masking* denote related techniques since they avoid one irregular condition to propagate into a more severe condition.

Fault tolerance and error masking techniques employ *redundancy* and *fail-over mechanisms* on different places in the system. Sensors, service nodes, communication media, storage media and effectors can be duplicated to make them more reliable, provided a sufficiently safe fail-over or voting mechanism (which themselves may fail and must be made fault-tolerant).

Fault tolerance research suggests a distinction between *transient*, *intermittent* and *permanent* faults and errors (Tanenbaum and Steen, 2002). Faults/errors are:

Transient — when they appear once, and then disappear. A radio beam momentarily interrupted by a passing bird is an example of a transient error.

Intermittent — when they occur and vanishes spontaneously, but reappear later. A poor electrical connection which generates a disconnection due to external vibration is an example of an intermittent fault.

Permanent — when they appear once, and require manual repair to the faulty component, e.g. a burned fuse.

On top of this framework, researchers (i.e. Verissimo, 2002) attempt to build an intrusion tolerant system. The sequence of events shown in section 4.2.4 (Intrusion → Detection → Termination → Recovery) must then be mapped onto the fault-tolerant framework.

Intrusion – A Fault Tolerance framework as outlined above does not deal with intrusion prevention, since faults are considered to be an unavoidable physical phenomenon.

Detection – offered through *fault detection*. Fault detection will use different techniques for different failures: *Omissive failures* (no response from service, or too late response) are detected by timers, whereas *response failure* (timely response, but with syntax errors or incorrect values) are detected by syntax inspection, value range constraints or voting mechanisms. More on this in section 5.3.

Termination – the detection phase may trigger a fail-over mechanism which switches the clients from the compromised to an uncompromised service. The issues regarding state migration during this process will be mentioned during the discussion of Distributed Systems. (In case of a stateful server the process state should also be migrated, but only if this can be done without risk for the uncompromised server.)

Recovery – recovery from crash and failures are most oftenly researched under the field of Distributed Systems (although these two fields often merge their efforts). These issues will therefore be mentioned under the presentation of Distributed Systems research (Section 5.3).

5.2 Computing security approach

For the purpose of the discussion in this report, computing security research can be divided in 4 categories:

Prevention – The traditional approach of computing security has been how to prevent incidents (either malicious or accidental) through operating procedures, training, certification, cryptography and operating systems software. On the technical level, the use of hardware-supported separation of operating system processes enables the operating system software to monitor and arbitrate the resources allocated to processes and the interactions between processes. These mechanisms form the basis for protection of integrity, confidentiality and availability of resources (files, programs etc.) in the system. Cryptography techniques prevent attacks on confidentiality and integrity, but not availability of information (e.g. in the form of a DoS attack). Applications for cryptography protocols in intrusion tolerant system will be studied in more detail in section 6.1.

On the organizational level, prevention means to *manage* risk and trust. Computing security addresses trust and risk management through comprehensive frameworks and analysis (Abererer and Despotovic, 2001, Naldurg and Campbell, 2003), but the proposed solutions are hard to verify experimentally.

Mitigation – starts with detection of the incident. In the case of computer attacks, this is called *Intrusion Detection*. Upon a positive detection the system should (manually or automatically) limit bandwidth used by the attacker, shut down selected services, migrate legitimate users away from compromised resources etc. Intrusion detection technology will be discussed in more detail in section 6.2.

Recovery – The field of computing security does not offer an independent view on recovery issues, but offers the same operating principles as other system sciences: Redundant servers, backup of storage media, transaction logs and checkpoints, antivirus toolkits and recovery plans (well known and drilled).

Investigation – The field of computer forensics offers guidelines and best practices for forensics in connection with computer crime. To a large degree, they are application-aware mechanisms that assure that all business operations are permanently logged in a manner that allows the investigators to view the sequence of events that took place during an attack. In a distributed environment, this requires data fusion activities and well synchronized clocks.

Computing security research is highly relevant for work on intrusion tolerance. Its focus on the combination of organizational and technical measures for incident prevention, and the use of formal logic and mathematical evidence have provided many sound and well-founded results.

5.3 Distributed Systems approach

Distributed Systems is a wide research area with activities that overlap activities from other research fields, e.g. computing security and fault tolerance. In this report, the presentation of distributed systems research issues focus on the organization of software components in reliable, resilient and consistent systems.

Distributed systems theory promotes the *single-system image* where the physical distribution of resources is kept invisible or *transparent*. Layering and encapsulation techniques are employed to e.g. hide fail-over mechanisms behind a stable interface. Matters of failure detection, service replication, process migration and crash recovery are kept out of the client's view. Of special relevance to intrusion tolerance are the issues of distributed systems that deal with resilience and reliability. A number of techniques used in this area will be briefly presented:

Failure detection - A node or communication channel can fail in different ways. We distinguish between *omissive failure* (no response from service, or to late response), *response failure* (incorrect response or incorrect response value) or *byzantine failure* (any behavior, possibly malign). Omissive failures are detected by time-out watchdogs. On the other hand, it has been proven (Fisher, 1983) that perfect fault detection is impossible in asynchronous network (and any IP network is asynchronous). Response failure is detected through inspection of message syntax or by value range constraints. Byzantine errors are detected through voting mechanisms in combination with the other techniques. It is easy to see that failure detection never will be perfect, so any mechanism that relies on failure detection must take the chance for incorrect detection into account.

Data replication - Both for optimization and reliability purposes, a data store can be distributed or replicated over several computer nodes. Where data is stored in multiple copies (replicated) there must be a synchronization procedure to make all copies up-to-date. Distributed Systems theory proposes several *consistency models* whereby the ordering and propagation of updates are formally described. A group of models called *user centric consistency models*

(Tanenbaum, 2002) are able to describe the situation which occurs when a client is switched from one store to another, e.g. during a fail-over operation.

Process replication and migration – For reliability purposes, a service may be offered by several processes. A fail-over mechanism may switch the clients from a faulty to a healthy server in order to sustain an essential service. In case the service is stateful, i.e. it maintains a session context between service calls, the context has to be migrated too. The server *process* is the operating system entity that maintains the process context, thus the term *process migration* denotes the migration of a stateful service. In general, process migration is not possible using the mainstream operating systems. What is possible is to write services in such a manner that the necessary context is maintained within specific tranferable objects.

Distributed snapshots – for recovery purposes, the system could make snapshots of its distributed state at regular intervals. Due to messages in transit, the system state is not simply a perfectly coordinated snapshot of every node's state. Every node must take their snapshots in a sequence so that the snapshots together can form a coherent picture of the entire system. The algorithm for taking a distributed snapshot is well understood, but costly in terms of computing and communication resources. For recovery of distributed systems other approaches are more popular, e.g. using stateless servers[3] and idempotent (repeatable) operations (cfr. Section 6.3.3).

Distributed transactions – also used for recovery purposes in the same manner as centralized transactions, i.e. to obtain atomic state transitions in the system without exposing intermediate results of an operation. Distributed transactions offer the same semantics as other transactions (the ACID properties) through the use of a "conductor" which controls participating nodes. The conductor (often called a transaction monitor) becomes a single-point of failure which contradicts some of the advantages we seek when employing e.g. a replicated store which must be transactionally coordinated.

Crash recovery – for a reliable system, crash recovery is an important part which should be able to recover the system to a valid and consistent state without any operations being lost or duplicated. Crash recovery can take two approaches: Full transactional support of all operations (using checkpoints and transaction logs) so that the system always knows the state to which to recover, or using stateless servers and idempotent operations and re-run all operations not positively known to have completed. Detailed study reveals that neither of these approaches is perfect. Transactional support does not include the state of the client, and idempotent operations is not always possible to use (e.g. in cases where nodes communicate with data streams instead of messages). Therefore, crash recovery becomes a part of the application development

[3] Stateless servers do not maintain a session context, all operations are unrelated (e.g. no notion of logged-on users). They may operate on stateful resources (e.g. a database) though.

process which cannot be solved fully in middleware (Moore and Ellison, 2003). More on crash recovery in section 6.3.

Researchers of distributed system employ these techniques in order to make systems that are reliable and able to recover safely under any circumstances. In an intrusion tolerant perspective, the techniques are adapted to the threat model that applies to this perspective.

Distributed systems will employ many techniques from computing security in order to prevent successful attacks. These techniques have been discussed in section 5.2.

5.4 Limitations of the three approaches

The research fields of fault tolerance and distributed systems have a large overlap in an area often called *Reliable Systems*. They offer a range of techniques to mask and correct faults and errors on different levels in the system, but they also share *the lack of a threat model*. Spontaneous errors due to physical processes, lack of operating procedures, poor training etc. may be subject to probabilistic modeling. The behavior of a human adversary, however, does not lend itself to probabilistic or statistical modeling. Human behavior may be modeled using game theory or economic theory (Liu 2003).

6 Supporting technologies

As described in section 5, the research on intrusion tolerant systems is based on fault-tolerant computing, distributed systems and computing security. To some extent, research efforts on intrusion tolerance have been found to have their basis in one of these areas. It has also been discussed how these research areas have failed to model adversary behavior.

During the construction of intrusion tolerant systems, several building blocks may be used. These building blocks consist of solutions to well known problems which are relevant for the construction work. In the following sections some technology areas will be examined in more detail in order to identify their contribution to the subject.

6.1 Cryptography

6.1.1 Symmetric vs. asymmetric cryptography

When discussing contributions of cryptography, a distinction should be made between *symmetric* and *asymmetric* cryptography.

Symmetric cryptography: One shared secret key is used for encryption and decryption. It is computationally inexpensive and protects confidentiality and integrity[4] of data. It

[4] since it is not feasible to alter an encrypted message so that the decrypted message still is “meaningful”. A meaningless message will violate syntax requirements etc. and be detected. A *Hash algorithm* may also be used to detect integrity violaton.

also offers a simple authentication (since a valid encrypted message can only be made by one of those who know the secret key), but leaves behind an unsolved key distribution problem, since keys have to be disseminated on a separate protected channel.

Asymmetric cryptography: Separate keys are used for encryption and decryption. The key used for encryption is called ”public” and is distributed freely. The key used for decryption is kept secret by the owner. It is computationally expensive, and protects confidentiality, integrity and authenticity. It alleviates the key distribution problem, since public keys do not need to be sent over a secret channel[5].

Even a simple encryption algorithm (one that can be attacked and broken in a matter of hours) shifts the economy of the attack. By adding to the cost (in terms of time) the attacker must bear for the attack, and adding to the uncertainty of the outcome, the value/risk ratio becomes smaller and the whole mission becomes less attractive. The same holds for the integrity of information; making “meaningful” and undetected changes to information will be much more expensive once even a simple cryptography algorithm is in use.

The electronic signature (a feature offered by asymmetric cryptography) offers strong protection of authenticity, provided that the public key used to control the signature is correctly disseminated and the private key (used to sign) is kept secret[6]. A signed piece of information has an identified source (the signer) and a guaranteed integrity. *Authenticated sessions* can form a basis for authorization control and management when the proven identity of the session owner is connected to an access control matrix.

6.1.2 Threshold cryptography

Threshold cryptography is a special cryptography technique where the key (symmetric or asymmetric) is transformed into an arbitrary number of partial keys. In order to construct the key for encryption/decryption/signature purposes a fixed number of partial keys must be combined. Threshold cryptography forces partial key holders to collaborate, and allows operations to take place only when a given number of them agree. If the required number of key holders forms a majority or a quorum, threshold cryptography may be used for voting or consensus arrangements.

Applications of threshold cryptography have been observed within distributed certificate authorities (Zhou et al., 2002). In these systems, no single point of failure is present, and any members can form a group with the ability to renew a digital certificate.

6.1.3 Contributions to intrusion tolerance

Cryptography supports intrusion tolerant systems by:

[5] The public key still needs to be *authenticated*, e.g. by a public key certificate.
[6] provided that the key is correctly generated and has a sufficient number of bits.

1. Keeping information confidential. The less information an attacker has, the more difficult it is to succeed with the attack.
2. Holding people responsible. Signed information and sessions will identify the source of the operations. Disloyal insiders or the owner of stolen private keys will be identified (if the log files remain intact during the incident).
3. Offering ”security-in-depth”, by presenting to the attacker a series of barriers that need to be crossed (through stolen keys or broken cryptotext). The different barriers would be implemented with different keys/algorithms to avoid a ”domino effect” when one barrier is broken.

There is a fly in the ointment though: Even though the cryptographic algorithms are rigorously verified, their implementation may contain program bugs, and they may rely on program libraries of unknown quality. For instance, a signature operation may fail if the algorithm is applied to information that has been altered by hostile code. Other problems include:

- Cryptography does not enforce autorization or access control. Access control must be done by kernel software which is vulnerable to a different range of attacks.
- The processes that use a secret key for signing or decryption need to store this key in memory cells. An adversary process may obtain access to these memory cells if the memory protection is circumvented. A privileged (kernel) process is allowed to access any memory cell.
- Public key cryptography practices recommend that public keys themselves are authenticated through digital certificates and operations on a *public key infrastructure* (PKI) over network connections. Through an attack on the network infrastructure an adversary can reduce the quality of the authentication, and possibly cause an application to fail due to lack of authentication.

6.2 Intrusion Detection

Prior to any countermeasures, an incident must be detected (section 4.2.4.1). An undetected intrusion is a very dangerous situation since no active countermeasures can be deployed. Passive measures like offline backups, fail-over mechanisms, access control and cryptography protection will only delay the progress of the attack as well as mitigate the effects from it. Consequently, research efforts have been made to detect intrusion (and other security related events, like a virus infection) as fast and precise as possible.

Although research on the field started as early as 1980[7], the seminal paper on the field probably is ”An Intrusion-Detection Model” by Dorothy Denning (1987). It proposes that intrusion/abuse of a computer system can be distinguished from legitimate use through observation of the computer (file access, I/O activity etc.) and the connected network (transported volume, timing pattern, distribution of network addresses, content of PDUs).

[7] http://www.securityfocus.com/infocus/1514 [Oct 31, 2007]

Now a large and growing field of research, intrusion detection has been divided into two technical categories: *Host based* and *network based* intrusion detection systems, abbreviated HIDS and NIDS. They differ in the following manner:

HIDS is installed as a computer program and monitors the activity inside one computer (file accesses, process behavior and log file analysis) and responds to patterns which indicate malicious activity. Since a HIDS can monitor the state of the processes in the computer, not only I/O-activity, it gets a more detailed situation picture. On the other hand, it does not see activity in other computers, and a HIDS is as vulnerable to attacks as other processes in that computer.

NIDS is installed as network probes (or integrated in network switches) and monitors network traffic. A NIDS can monitor a coordinated attack on several computers in real time, and can detect attacks based on violation of network protocols (e.g. a SYN flood), something a HIDS is unable to do. Besides, a NIDS can be highly resistant to attacks since it does not need be addressable on the network (does not need an IP address, not even a transmit wire). Its disadvantage is its limited scope without access to the internal state of the computers, and its inability to monitor encrypted traffic.

Intrusion detection systems can be distinguished on other properties as well:

Rule checking or anomaly detection: An IDS may be driven by a rule set to which all observations are compared. These rules are patterns and a pattern matching algorithm decides if the observations violate the rules and what action to take. On the other hand, anomaly detection reacts to variations in the observed situation using fuzzy logic or calculations on high-dimensional space (Yao, Zhao, Saxton 2005, Yao, Zhao, Fan 2006).

Pre-configured/self-learning: An IDS may be set up with a static set of patterns to which observations are compared, or may be self-learning through feedback mechanisms.

Networked/Standalone: An IDS not connected to any other IDSs would be termed a standalone IDS, and make its decision based on the limited scope of the system (discussed above). Several systems (HIDS and NIDS) can complement each other's scope though network communication and get a better situation picture for its decisions (Jahnke et. al, 2006). A networked IDS would be more visible on the surrounding network and thus exposed to attacks.

The contribution to Intrusion Tolerant systems should be obvious; it reduces the duration between intrusion and detection (called T1 in section 4.2.4.1). Some final remarks on what should be kept in mind about intrusion detection:

- Intrusion detection mechanisms will be a likely target during the early stages of an attack

- An attack will try to remain undetected through knowledge about the algorithms and patterns used by the detection system. Intrusion detection should therefore have unpredictable elements in its mechanisms.

6.3 Distributed Recovery

In the perspective of fault tolerant computing, the issue of recovery is a matter of recovering a system state from a spontaneous error, possibly after the cause of the error (e.g. a faulty component) has been replaced. It is therefore safe to assume that the recovery process can proceed under normal circumstances and a normal error rate.

In an intrusion tolerant perspective this becomes much more complicated. Data gathered with recovery in mind (distributed snapshots, transaction log, and shadow disks) may have been tampered with during the attack (and before it was detected). In principle, there is nothing an adversary process *cannot* do to a system during an undetected attack, including affecting the future recovery process.

Recovery can be based on data gathered for this purpose alone (e.g. backup disks) or based on data gathered for intrusion detection purposes. A host-based intrusion detection system (HIDS) makes its decision on observations of the processes in the host, which can be useful during a recovery process; if a process is deemed to be hostile, the earlier actions of this process could be cancelled or reversed (e.g. all files that have been created are deleted). An implementation of a recovery mechanism based on the Windows event log has been demonstrated by Reynolds and Clough (2003).

Recovery can be done *forwards* or *backwards*, in both cases is the objective of the operation the transition to a valid and consistent system state. Forward recovery can be seen in e.g. network protocols, where a corrupt packet can be restored using Forward Error Correction (FEC). The checksum of a packet can have the property that it not only detects errors, but to some extent also corrects it. The FEC code lets the system to calculate the next valid state of the protocol and moves the protocol to this state. Backward recovery is a ”rewind” of the operations back to a valid state, e.g. by copying the last system backup.

Forward recovery is seen on small scale systems like a network protocol or in a disk controller, but rarely in full-scale, distributed systems. Backward recovery aims to bring the system back to a safe and valid state, but it is *not theoretically possible* to guarantee that clients and servers will agree upon what that state actually is. This fact is presented in any textbook on distributed systems (e.g. Tanenbaum 2002) and will now be briefly discussed:

6.3.1 Call semantics in the presence of failures

During a client/server invocation, a pair of messages is sent between the client and the server, shown in Figure 6.1. The transmission of the response message indicates that the server has completely processed the request. Lack of a response message does not, however, indicate to the

client that the request message has not been processed. The response message may been lost during transmission (1), or the server may have crashed after the request has been processed, but before the response was sent (2), in which case the request has been processed. Or the server may have crashed before the request was processed (3), or the request message may have been lost (4). All situations look the same to the client, which is *unable to know the state of a crashed server*.

In the same manner, the server does not know the state of a crashed client. The client might have crashed before (5) or after (6) the response message has been processed, and the knowledge about the completed service has been stored on a non-volatile storage.

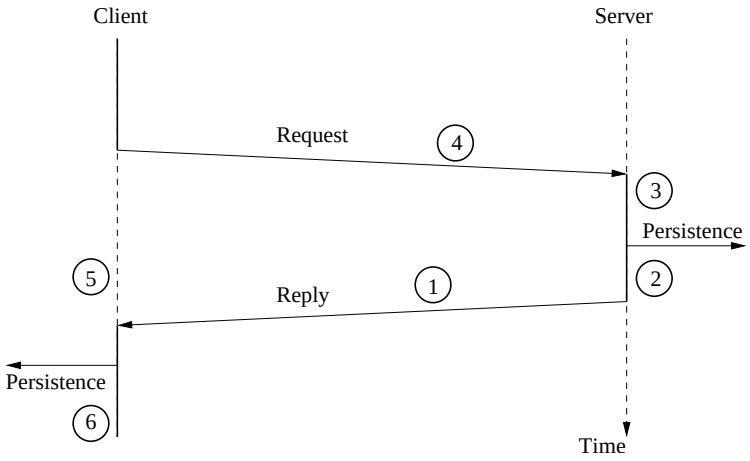

Figure 6.1 - The messages flowing in a client server interaction

This situation only applies to crash recovery, not as long as the system runs normally. When a crashed server is restarted, the client will not know if the last issued request was processed, and will have to decide if the request should be re-issued. Based on its decision and the state of the server, the request may be processed once, twice or not at all. So called *Exactly-once semantics* is not possible to obtain.

Distributed snapshots (section 5.3) can establish a common picture of the system state at a given instant. But this is a state of a system running normally, and the snapshot becomes momentarily invalid. The distributed snapshot can be used for backwards recovery, but the same uncertainties about the processes which took place during the crash remain unsolved.

6.3.2 Stateless servers

Servers that do not keep context information are called *stateless.* A web server only handling request to static HTML pages would be stateless. A client user may follow a series of hyperlinks without knowing if the server has been restarted between the mouse clicks. If the server was to keep session information, e.g. the user's log-on id, then a restart would force the client to log on again, interrupting the normal flow of work.

This flexibility offered by a stateless server is valuable in a recovery perspective, since it does not contribute to the state of the system, i.e. it does not need to take part in distributed snapshots etc. The problem called *partial crash and failure* (referring to the problem of recovering one crashed node in an otherwise normally running system and synchronizing its state with the rest of the system) becomes greatly alleviated if stateless nodes are used.

6.3.3 Idempotent operations

The term *idempotent operations* (Bacon and Harris, 2003, p. 431) refer to operations that can be repeated several times with the same result as if it was done once. Assigning a value to a variable is an idempotent operation, whereas adding data to a sequential storage media is not. Any read/retrieval operation that does not affect the state of the server is idempotent.

Idempotent operations alleviate the problem of *exactly-once semantics* presented in section 6.3.1. Recall that exactly-once semantics is not possible to obtain in the presence of independent crash and failure. Applying idempotent operations to an *at-least-once semantics*, which is possible to obtain by re-issuing requests until they succeed, solves the presented problem in an elegant way.

It is not feasible, however, to construct an entire system that only consists of idempotent operations. The operations of a server tends to build on the present state of that server, e.g. the SQL ”insert” and ”update” commands. Idempotent database insert operations require new unique fields to be added to every table, which may be impossible in a legacy system. Dataflow-based applications are also hard to construct with idempotent operations only.

6.3.4 Replicated storage

The most obvious measure for crash recovery would be the mirrored storage. If several storage media contain mirror-images of the same data, then an intact disk drive can take over immediately when another drive fails. This is a commonly deployed solution in the form of RAID systems (Redundant Array of Inexpensive Disks). RAID can be configured with mirror-imaged disks or with parity disks, so that the data will survive one or a small number of disk crashes. RAID solutions are successful because:

1. The disks are located together, so they may be controlled by highly reliable controllers over a high-speed data bus. The disks are kept in sync through lock-step operation and by sending identical write operations to each disk.
2. Disks have a simple *crash model:* They work fine until they suddenly crashes (called *fail-stop*). As long as a disk completes an operation it can be assumed that the information is correct.

Neither of these assumptions hold in a distributed environment, even less during a successful attack. The fail-stop model does not apply to a compromised system, which may violate the integrity of information before the attack is detected. The delay of communication may inhibit the replicas from operating in lock-step, leaving the designers with other options like a *replicated store.*

A replicated store consists of storage nodes that keep themselves synchronized through replication protocols. A replicated store does not offer mirror-imaged copies, but a well-defined *ordering semantics* which describes the propagation of updates to the store. The clients of the replicated store should *not* regard it as a simple disk system, but reflect the ordering semantics in their application code.

6.3.5 Contributions to intrusion tolerance

The theory on distributed systems offers a thorough discussion on crash recovery and presents formal frameworks for *crash models, forward/backward recovery, invocation semantics, idempotent operations, replication protocols* and *ordering semantics.* These frameworks also suggest protocols for the fine-grained operation of a crash-resilient system.

The larger crash-resilient architectural solution, however, must be integrated into the system design after the essential system services have been identified and the recovery process has been accepted from a business process perspective.

Likewise, in an intrusion tolerant perspective, the frameworks make the building blocks for a system which is resilient to the threats from an adversary, which have been discussed in section 4. The differences from crash-resilience which a designer should take into account are:

1. The system components are to varying degrees exposed to compromise. Components who represent a single point of failure, servers which have service access points connected to the Internet and nodes responsible for intrusion detection and event logging is likely to be interesting targets to an intruder.
2. The compromise of a node should not be considered as an isolated event, but as a part of an attack scenario. Both prevention and detection mechanisms should be verified against valid attack scenarios.
3. The recovery mechanisms may be self-protecting once an intrusion has been detected, but until then (during T1, cfr. section 4.2.4.1) they are attack targets as well. Before recovery, it must be evident that the recovery mechanisms have not been tampered with.

6.4 Diversified systems

6.4.1 The perils of a well known memory layout

All software has bugs, and bugs may open security flaws. Security holes may be the result of a program error that inadvertently modifies other memory cells, including those cells which contain program code. These errors can be exploited to enter hostile program code in the computer.

A famous type of attacks is the ”buffer overflow attack”, where a large input string (e.g. a very long URL string[8]) exceeds the space allocated for it and overwrites the subsequent memory locations. If these locations contain program code, the program may be modified with the content of the input string, and consequently alter its behavior under the control of the attacker. Related attacks can use illegal values to bring pointers to refer to code areas.

Viruses take a similar approach, but overwrite memory cells in memory as well as binary code in executable files. A virus will not attack through a program bug, but hijack a running process e.g. through executable e-mail attachments.

Both worms and viruses are known for their self-replication properties, meaning that a compromised computer becomes an agent for further attack on the same or other computers.

A requirement for the scenarios mentioned above is that an attacker knows the memory layout of the program, in order to modify the expected program code with the intended malicious code.

Program code is, with few exceptions, distributed in compiled form and in binary executables, so every user of a program uses identical program files (on disk) to start the program. When this program is started in a computer, the program loader of the operating system will read the information in the binary executable and set up memory segments for code, data, stack etc. Since the program code is the same in all computers running the same operating system, this layout will be the same almost everywhere.

6.4.2 Immunity through diversification

Since more than 90 % of the world’s computers run a variant of Microsoft Windows, the Internet becomes a ”monoculture” of computers with very similar DNA, and more or less the same vulnerabilities. A program that exploits a recent (not yet fixed) program bug can affect an enormous number of computers. This is why users of Microsoft Windows are bombarded with system updates which must be installed immediately.

This unfortunate chain of events can be broken if systems were *diversified*, i.e. never use the same memory layout. Diversification can be implemented in two different ways:

6.4.2.1 Compilation / Linking

During the compilation and linking of a source program, the development tools behave in a predictive manner, and will always make the same executable from the same source programs. It would be possible to introduce random behavior in the compile-time process in order to make binary executable files different in each run. This is not a good idea, however, due to the following reasons:

[8] The ”Code Red” worm took this approach, see http://en.wikipedia.org/wiki/Code_Red_worm [6 Nov 2007]

- Program distribution is done by copying the executable from one program build, so they all will use the same memory layout anyway
- Distribution of program updates becomes more difficult since the executables no longer can be patched

6.4.2.2 Program loading

For the above mentioned reasons, diversity would rather be introduced during the program loading process, e.g. by introducing a format of the executable file that could instruct the loader about which segments that could be transposed. This arrangement would require the development of a new file format on binary executables, something that is unrealistic for the mainstream operating systems.

The other option is to diversify standard COTS executables. Just and Cornwell (2004) review a number of diversification projects and propose their own solution where an offline analyzer of the executable identifies the points where code segments can be split and transposed.

The identification of the points in the code that can be split in independent segments requires a deep analysis of the execution graph[9] of the program, which is a computing intensive task. But, since it is required only once, it is well suited for running during a preparation phase, where the executable is transformed into a different format suited for a diversifying program loader.

Just and Cornwell (ibid.) also present promising experimental results where well known worms and viruses are let loose on a diversified system, and they show that the spreading pace of the attack becomes much slower than normal.

6.4.2.3 Present state of implementation

The system providers are situated ”upstream”, and their control over the operating system’s architecture enables them to offer diversity solutions in modified or updated operating systems.

There exist a diversity solution for Linux, called PaX[10], which is distributed as a kernel patch that offers diversity of memory space layout in addition to other memory protection mechanisms (like write protection of executable segments). Memory diversity is enabled by default in the Linux kernel since ver. 2.6.12.

OpenBSD and Mac OS X also offer memory diversity, apparently in the form of *library randomization*, where the segment of a library file are not rearranged, but the entire library is loaded at an arbitrary start address.

[9] An execution graph is the collection of all execution paths in the binary program code. Branches and merge points become the nodes of the graph, and straight sequences of instructions become the arcs.
[10] http://pax.grsecurity.net/ and http://en.wikipedia.org/wiki/PaX [Nov 7, 2007]

Microsoft Vista OS is equipped with a diversity solution (called *Address Space Layout Randomization, ASLR*) [11], which offers library randomization, and semi-random placement of the allocated memory segments.

6.4.2.4 Opaque overlay networks

Related to diversification, and therefore mentioned in this section, is the use of *overlay networks* with the intention to hide the servers behind proxies which pass on the requests to the servers. The term ”overlay network” describes a network structure on top of the IP routing structure which run its own signalling protocols to control membership, forwarding paths and other relevant properties.

The (overlay) network of proxies is configured so that several proxies offer a path to one given server (Wang et al., 2003b). If there is a DoS attack on one proxy, other proxies will be able to offer the same service. Therefore, an attacker must conquer several access points at once, not only one.

Overlay networks can also be observed in intrusion tolerant experiments to control a redundant set of nodes and correct/isolate those with erroneous behavior. Every service call is sent to several servers and their responses subject to inspection/voting before sent back to the client (Johansen et al., 2006)

6.4.2.5 The contribution to intrusion tolerance

System diversification should be seen as a preventive measure for intrusion tolerance. Viruses and worms are extremely common security problems, which is partly due to the monoculture of system software.

Breaking this monoculture is a sound and practical approach to break the ”chain of events” during a virus/worm attack. Other approaches to system diversification could be a wider variation of operating systems, web servers, databases and middleware (Wang et al., 2003), but such an approach is not addressed in this report.

7 Research results

The ”silver bullet” solution to intrusion tolerance has not been found. The simple, elegant organization of data, program code and computer nodes which would react correctly to all friendly requests but ignore all hostile ones is what many researchers are looking for. From the perspective of information theory, it is not likely that such a solution does exist. The trusted operator’s action of fraud or loyal work does not differ by more than an amount or a bank account number, which would be undistinguishable for a detection automaton.

[11] http://en.wikipedia.org/wiki/Address_space_layout_randomization [Nov 7, 2007]

In the absence of one single solution, the presented research projects aim to combine the myriad of techniques for partial solutions (prevention, detection, mitigation and recovery) into a system that offers effective and reliable operation under hostile conditions.

A few major intrusion tolerant projects will be reviewed in this section:

7.1 MAFTIA

MAFTIA (2003) (Malicious- and Accidental-Fault Tolerance for Internet Applications) is a research program funded by the European Union (the ESPRIT program) which was completed in 2003. The MAFTIA project had the following objectives[12]:

- The *conceptual model and architecture*: Providing a framework that ensures the dependability of distributed applications in the face of a wide class of faults and attacks.
- The *design of mechanisms and protocols*: providing the required building blocks to implement large scale dependable applications.
- The *formal assessment* of the work: Rigorously defining the basic concepts developed by MAFTIA and verifying the results of the work on dependable middleware.

Paulo Veríssimo was very much involved in the MAFTIA project, and his ideas on intrusion tolerance from the Navigator group (Veríssimo, 2002) can be found in the MAFTIA deliverables. The project proposes the use of *trusted* components (with are hardened and less likely to be compromised) like servers and authentication agents, which are able to communicate through a *wormhole* network. A wormhole network is separate and isolated from the public Internet and therefore less exposed to DoS attacks and intrusions. A wormhole network can use e.g. separate LAN structures or circuit-switched communication like POTS, ISDN or 3G. Replication of trusted services takes place over wormhole network, as well as the coordination between the *Trusted Timely Computing Base* (TTCB). The TTCB is a small security kernel which offers distributed authentication, agreement and timestamps.

The MAFTIA deliverables consist of comprehensive specification of the protocols and mechanisms, a set of intrusion and attack scenarios, and a qualitative analysis of the intrusion tolerance capabilities.

The MAFTIA project offers formal methods for assessment of fault tolerant capabilities. One interesting contribution is the *fault tree* (Figure 7.1), which offers to the analyzer a tool for viewing how faults and errors may aggregate during and attack, and clearly visualize how attacks may rely on the presence of several faults in the system.

[12] From http://www.terena.org/activities/tf-csirt/meeting16/maftia.pdf [Nov 13, 2007]

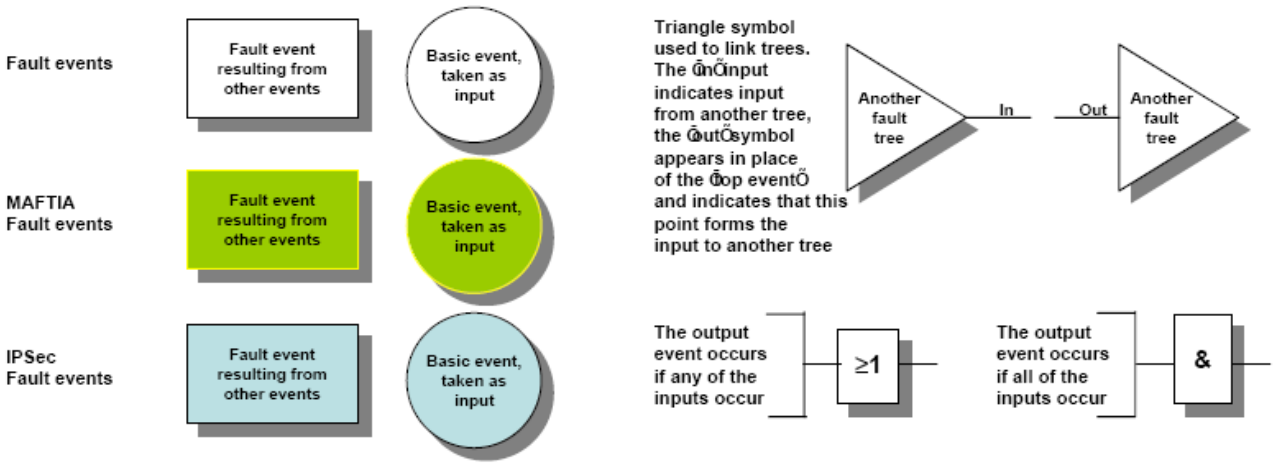

Figure 7.1 - The MAFTIA fault tree

7.2 ITUA

An intrusion tolerant research project called ”Intrusion Tolerance through Unpredictable Adaptation” (ITUA, 2003) is a part of the OASIS project (Organically Assured and Survivable Information Systems) sponsored by DARPA. It is a joint effort of BBN Technologies, the University of Illinois, the University of Maryland, and Boeing Corporation. In the same manner as the MAFTIA project, the ITUA project attempts to build a system framework for an intrusion tolerant system consisting of cooperating agents for intrusion detection, effect mitigation and recovery.

As the name of the project indicates, the employed strategy attempt to adapt to threats (in addition to the normal measures of prevention, detection, eradication and recovery) in a range of actions, from local and fast to coordinated. Examples are:

Local and fast: Block source IP address, recover corrupt file
Coordinated: Isolate and replace corrupt replica (e.g. a service), put a compromised security domain in quarantine

The actions may be expected (from the perspective of the attacker) but should be *unpredictable*. Offering unpredictable adaptive responses reduces the probability of a planned, multistage attack to succeed (Pal, 2000)

Action	***Unpredictable element***
Block IP addresses	Reject? Drop? Delay?
Restore corrupted file	Only file, or file tree?
Isolate/replace replica	Which one to replace with?
Select application object	Which object?

7.3 DARPA SRS

DARPA's activity on "Self Regenerative Systems"[13] is an ongoing project, started in 2004. The objective of the project is:

- Biologically-Inspired Diversity
- Cognitive Immunity and Regeneration
- Granular, Scalable Redundancy
- Reasoning About Insider Threats

The project addresses the field of diversified systems (Section 6.4) and replica management (Section 6.3.4) as well as research on cognitive and reasoning activities for detection and analysis of threats. No full list of publications resulting from the project has been found.

8 Intrusion Tolerance in mobile systems

In mobile systems, communication takes place over radio systems, which means that the intruder does not need physical access to any communication lines in order to conduct an attack. A Denial-of-Service type of attack may under these conditions be targeted on the radio signal level, i.e. as radio jamming. A discussion on radio signal jamming is beyond the scope of the analysis in this report, however. When intrusion tolerance in mobile systems is addressed, there is a list of new issues to consider:

- The network is more exposed for surveillance. Several links can be monitored at the same time, for traffic analysis or information tapping. Routing paths can be discovered through traffic analysis, as well as the location of nodes with centralized (important) roles.
- Intermediary nodes (proxies, firewalls etc.) will have reduced protective effect, since the radio links on the "inside" will be exposed to intrusion and eavesdropping. For the same reason the use of opaque overlay networks will have reduced effect.
- Nodes representing a single point of failure will be as exposed to intrusion and DoS as any other nodes, since all wireless links are equally exposed.[14]
- The intermittent (on/off) nature and the mobility of nodes make it harder to detect adversary nodes.
- More decisions (intrusion detection, trust management, replica management) previously done by well shielded centralized nodes must be done in a decentralized manner, without *single point of failures* nodes and with *byzantine errors* considered. Threshold cryptography (section 6.1.2) and distributed voting mechanisms could be used in these processes.

[13] http://www.darpa.mil/ipto/programs/srs/srs.asp [Nov 14, 2007]
http://www.tolerantsystems.org/srs.html [Nov 14, 2007]
[14] We disregard the effect of directive antennas in this discussion.

- The bandwidth in a radio network is lower than in wired network. The capacity needed for distributed error/intrusion detection, voting and coordination may hinder the flow of application data.

It is possible to rank wireless systems according to how difficult it is to make them intrusion tolerant:

Stationary nodes	All communication links are exposed to tapping and attacks, proxies are not able to encapsulate resources
Mobile nodes	As stationary nodes, but it is more difficult to detect irregular activities due to mobility
Tactical nodes	As mobile nodes, but the enemies must be expected to have more resources (information, skills and technology)

Intrusion tolerance in tactical mobile systems may therefore be regarded as a very challenging research area, where many questions have not yet been investigated. This report conludes with suggestions to further research within this area.

9 Conclusions and suggestions for further research

This report has reviewed the research area of intrusion tolerance, and presented different approaches to the field. The report has also presented a selection of technological building blocks which will be useful during the construction of an intrusion tolerant system.

It should be evident from this report that there does not exist an “Intrusion tolerant system” in the sense that all threats have been taken care of in a secure, manageable and scalable manner. Several smaller problems have been successfully researched, like intrusion detection and crash recovery, but a framework that connects all these parts together seems to impose restrictions on the system. Requirements for interoperability, compatibility, scalability and flexibility may be impossible to combine with these frameworks, and the system architect has to build some overarching arrangement on his own.

Although intrusion detection, system diversification and other sub-areas are still actively researched, the idea of a single overarching structure connecting these parts together seems to have been abandoned. We find no publications since 2003 which indicates any major efforts on the construction of frameworks.

This report will not propose more research on the “macro” questions, but suggest that intrusion tolerance in *mobile tactical systems* is investigated further. Here are a few suggested research questions:

- At the radio link level, are there access protocols that keep attackers from eavesdropping or jamming? E.g., could all stations that do not hear the receiving station deliberately create collisions in order to avoid tapping?
- Could an attacker be detected through position tracking, e.g. would the mobility pattern of a node help to distinguish a friend from a foe?
- At the network level, could alternative routing mechanisms be resistant to hostile routing packets? Could e.g. a publish-subscribe based routing mechanism introduce the necessary unpredictability to make the routing service harder to interrupt?
- At the application level, could a “quality of authentication” parameter (generated at lower level of communication) be passed together with application data to indicate a trust given to the data elements?

References

Abererer K., Despotovic Z. (2001) Managing Trust in A Peer-2-Peer Information System. *ACM Conference on Information and Knowledge Management (CIKM'01) November 2001, Atlanta, Georgia*

Bacon J., Harris T. (2003) Operating Systems. *Addison-Wesley ISBN 0-321-11789-1*

CSI/FBI (2006) Computer Crime and Scurity Survey. Computer Security Institute, 2006. *Online: http://www.cse.msu.edu/~cse429/readings06/FBI2006.pdf* [Nov 19, 2007]

Denning D. (1987) An Intrusion-Detection Model. *IEEE Transactions on Software Engineering, Vol. SE-13 (2), pp. 222-232, February 1987*

Fisher M. J. (1983) The Consensus Problem in Unreliable Distributed Systems (A Brief Survey) *In: International Conference on Foundations of Computation Theory, Borgholm, Sweden* Springer pp.127-140.

ITUA (2003) Intrusion Tolerance through Unpredictable Adaptation. *Online: http://itua.bbn.com/* [Nov 14, 2007]

Jahnke M., Tölle J., Lettgen S. (2006) A Robust SNMP based Infrastructure for Intrusion Detection and Response in Tactical MANETs. *Proc. of the 3rd GI Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2006) 13.-14. July 2006*

Johansen H., Allavena A., Renesse R.von (2006) Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. *EuroSys'06, April 18-21, 2006, Leuven, Belgium*

Just J.E., Cornwell M. (2004) Review and analysis of synthetic diversity for breaking monocultures *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode, Washington DC, USA*

Liu P., Zang W. (2003) Incentive-based modeling and inference of attacker intent, objectives, and strategies *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, Washington D.C., USA, 2003, pp. 179-189*

MAFTIA (2003) Malicious- and Accidental-Fault Tolerance for Internet Applications. *Online: http://www.maftia.org/* [Oct. 2007]

Moore, A. P., Ellison R. J. (2003) TRIAD: A Framework for Survivability Architecting. *First ACM Workshop on Survivable and Self-Regenerative Systems (SSRS'03) October 31 2003, Fairfax, Virginia, USA*

Pal P., Webber P., Schantz R., Loyall J. P. (2000) Intrusion tolerant systems. *In: Proceedings of the IEEE Information Survivability Workshop (ISW-2000), pp 24-26, October 2000. Boston, MA.*

Peng T., Leckie C., Ramamohanarao K. (2007) Survey of Network-Based Defence Mechanisms Countering the Dos and DDos Problems. *ACM Computing Surveys, Vol. 39, No. 1, Article 3, April 2007*

Reynolds J. C., Clough (2003) Continual Repair for Windows Using the Event Log. *First ACM Workshop on Survivable and Self-Regenerative Systems (SSRS'03) October 31 2003, Fairfax, Virginia, USA*

Selvin G., Evans D., Marchette S. (2003) A Biological Programming Model for Self-Healing. *First ACM Workshop on Survivable and Self-Regenerative Systems (SSRS'03) October 31 2003, Fairfax, Virginia, USA*

Tanenbaum A. S., Steen M. van (2002) Distributed Systems Principles and Paradigms. *Prentice Hall ISBN0-13-088893-1*

Tenet G. J. (1998) Remarks of the Director of Central Intelligence George J. Tenet at the Sam Nunn Nations Bank Policy Forum on "Information Security Risks, Opportunities, and the Bottom Line" *Georgia Institute of Technology, Atlanta, April 6, 1998 Online: https://www.cia.gov/news-information/speeches-testimony/1998/dci_speech_040698.html* [Nov 19, 2007]

Veríssimo, P. (2002) Intrusion Tolerance: Concepts and Design Principles. A Tutorial. *Technical Report DI/FCUL TR02-6, Department of Informatics, University of Lisboa. Online: http://www.navigators.di.fc.ul.pt/it/* [Oct 2007]

Wang D., Madan B. B., Trivedi K. S. (2003) Security Analysis of SITAR Intrusion Tolerant System. *First ACM Workshop on Survivable and Self-Regenerative Systems (SSRS'03) October 31 2003, Fairfax, Virginia, USA*

Wang J., Lu L., Chien A. A. (2003b) Tolerating Denial-of-Service Attacks Using Overlay Network – Impact of Topology. *First ACM Workshop on Survivable and Self-Regenerative Systems (SSRS'03) October 31 2003, Fairfax, Virginia, USA*

Welch, I. Warne, J. Ryan, P. Stroud, R. (2003) Architectural Analysis of MAFTIA's Intrusion Tolerance Capabilities *Technical report series- university of newcastle upon tyne computing science, Online http://www.maftia.org/deliverables/D99.pdf* [Nov 8, 2007*]*

Yao J. T., Zhao S. L., Saxton L. V. (2005) A study on fuzzy intrusion detection. *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005. Edited by Dasarathy, Belur V. Proceedings of the SPIE, Volume 5812, pp. 23-30*

Yao J. T., Zhao S., Fan L. (2006) An enhanced Support Vector Machine Model for Intrusion Detection. *In: Rough Sets and Knowledge Technology, Springer Berlin, 2006, pp. 538-543*

Zhou, L., Schneider, F. B., and Van Renesse, R. (2002) COCA: A secure distributed online certification authority. *ACM Trans. Comput. Syst.* 20, 4 (Nov. 2002), pp. 329-368.