Information Assurance

Peng Liu, Pennsylvania State University, University Park
Meng Yu, Monmouth University
Jiwu Jing, Chinese Academy of Sciences

Keywords: Information Assurance, Security, Survivability, Intrusion masking, Defense-in-depth, Intrusion response

As society increasingly relies on digitally stored and accessed information, applications have increasingly higher requirements on supporting the availability, integrity, and confidentiality of this information. However, as the quantity and severity of cyber vulnerabilities and attacks continuously increase, traditional information security technologies are increasingly limited in satisfying the security requirements of applications due to their inability to survive successful attacks. As a result, Information Assurance (IA) technologies are introduced to not only prevent information from being disclosed, modified or destroyed, but also detect intrusions and operate through attacks in such a way that a certain level of information security can be ensured in the presence of attacks. In this article, we survey the natural evolution of information assurance technologies. Three generations of IA technologies are summarized, namely the technologies to prevent intrusions, the technologies to detect intrusions, and the technologies to operate through attacks, and the newest generation of IA technologies are discussed in detail. In particular, we survey intrusion masking technologies and defense-in-depth technologies in the context of survivable information storage systems, intrusion masking distributed computing, attack resistant certification authority, survivable network systems, and survivable of database systems.

INTRODUCTION

As society increasingly relies on digitally stored and accessed information, traditional information security technologies, policies, management and practices are found more and more limited in satisfying the security and assurance needs of modern information systems and applications, for several reasons. In general, addressing only the protection of information against unauthorized disclosure, transfer, modification, or destruction, traditional information security cannot deliver the level of information assurance that

modern applications require. In particular, first, as applications increasingly rely on digitally stored and accessed information, they increasingly rely on the availability of this information and the reliability of the corresponding information system. However, availability and reliability are largely neglected by traditional information security.

Second, although information confidentiality, privacy and integrity protection are certainly crucial in meeting the security needs of modern applications, not all attacks can be prevented and some attacks do succeed. (Readers can refer to Volume III of this handbook for a detailed discussion of the threats and vulnerabilities to modern information systems.) These attacks can cause substantial confidentiality and privacy loss (via unauthorized disclosure of information), substantial integrity loss (via unauthorized modification of information), substantial availability/reliability loss and serious denial-of-service (via destruction of some critical components of the information system), and substantial non-repudiation loss (via destruction of evidence and audit data). When applications were lightly dependent upon digitally stored and accessed information, such information security losses might be able to be tolerated. But as applications increasingly rely on digitally stored and accessed information, such security losses can be disastrous and may no longer be able to be tolerated. Hence, another fundamental limitation of traditional information security is that how to address these successful attacks, or intrusions.

As a result, to meet the security and assurance needs of modern information systems and applications, a broader perspective is introduced, saying that, in addition to preventing information from being disclosed, modified or destroyed, intrusions should be detected; countermeasures (e.g., responses) to intrusions should be planned and deployed in advance; security and fault tolerance mechanisms should work together to ensure confidentiality, privacy, integrity, non-repudiation, authenticity, availability and reliability in the presence of attacks; and the damage caused on the information and the information system should be repaired and restored (or recovered). In this literature, this is referred to as *information assurance*. For example, from the military perspective, Information Assurance must address the delivery of authentic, accurate, secure, reliable, timely information, regardless of threat conditions, within the distributed and heterogeneous computing and communication environment.

The basic meaning of information assurance is well captured by the definition from National Information Systems Security Glossary, which is as follows:

> **Information Assurance (IA)**: Information operations that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities. [*Definition from National Information Systems Security (INFOSEC) Glossary, NSTISSI No.4009, Aug. 1997*]

Compared with the concepts of Information Security and Information Systems Security, whose definitions are quoted below, it is not difficult to see that the concept of Information Assurance is much broader than that of Information Security. In particular, (a) the focus of Information Security is on protection or prevention, while the focus of information assurance is on integration of protection, detection and reaction; (b) intrusion detection and reaction are not a major concern of Information Security, but they are certainly crucial for Information Assurance; (c) attack recovery or restoration may be a

topic out of the scope of Information Security, but it is certainly a critical component of Information Assurance; (d) the goal of Information Security technologies is to prevent attacks from happening, while the goal of Information Assurance is to ensure that even if some attacks intrude into an information system, certain levels of availability, integrity, authentication, confidentiality, or non-repudiation can still be guaranteed.

> **Information security:** The protection of information against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional.
>
> **Information systems security (INFOSEC):** [The] protection of information systems against unauthorized access to or modification of information, whether in storage, processing or transit, and against the denial of service to authorized users, including those measures necessary to detect, document, and counter such threats.

It is no doubt that Information Assurance involves many disciplines and has a variety of aspects, such as the policy, legal, ethical, social, management, evaluation, and technical aspects of information assurance. Compared with traditional information security practices, Information Assurance not only involves the design and development of a variety of new security technologies, but also involves a variety of emerging policy, legal, ethical, social, economical, management, evaluation and assurance issues as Information Assurance evolves people's practices of information security in an ever quicker pace. Nevertheless, to make this article more tangible, this article will primarily focus on the technical aspect of information assurance, though some relevant policy, management and evaluation issues will also be addressed. Readers can refer to Chapter *Security Policy Guidelines* and Chapter *Security Policy Enforcement* for detailed discussion on the policy aspect. Readers can refer to Chapter *Managing a Network Environment* for detailed discussion on the management aspect. Readers can refer to Chapter *Common Criteria* for detailed discussion on the evaluation aspect. Readers can refer to the *Social and Legal Issues* part of Volume II for detailed discussion on the social, ethical and legal aspects of IA.

The rest of the article is organized as follows. In Section 2, an overview of IA technologies is given. Three generations of IA technologies are identified and summarized. And the 3rd generation IA technologies are classified into two categories: intrusion masking technologies and defense-in-depth technologies. In Section 3, we survey intrusion masking technologies. In Section 4, we survey defense-in-depth technologies. In Section 5, we conclude the paper.

OVERVIEW OF IA TECHNOLOGIES

In this section, we will give a comprehensive overview of information assurance technologies, with a focus on the emerging 3rd generation information assurance technologies and their relation with more established intrusion prevention and detection IA technologies.

Three Generations of IA Technologies

In general, existing IA technologies can be "clustered" into three generations as shown below. There is a natural evolution or maturing that has occurred in the IA community, and these generations offer evidence of the evolution.

- **1st Generation: prevent intrusions.** The goal is to prevent attacks from succeeding. The representative technologies are Trusted Computing Base, access control and physical security, multiple levels of security, and cryptography.
- **2nd Generation: detect intrusions.** Since not all attacks can be prevented, intrusions will occur. Hence, the goal of 2nd generation IA technologies is to detect intrusions. Some representative technologies are firewalls, intrusion detection systems, and boundary controllers.
- **3rd Generation: operate through attacks (or survivability).** Since some attacks will succeed, we need the 3rd generation IA technologies. The goal is to enable information systems to continue delivering essential services with security assurance in the presence of sustained attacks. Some representative technologies are real-time situation awareness & response, real-time tradeoff of performance, functionality and security, intrusion tolerance, and graceful degradation. It should be noticed that the 3rd generation IA technologies are not simply focusing on the availability domain; their dimensions are much broader. In particular, without delivering such security assurance as confidentiality (privacy), integrity, authenticity and non-repudiation, essential services cannot be continuously delivered under sustained attacks. In general, survivability not only means availability under attacks, but also means confidentiality (privacy), integrity, authenticity and non-repudiation under attacks. Moreover, in many situations, survivability implies reliability.

It should be noticed that among the three generations of IA technologies, each generation is crucial in achieving the goals of Information Assurance, and no one can replace another. (The 2nd generation IA technologies do not subsume the 1st generation IA technologies, and the 3rd generation IA technologies do not subsume the 2nd generation IA technologies either.) In particular, first, the 1st generation IA technologies build the foundation for information assurance, since without strong protection of information confidentiality, privacy, integrity, authenticity and non-repudiation, there can be too many successful attacks for the information system to survive, which in fact makes survivability infeasible. Moreover, intrusion prevention, intrusion detection and intrusion tolerance (or survivability) actually share primarily the same goal, i.e., to ensure the information confidentiality, privacy, integrity, availability, authenticity and non-repudiation in the face of attacks. A highly trusted and assured information system should be able to prevent as many attacks as possible from breaking into the system, detect the attacks that could not be prevented with accuracy and agility, and robustly operate through and recover from these successful attacks without losing availability, reliability, and accountability. Second, the 3rd generation IA technologies are largely dependent upon the 2nd generation IA technologies, since many 3rd generation IA technologies assume that the intrusions can be detected in a timely manner with good accuracy (e.g., low false positive rate and false negative rate).

Nevertheless, in this article we will focus on the 3rd generation IA technologies, since the 1st and 2nd generation IA technologies will be well covered by the other Chapters of this handbook. In particular, we will survey the technologies for developing survivable (networked) information systems**.** Readers can refer to chapters *Secure Public Networks*, *IPsec*, *SSL/TLS, Secret Key Cryptography, Database Security, Medical Record Security, Access Control: Principles and Solutions, PGP (Pretty Good Privacy), P3P (Platform for*

Privacy Preferences Project), Anonymity and Identity in the Internet, Privacy Law, Privacy Issues in Wired and Wireless Networks, and *Medical Record Security* for detailed discussions on the 1st generation IA technologies. Readers can refer to chapters *Intrusion Detection: Detection Technology and Techniques, Intrusion Detection Systems Basics, Host-Based Intrusion Detection Systems,* and *Network-Based Intrusion Detection Systems* for detailed discussions on the 2nd generation IA technologies.

Third Generation IA Technologies

Related to the fault tolerance concept and drawing from that discipline is the area of intrusion tolerance or survivability. Intrusion tolerance is emerging as one of the most important R&D areas in Cyber Operations today since the systems and networks we depend upon must continue to operate through intrusions and keep operating, although in a degraded mode, in spite of a sequence of successful cyber attacks.

Classification of Survivability Technologies

We can classify existing survivability technologies into two categories: intrusion masking and defense-in-depth. Note that many well-known intrusion tolerance projects are MAFTIA [34] and OASIS [35] projects.

Intrusion Masking From the design perspective, one system design can be inherently much more *attack resilient* than another. The goal of intrusion masking is to redesign a regular vulnerable computer system with enough redundancy so that the new survivable design can function correctly even part of the system is hacked. In this sense, we say the new survivable design can *mask* intrusions. Techniques in this category focus on how to enhance the inherent resilience of a secure system, and their effectiveness is typically much less sensitive to the agility and accuracy of intrusion detection than pragmatic, run-time intrusion response techniques. General principles in developing attack-resistant designs include but not limited to (a) redundancy & replication; (b) diversity; (c) randomization; (d) fragmentation & threshold cryptography; and (e) increased layers of indirections. Techniques in this perspective include but not limited to Byzantine intrusion masking techniques [4,19,24,29] that follow the redundancy & replication principle, threshold-cryptography-based attack resilient systems [7,30,31] that follow the fragmentation principle, multi-path routing [27] that follow the redundancy principle, and resilient overlay networks [3] that follow the "increased layers of indirections" principle.

Defense-in-Depth Instead of redesigning a system, the goal of defense-in-depth technologies is to arm the system with a set of attack & threat response facilities which, with the help of the intrusion detection, can respond to intrusions in such a way that the system can operate through attacks. Technologies in this category include (a) *boundary controllers* such as firewalls and access control; (b) *intrusion detection*; and (c) *threat/attack/intrusion response*. It is well known that boundary controllers cannot prevent every attack.

Intrusion detection [8,15,20,21,25,36] is a key part of many survivable systems, but existing intrusion detection technologies in general suffer the high false positive (negative) rate problem, especially when the detection is anomaly-based or specification-based. Since intrusion detection techniques cannot guide us to respond to intrusions,

existing Defense-in-Depth technologies focus on intrusion response, which can be classified into three categories:

Type 1: Reactive response. Techniques in this category are activated only when an intrusion is identified and their effectiveness is highly dependent on the accuracy and latency of intrusion detection. For example, attack recovery techniques [2,14,28] belong to this category. If the detection is quick and accurate, then the contaminated part of the system can be quickly repaired without causing serious integrity degradation. However, if there are many false alarms, a lot of clean elements could be corrupted by wrong "repairs". Some other Type 1 techniques include but not limited to reactive one-phase damage containment techniques, detection-based (firewall) reconfiguration techniques and patching techniques.

Type 2: Proactive response. Techniques in this category are activated in a proactive manner based on suspicious activities (or signs) before an intrusion is confirmed. Although proactive response may consume more resources, it may immunize the system from the damage caused by many attacks. Moreover, many proactive response mechanisms are transparent to users. Type 2 techniques include but not limited to isolation [13], multi-phase damage containment [12], and sandboxing [18].

Type 3: Adaptive response. Feedback based adaptation is a nice feature of many survivable systems, where the defense posture (i.e., security mechanism configuration) of the system is dynamically adjusted based on the changing environment. Adaptive response addresses the reconfigurable computing and communication aspect of survivable information systems. Type 3 techniques include but not limited to the OASIS Willard project and adaptive ITDB [16].

Since intrusion detection makes the system attack-aware but not attack resilient, that is, intrusion detection itself cannot maintain confidentiality, integrity and availability of information in the face of attacks, intrusion response is crucial is building survivable systems. Moreover, since the fundamental sciences, principles and arts of survivable systems and networks are almost the same, we do not address systems and networks separately; instead, we focus on these common sciences, principles and arts. Finally, malicious code defense is an important aspect of IA, but this aspect will not be addressed in this article.

Survivability vs. Fault Tolerance

As a core concept of the 3rd generation IA technologies, survivability [32] builds on the top of several related fields of study (e.g., security, fault tolerance, safety, reliability) and introduces new concepts and principles. In particular, since many survivability technologies are motivated by fault tolerance technologies, people may wonder the differences between these two fields. In the following, we highlight three major differences between survivability and fault tolerance.

First, in fault tolerance, failures randomly happen; but in security, attacks are typically intentional and do not randomly happen. Moreover, attacks are more intelligent and active (i.e., more intentional and better planned) than failures, so more proactive tolerance techniques are needed for survivability.

Second, intrusion detection is typically much more complicated than failure detection. This is why there are so many new research challenges in intrusion detection.

Third, in the literature of fault tolerance, intrusions in many cases are modeled and tolerated as Byzantine faults, or arbitrary faults. Therefore, if a system is Byzantine fault tolerant, it is able to tolerate intrusions in some degree. BFT [4], a practical Byzantine fault tolerant system, can tolerate both faults and intrusions. However, it should be noticed that not all damages to the system are caused by faults and not all intrusions can be modeled as Byzantine faults. For example, successful intrusions at the application level (e.g., corrupted transactions of database systems), and data corruption usually do not appear as faults and cannot be handled by Byzantine fault tolerance.

INTRUSION MASKING TECHNOLOGIES

In this section, we will survey four representative intrusion masking technologies, namely survivable storage systems, intrusion masking distributed computing, attack resistant certification authority, and survivable network systems.

Survivable Information Storage Systems

As society increasingly relies on digitally stored and processed information, supporting the availability, integrity, and confidentiality of this information is crucial. We need systems in which users can securely store critical information, ensuring that it persists, is continuously accessible, cannot be destroyed, and is kept confidential. A *survivable storage system* would provide these guarantees over time and despite malicious compromises of storage node subnets.

In [5] and [31], how to build a survivable distributed information storage system (e.g., a distributed file storage system) has been investigated. Current distributed file systems are not survivable; whenever a node is hacked, the attacker can not only know the content of the files stored on that node but also modify or destroy them. In [5], a fragmentation-based survivability design is proposed, where (a) every file is fragmented into a set of *fragments* before it is stored on a storage node; (b) fragments are replicated to yield more availability; (c) fragment *replicas* of a file are carefully scattered across the network of storage nodes in such a way that when a node is broken, the attacker can never get a complete set of the fragments for this file. Moreover, to prevent the attacker from getting partial information about the file from the fragments he captures after breaking into a node, each file is encrypted before fragmentation and the cipher-text fragments are made inter-dependent upon each other (using such technique as CBC mode encryption), and the ensemble order among the fragments for the file is kept confidential. In this way, when the attacker captures a set of encrypted fragments, even if he knows the key, he is still unable to decrypt these fragments (in most cases).

In [31], a different secret-sharing-based survivable design, called PASIS, is proposed, although the goal is also to develop a survivable information storage system. The PASIS architecture, shown in Figure 1, flexibly and efficiently combines decentralized storage systems, data redundancy and encoding, and dynamic self-maintenance to achieve survivable information storage. A PASIS system applies threshold cryptography schemes [37] to spread information across a decentralized collection of storage nodes. Client-side agents communicate with the collection of storage nodes to read and write information,

hiding decentralization from the client system. Automated monitoring and repair agents on storage nodes provide self-maintenance features.

In particular, in PASIS a *m-n* secret sharing scheme is used to break a data object (e.g., a file) into *n shares* so that (a) every shareholder (i.e., a storage node) has one of the shares; (b) any *m* of the shareholders can reconstruct the object; (c) but a group of fewer than *m* shareholders cannot gain any information about the object. Note that a share in [31] and a fragment in [5] are very different. A (clear-text) fragment can tell the attacker partial information about the corresponding data object, but a share could tell the attacker nothing about the data object. This nice property of secret-sharing schemes allows PASIS to achieve the same amount of security as [5] without doing any encryption.

Correspondingly, the read and write operations need to be redesigned in PASIS. In particular, when a client reads a file, first, the client will look up, in the directory service, the names of the *n* shares that comprise the file. Second, the client sends read requests to at least *m* of the *n* storage nodes. Third, the client collects the responses. Fourth, the client reconstructs the file. On the other hand, when a client writes a file, the client needs to a set of operations similar to those involved in a read except that the write operation does not complete until at least *n-m+1* (or *m*, whichever is greater) storage nodes have stored their shares.

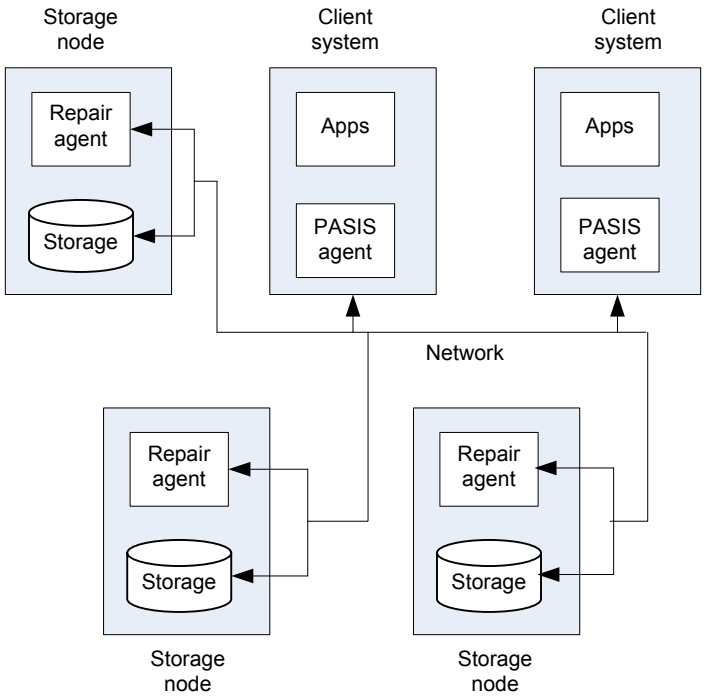

Figure 1. The PASIS architecture. Client systems and storage nodes are attached to the network. Client applications interact with a PASIS storage system through a PASIS agent. Storage devices and repair agents that monitor system status comprise the storage nodes.

In summary, the PASIS architecture illustrates the following ideas in building survivable storage systems. First, use *decentralized storage systems* to partition information among nodes. "Using data distribution and redundancy schemes commonly

associated with disk array systems such as RAID (Redundant Array of Independent Disks) ensures scalable performance and fault tolerance"[31]. "Elimination of single failure points provides a starting point for developing survivable storage systems"[31].

Second, exploit *data redundancy and encoding*. Threshold or secret-sharing schemes provide both information confidentiality and availability. "These schemes encode, replicate, and divide information into multiple pieces (or shares) that can be stored at different storage nodes" [31]. The system can only construct the information when enough shares are available. Third, perform *dynamic self-maintenance*. "Over time, all systems need maintenance. Truly survivable systems automatically perform some self-maintenance includes regular monitoring for potential problems, such as failed or compromised nodes, performance bottlenecks, and denial-of-service attacks" [31].

Finally, accomplish good trade-offs among information confidentiality, information availability, and storage requirements. Different threshold schemes will yield different confidentiality, availability, and storage requirements tradeoffs. For example, as n increases, information availability increases (it's more probable that m shares are available), but the storage required for the information increases (more shares are stored) and confidentiality decreases (there are more shared to steal).

Intrusion Masking Distributed Computing

As businesses and applications are becoming more and more distributed (on the Internet), distributed computing and distributed software are more and more popular. To satisfying the IA needs of these distributed businesses and applications, survivable distributed computing is in urgent need. Nevertheless, distribution of computing can make information systems more vulnerable since the hacker will have more choices regarding where and when to enforce the attack; and any local breach may lead to serious global compromise through the interdependencies among distributed operations.

Distributed software is often structured in terms of clients and services. Each service comprises one or more servers and exports operations that clients invoke by making requests. Although using a single centralized server is the simplest way to implement a service, the resulted service can only be as secure and reliable as that server. If this level of fault and intrusion tolerance is unacceptable, then multiple servers that fail independently must be used. Usually, replicas of a single server are executed on separate processors of a distributed system, and protocols are used to coordinate client interactions with these replicas. Moreover, to make a replicated system more resilient to attacks, various *diversifying* technologies such as diverse operating systems can be used.

The *state machine (replication) approach* [24] is a general method for implementing an intrusion-masking service by replicating servers and coordinating client interactions with server replicas. In this approach, an intrusion-masking server (modeled as a state machine) is implemented by replicating that server (i.e., both services and data) and running a (server) *replica* on each of the nodes in a distributed system. In the state machine approach, given the same sequence of requests (from probably a set of clients) to each replica, a group of non-faulty replicas which start consistent (i.e., having the same state) will remain consistent (after the sequence of requests are processed). Hence, when a group of server replicas is serving a set of clients, if the requests of the clients can be delivered to the replicas in such a way that the same sequence of requests will always

be received by each replica, then if the group has *2t+1* replicas, it can *mask t* intruded replicas, since each client can use majority voting to identify both the correct and the malicious responses.

Ensuring that the same sequence of requests (i.e., the same messages and the same order) will be delivered to each replica is, however, fairly difficult, due to the complexities of the networking environment and the fact that any node or (communication) link in a distributed system could be faulty or vulnerable. For one example, if we let a replica be the (designated) *sender* that transmits the clients' messages (or requests) to the other replicas, then if the sender is faulty, then the group of replicas can receive inconsistent requests. On the other hand, even if the sender is not faulty, communication failures can still cause replicas to receive inconsistent or differently-ordered requests. For another example, if we let each client directly send its requests to each replica, then even if nothing is faulty, two replicas could receive two requests from two clients, respectively, in different orders, due to such reasons as delay and competition. According to [24], two requirements need to be satisfied to achieve this goal: (a) *Consistency*. Every non-faulty server replica receives every request. (b) *Total Order*. Every non-faulty replica processes the requests it receives in the same relative order. Developing the protocols that can satisfy these two requirements has raised a tremendous amount of interests, and fortunately as a result, a family of reliable totally ordered group communication services which can satisfy the two requirements are developed (e.g., [33]). And these protocols (or services) have naturally become a key component of a typical implementation of the state machine approach. Since the state machine approach can handle arbitrary Byzantine faults such as software bugs, operator mistakes and malicious attacks, the corresponding systems (services) built using this approach are also called Byzantine fault tolerant distributed systems (services). Nevertheless, to guarantee correctness and security, in general these reliable totally ordered group communication services require that less than 1/3 of replica group members be faulty. Therefore, a survivable distribute computing system built using the state machine approach actually uses *3k+1* or more server replicas to mask *k* intruded replicas.

Although many solutions for state machine replication have been proposed, state machine replication has not been widely deployed in the real world primarily due to the fact that most of these solutions have significant performance overhead (another reason might be that once one replica is compromised, it is fairly easy to compromise the remaining replicas). To mitigate this problem, in [4] a new practical algorithm for state machine replication called *BFT* is proposed, which can be used to build highly-survivable systems that tolerate Byzantine faults. BFT shows how to build Byzantine-fault-tolerant systems that can be used in practice to implement real services because they do not rely on unrealistic assumptions and they perform well. Many existing solutions for state machine replication rely on *synchrony* assumption for correctness, i.e., rely on know bounds on message delays and process speeds, which is dangerous in the presence of malicious attacks. An attacker may compromise the safety of a service by delaying non-faulty nodes or the communication between them until they are tagged as faulty and excluded from the replica group. Such a denial-of-service attack is generally easier than gaining control over a non-faulty node.

In contrast, BFT works in *asynchronous* environments like the Internet, it incorporates mechanisms to defend against Byzantine-faulty replicas, and it recovers replicas proactively. The recovery mechanism allows the algorithm to tolerate any number of faults over the lifetime of the system provided less than *1/3* of the replicas become faulty within a small window of vulnerability. The window may increase under a denial-of-service attack but the algorithm can detect and respond to such attacks and it can also detect when the state of a replica is corrupted by an attacker. BFT has been implemented as a generic program library with a simple interface. The BFT library provides a complete solution to the problem of building real services that tolerate Byzantine faults. The library is used to implement the first Byzantine-fault-tolerant NFS file system, BFS. The BFT library and BFS perform well because the library incorporates several important optimizations. The most important optimization is the use of symmetric cryptography to authenticate messages. Public-key cryptography, which was the major bottleneck in previous systems, is used only to exchange the symmetric keys. The performance results show that BFS performs 2% faster to 24% slower than production implementations of the NFS protocol that are not replicated. Therefore, the BFT library is believed to be able to be used to build practical systems that tolerate Byzantine faults. Accordingly, in [29] BFT is extended to build practical Byzantine fault tolerant two-phase commit protocols (BFT-2PC) to tolerate both malicious coordinator and malicious participants in distributed transaction processing.

Finally, although BFT is quite efficient when the replica group is relatively small, BFT may not scale well for large groups. To improve scalability, a technology called *Byzantine Quorum* is proposed [17], where quorum replication techniques are applied to achieve Byzantine fault-tolerance in asynchronous systems. This technology may provide more scalability since each operation is processed by only a subset of replicas instead of every replica. Nevertheless, this approach to scalability is fairly expensive: it requires $n > 4f + 1$ to tolerate f faults; each replica needs a copy of the state; and the load of each replica decreases slowly with n (it is $O(1/\sqrt{n})$).

Attack Resistant Certification Authority

In a public key infrastructure, a *certificate* specifies a binding between a name and a set of attributes especially the public key. Certificates are the core of PKI technologies. "Over time, public keys and attributes can change: a private key might be compromised, leading to selection of a new public key, for example. The old binding and any certificate that specifies that binding then become *invalid*". A *certification authority* (CA) attests to the validity of bindings by issuing digitally signed certificates that certify these bindings and by providing a means for clients to check the validity of certificates. With an online CA, principals can check the validity of certificates just before using them" [30]. An online CA needs not only to be secure (the CA's private key cannot be compromised) but also to be available and reliable, which is exactly the goal of survivable CA.

COCA (Cornell certification authority) [30] is a fault-tolerant and secure online certification authority that has been built and deployed both in a local area network and on the Internet. Extremely weak assumptions characterize environments in which COCA's protocols execute correctly: no assumption is made about execution speed and message delivery delays (i.e., COCA assumes asynchrony); channels are expected to

exhibit only intermittent reliability; and with *3t+1* COCA servers up to *t* may be faulty or compromised. These extremely weak assumptions inversely make COCA extremely resilient to malicious attacks. COCA is the first system to integrate a Byzantine quorum system [17] (used to achieve availability and scalability) with threshold cryptography and proactive recovery (used to defend against mobile adversaries which attack, compromise, and control one replica for a limited period of time before moving on to another). In addition to tackling problems associated with combining fault-tolerance and security, COCA develops new proactive recovery protocols, and gives a quantitative evaluation on its cost and effectiveness.

The idea of COCA has four major aspects. First, the CA service is supported by a set of replicated COCA servers, but the private key of the CA service (namely the signing key) is held by no COCA server. Instead, different shares of the key are stored on each of the servers, and threshold cryptography is used to construct signatures on responses and certificates. To sign a message from a client, each COCA server generates a *partial signature* from the message and that server's share of the service private key, some COCA server combines these partial signatures and obtains the signed message. In this way, even when several COCA servers are compromised, the private key will still not be disclosed if the number of compromised servers is below the threshold of the scheme.

Second, to be resilient to server failures and provide more availability, every client request is processed by multiple servers and every certificate is replicated on multiple servers. (Note that COCA supports two types of requests: a Query request retrieves a certificate, while an Update request creates, updates, or invalidates a certificate.) The replication is managed as a dissemination Byzantine quorum system [17], where servers are organized by COCA into sets, called *quorums*, and accordingly, each client request is processed by a quorum instead every replica server. However, since a client making a request cannot authenticate messages from a COCA server (because in COCA clients do not know server public keys) and, therefore, cannot determine whether a quorum of servers has processed that request. COCA let some servers to become *delegates* for each request. A delegate presides over the processing of a client request and assembles the needed partial signatures from other COCA servers. A client request is handled by *t+1* delegates to ensure that at least one of the delegates is correct.

Figure 2 gives a high-level overview of how COCA operates by depicting one of the *t+1* delegates and the quorum of servers working with that delegate to handle a client request. The figure shows a client making its request by sending a signed message to *t+1* COCA servers. Each server that receives this message assumes the role of delegate for the request. A delegate engages a quorum of servers to handle the request and constructs a response to the request based on the responses (i.e., partial signatures) received from that quorum. The delegate then assembles these responses into a signature signed by the CA service. After receiving this signature, the client checks that the response is correctly signed by the service and incorrectly signed responses will be discarded.

Third, a mobile adversary might compromise *t+1* servers over a period of time and, in doing so, collect the *t+1* shares of the service private key. To counter this attack, COCA employs a proactive secret-sharing (PSS) protocol to refresh these shares, periodically generating a new set of shares for the service private key and deleting the old set. New

shares cannot be combined with old shares to construct signatures. Fourth, to support the asynchrony assumption, COCA develops an asynchronous PSS protocol.

COCA is motivated by an earlier work denoted Ω ("Omega") [23]. Ω is a survivable key management service for open networks whose goal is to provide flexible and powerful interfaces to meet the demands of an ever widening range of applications. Ω provides the flexibility of an on-line server without incurring the fault-tolerance or security vulnerabilities usually associated with such servers. Ω is built on top of the Byzantine quorum technology, but it does not involve threshold cryptography. Finally, besides distributed survivable CA such as COCA and Ω, in [7] a centralized attack resilient CA called ARECA is proposed which is built on the top of threshold cryptography and a new *two phase signature generation* technique.

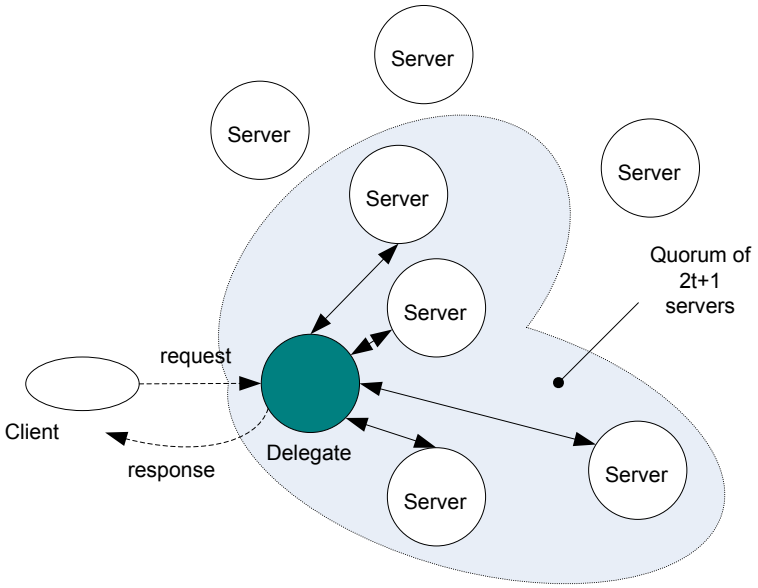

Figure 2. Overview of client request processing in COCA.

Survivable Network Systems

In [3] resilient overlay-network architectures are addressed. "A Resilient Overlay Network (RON) is an architecture that allows distributed Internet applications to detect and recover from path outages and periods of degraded performance within several seconds, improving over today's wide-area routing protocols that take at least several minutes to recover" [3]. "A RON is an application-layer overlay on top of the existing Internet routing substrate. The RON nodes monitor the functioning and quality of the Internet paths among themselves, and use this information to decide whether to route packets directly over the Internet or by way of other RON nodes, optimizing application-specific routing metrics" [3]. It is clear that a RON can mask such attacks as distributed denial-of-service attacks and attacks on routers and physical communication links.

Results from two sets of measurements of a working RON deployed at sites scattered across the Internet demonstrate the benefits of the RON architecture. For instance, over a 64-hour sampling period in March 2001 across a twelve-node RON, there were 32 significant outages, each lasting over thirty minutes, over the 132 measured paths. RON's routing mechanism was able to detect, recover, route around all of them, in less than

twenty seconds on average, showing that its methods for fault (and intrusion) detection and recovery work well at discovering alternate paths in the Internet. Furthermore, RON was able to improve the loss rate, latency, or throughput perceived by data transfers; for example, about 5% of the transfers doubled their TCP throughput and 5% of transfers saw their loss probability reduced by 0.05.

RON node is sufficient to overcome faults or intrusions and improve performance in most cases. These improvements, particularly in the area of fault detection and recovery, demonstrate the benefits of moving some of the control over routing into the hands of end-systems.

DEFENSE-IN-DEPTH TECHNOLOGIES

Compared with intrusion masking technologies, where many attacks may be masked without causing any system security (e.g., integrity and availability) degradation, defense-in-depth technologies usually would introduce certain level of system security degradation. On the other hand, the advantage of defense-in-depth technologies is that (a) they do not require the system be redesigned and can be directly applied to legacy systems, and (b) their overhead is typically smaller than intrusion masking technologies.

The key issues and problems in developing defense-in-depth technologies are:

- How to quickly *contain/isolate* the intrusions so that their infection will not be too serious to operate through?
- How to quickly distinguish the damaged part for the undamaged part of the system?
- How to quickly repair the contaminated part of the system without bringing it offline?
- How to handle the impact of false alarms, undetected intrusions, and the detection latency?
- How to make the intrusion response facilities adaptive and proactive?
- How to validate the cost-effectiveness of defense-in-depth technologies?

In the rest of this section, we will break the possible defense-in-depth technologies into phases of in-depth defense in the first sub-section. In the second sub-section, we use an intrusion tolerant database system to illustrate some important techniques of defense-in-depth.

Phases of Defense-in-Depth

In the literature, defense-in-depth is usually referred to as Information Warfare Defense. Information warfare defense does everything possible to prevent attacks from succeeding, but it also assumes that not all attacks will be averted at the outset. This places increased emphasis on the ability to live through and recover from successful attacks. Information warfare defense must consider all phases of the attack and recovery process. These phases, and the activities that occur in each, are proposed in [1] and quoted as follows.

- *Prevention.* The defender puts protective measures in place.
- *Intelligence gathering.* The attacker observes the system to determine its vulnerabilities and find the most critical functions or data to target.

- *Attack.* The attacker carries out the resulting attack plan.
- *Detection.* The defender observes symptoms of a problem and determines that an attack may have taken place or be in progress.
- *Containment.* The defender takes immediate action to eliminate the attacker's access to the system and to isolate or contain the problem, preventing it from spreading further.
- *Damage assessment.* The defender determines the extent of the problem, including failed functions and corrupted data.
- *Reconfiguration.* The defender may reconfigure to allow continued operation in a degraded mode while recovery proceeds.
- *Repair.* The defender recovers corrupted or lost data and repairs or reinstalls failed system functions to reestablish normal operations.
- *Fault treatment.* To the greatest extent possible, the defender identifies weaknesses exploited in the attack and takes steps to prevent a recurrence.

Some phases - such as prevention, intelligence gathering, detection, containment, reconfiguration, and repair - lend themselves to automated mechanisms and support within the system being attacked. Others, such as fault treatment and some aspects of damage assessment, typically require human intervention.

It should be noticed that the above phases are motivated by the life cycle of fault tolerance. Fault tolerance is a natural approach for dealing with information attacks because it is designed to address system loss, compromise, and damage during operation. Traditional fault-tolerance-approach phases include detection, containment, adaptation, and recovery. Fault semantics for information attacks differ from the traditional fault-tolerance model because in such cases faults are intentionally introduced and malicious, and some attacks may be disguised to appear like normal operations. Therefore, semantics for countermeasures must differ correspondingly, as we mentioned in Section 2.

The information warfare defender's goal is to keep the system operating to support as much critical processing as possible, even if the system is contaminated (or infected) by an attack. One way to ensure continued service is to explicitly address integrity losses caused to the systems in the presence of information warfare attacks. To some degree, real systems lack integrity most of the time. These integrity losses do not always prevent the systems from achieving their critical objectives. The challenge in information warfare is to anticipate acceptable integrity losses and design systems to operate in these degraded modes.

Survivable Database Systems

Existing survivable database technologies can be roughly broken into two categories: *transaction-based* database survivability [2,9,38], which enables a database to operate through attacks via identifying and "rolling back" malicious and affected transactions, and *data-object-based* database survivability [39], which enables a database to operate through attacks via identifying and repairing each corrupted data object. In this section,

we use ITDB (Intrusion Tolerant Data Base), a transaction-based survivable database framework, to illustrate the design principles of survivable database systems.

ITDB [2,9,11,13,14] is a transaction-level self-healing database framework. Since preventing malicious transaction from being executed is in general not a realistic solution, ITDB focuses on how to enable a database to *heal* itself under sustained malicious-transaction attacks in such a way that the database can continue delivering (to a large extent valid) transaction-processing services in the face of such attacks.

ITDB focuses on the intrusions enforced by authorized but malicious transactions. ITDB views a database as a set of data *objects*. At a moment, the *state* of the database is determined by the values of these objects. The database is accessed by transactions for the ACID properties. A *transaction* is a partial order of *read* and *write* operations that either *commits* or *aborts*. The execution of a transaction usually transforms the database for one state to another. Moreover, ITDB models the (usually concurrent) execution of a set of transactions by a structure called a *history*.

ITDB focuses on the *damage* caused by malicious, committed transactions. Since an *active*, malicious transaction will not cause any damage before it commits (due to the *atomicity* property), it is theoretically true that if we can detect every malicious transaction before it commits, then we can roll back the transaction before it causes any damage. However, this "perfect" solution is not practical for two reasons. First, transaction execution is, in general, much quicker than detection, and slowing down transaction execution can cause very serious denial-of-service. For example, Microsoft SQL Server can execute over 1000 (TPC-C) transactions within one second, while the average anomaly *detection latency* is typically in the scale of minutes or seconds (due to the difficulty of anomaly detection).

Hence ITDB is motivated by the following practical goal: "After the database is damaged, locate the damaged part and repair it as soon as possible, so that the database can continue being useful in the face of attacks." In other words, ITDB wants to provide sustained levels of data integrity and availability to applications in the face of attacks. The major components of ITDB are shown in Figure 3. Note that all operations of ITDB are on-the-fly without blocking the execution of (most) normal user transactions. The job of the *Intrusion Detector* is to identify malicious transactions. In the rest of this section, we give an overview of the jobs that the other ITDB components do.

The complexity of ITDB is mainly caused by a phenomenon called *damage spreading*. In a database, the results of one transaction can affect the execution of some other transactions. Informally, when a transaction T_i reads an object x updated by another transaction T_j, T_i is directly *affected* by T_j. If a third transaction T_k is affected by T_i, but not directly affected by T_j, T_k is indirectly affected by T_j. It is easy to see that when a (relatively old) transaction B_i that updates x is identified malicious, the damage on x can spread to every object updated by a *good* transaction that is affected by B_i, directly or indirectly. The job of the *Damage Assessor* is to identify every affected good transaction. The job of the *Damage Repairer* is to recover the database from the damage caused on the objects updated by malicious transactions as well as affected good transactions. In particular, when an affected transaction is located, the Damage Repairer builds a specific *cleaning* transaction to clean each object updated by the transaction (and not cleaned yet).

Cleaning an object is simply done by restoring the value of the object to its latest undamaged version. This job gets even more difficult as the execution of new transactions continues because the damage can spread to new transactions and cleaned objects can be re-damaged. Therefore, the main objective of ITDB is to guarantee that damage spreading is (dynamically) controlled in such a way that the database will not be damaged to a degree that is useless.

The developers of ITDB believe the single most challenging problem in developing practical, cost-effective self-healing database systems (e.g., ITDB) is that during the detection latency, a tremendous amount of damage spreading can be caused. This is because of the fact that intrusion detection is in general much slower than transaction processing. So when a malicious transaction is detected, a lot of affected transactions may have already been committed. Therefore, a practical, cost-effective self-healing database system must be able to live with substantially longer detection latency relative to transaction processing.

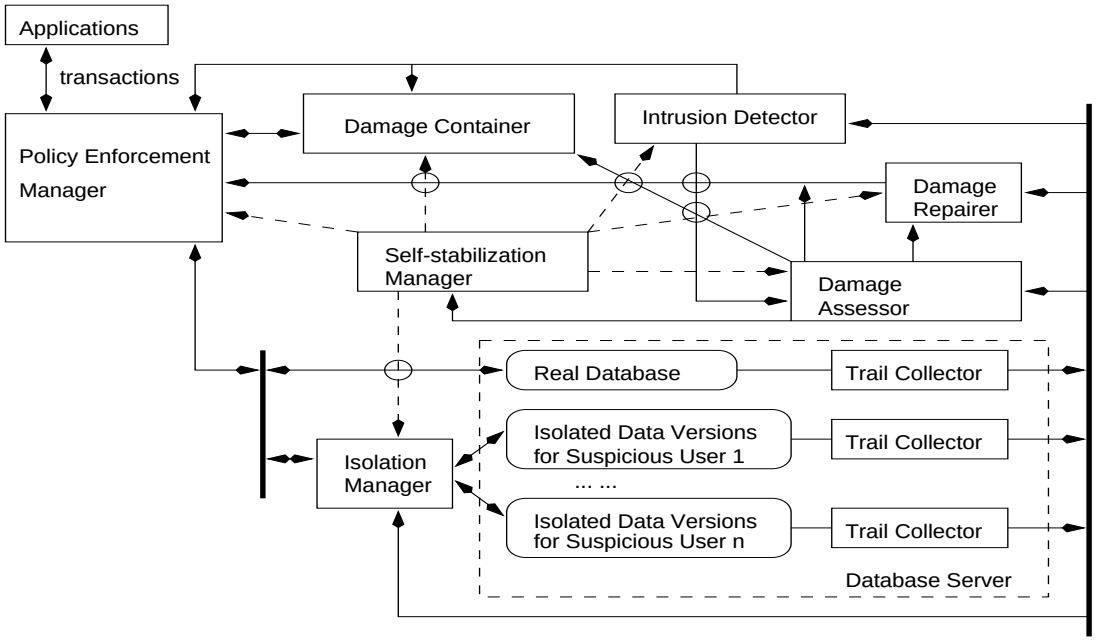

Figure 3. The ITDB Architecture

A unique technical contribution of ITDB is that it can live with long detection latency without suffering serious damage spreading. Allowing long detection latency not only lowers ITDB's requirement on detection agility, but also indirectly lowers ITDB's requirements on detection rate and false alarm rate, because in many cases, longer detection latency can lead to higher detection rate and lower false alarm rate.

However, living with long detection latency is not an easy task. In ITDB, the impact of detection latency is three folds: (1) during the detection latency, the damage can spread to many objects; (2) a significant *assessment latency* can be caused, and during the assessment latency the damage can further spread to many more objects; (3) significant assessment latency can cause ineffective - to some degree at least - damage containment. These three aspects of impact can cause the database to be too damaged to be useful. The job of the Isolation Manager and the Damage Container is to mitigate this impact.

It is easy to see that if every malicious transaction B_i can be identified just after it commits, very little damage can spread, and damage assessment can be done quickly. However, with significant detection latency, when B_i is identified, in the history there can already be many good transactions following B_i and many of them may have already been affected by B_i. Damage assessment at this situation can cause substantial delay, since as shown in [2] damage assessment can spend substantial computation time to scan a long sub-history log for identifying the affected transactions.

Significant assessment latency could cause ineffective damage confinement. At the first glance, it seems that to prevent damage spreading during repair, containing the damage that is located by the Damage Assessor is a good idea. However, in this approach damage will not be contained until an object is identified by the Damage Assessor as damaged. Hence damage containment depends on damage assessment. As a result, when there is a significant latency in locating a damaged object x, during the latency many new transactions may read x and spread the damage on x to the objects updated by them. As a result, when x is confined many other objects may have already been damaged, and the situation can feed upon itself and become worse because as the damage spreads the assessment latency could become even longer. This clearly contradicts with our original motivation of damage containment.

ITDB tackles the three challenges through two novel techniques: attack isolation and multiphase damage containment. Although working towards the same goal, the Isolation Manager and the Damage Container take two very different approaches. And these two approaches compensate each other. In particular, Damage Container takes a novel *multi-phase* damage containment approach which first instantly contains the damage that might have been caused by an intrusion as soon as the intrusion is identified, then tries to uncontain the objects that are previously contained by mistake. Multi-phase damage containment can ensure that no damage will spread during the assessment latency, although with some availability lost. However, Damage Container can do nothing to reduce the damage caused during the detection latency. In contrast, the Isolate Manager can reduce the damage caused during the detection latency (thus it indirectly reduces the damage caused during the assessment latency) by isolating the execution of a *suspicious* transaction that is very likely to cause damage later on. Isolation immunizes the database from the damage caused by the set of suspicious transactions without sacrificing substantial availability, since if an isolated user turns out to be innocent, most - if not all - of his or her updates can be merged back to the real database.

Finally, the job of the *Self-Stabilization Manager* (SSM) is to dynamically reconfigure the other ITDB components based on (a) the current attacks, (b) the current workload, (c) the current system state, and (d) the current defense *behavior* of ITDB, in such a way that stabilized levels of data integrity and availability can be provided to applications in a cost-effective way. Factors (a), (b), and (c) are called the *environment* of ITDB. And the jobs of the *Policy Enforcement Manager* (PEM) are to proxy user transactions and enforce system-wide intrusion tolerance policies. For example, a policy may require the PEM to reject every new transaction submitted by a user as soon as the Intrusion Detector finds that a malicious transaction is executed by the user. Intrusion response policies and security manager interfaces are certainly a crucial aspect of ITDB.

In summary, ITDB develops a family of novel defense-in-depth techniques, such as on-the-fly attack recovery, multiphase damage containment, attack isolation, and rule-based adaptive intrusion-tolerance.

From the self-healing perspective, ITDB considers recovery from malicious but committed transactions. Traditional recovery mechanisms do not address this problem, except for complete *rollbacks* to a previous *checkpoint*, which undo the work of good transactions as well as malicious ones, and compensating transactions, whose utility depends on application semantics. ITDB develops two attack recovery mechanisms: on-the-fly repair and history rewriting. For on-the-fly repair, instead of rolling back the database to the latest checkpoint, the *write-read dependency* between transactions are analyzed on-the-fly to determine which good transactions are affected by a bad transaction, directly or indirectly. Then a specific *cleaning transaction* is composed to clean the infection caused by each malicious or affected transaction, and the concurrency control algorithm is adapted in such a way that new user transactions can be executed simultaneously together with cleaning transactions without affecting the correctness of repair. Finally, when an on-the-fly repair terminates with all the damage repaired, ITDB can detect the termination in a timely manner.

For history rewriting, ITDB rewrites execution histories for the purpose of backing out malicious transactions [14]. Good transactions that are a directly or indirectly, by malicious transactions complicate the process of backing out undesirable transactions. The prefix of a rewritten history produced by the algorithm serializes exactly the set of unaffected good transactions. The suffix of the rewritten history includes special state information to describe a good transactions as well as malicious transactions. ITDB can extract additional good transactions from this latter part of a rewritten history. The latter processing saves more good transactions than is possible with a dependency-graph based approach to recovery.

Finally, in [28], a self-healing workflow system is proposed. Compared with ITDB, [28] considers more types of dependency relations and introduces a set of theorems to trace damage spreading, construct repairing tasks, and create execution orders between recovery and normal tasks in such a way that correct, on-the-fly workflow attack recovery can be achieved.

CONCLUSION

As society increasingly relies on digitally stored and accessed information, applications have increasingly higher requirements on supporting the availability, integrity, and confidentiality of this information, and traditional information security technologies are increasingly limited in satisfying the security requirements of applications due to their inability to survive successful attacks. As a result, Information Assurance technologies are introduced to not only prevent information from being disclosed, modified or destroyed, but also detect intrusions and operate through attacks in such a way that a certain level of information security can be ensured in the presence of attacks. In this article, we survey the natural evolution of information assurance technologies. Three generations of IA technologies are summarized, and the newest generation of IA technologies is discussed in detail. In summary, this article takes the first steps to give a comprehensive overview of the scope of IA technologies, the relation between the

emerging survivability technologies and the more established IA technologies such as information security and intrusion detection technologies, the characteristics of survivability technologies, and the representative ideas, principles and techniques of survivable systems development.

Although a variety of emerging IA technologies have been developed recently to ensure a certain level of information security in the presence of attacks for applications, existing IA technologies are still at their earlier stage and limited in many aspects, and advanced IA technologies have not been widely deployed in the real world so far. Hence, a lot of existing new IA technologies and practices are yet to come. Here we would like to mention several new research and development directions in IA technologies, which are illustrated as follows.

- The *threat* aspect of survivability. Without a tangible and accurate threat model, a highly assured information system cannot be developed. To build a good threat model, both the system's vulnerabilities and the attacks' characteristics (e.g., intent) are crucial. Some preliminary research has been done in analyzing the attacker's intent and strategies (e.g., [10]), but more research is certainly needed. Risk analysis is a relevant topic and readers can refer to Chapter *Risk Assessment and Risk Management* for information on this topic.
- Survivability requirements analysis. Without a clear specification of the users' *survivability requirements*, a survivable system may either over-react to attacks (and threats) or be not proactive enough, and the effectiveness of survivability mechanisms could not be well evaluated. Survivability requirements analysis is a challenging problem, especially when quantitative requirement specifications are expected.
- Survivability *metrics* and measurements. Information Assurance metrics are scarce and qualitative. Given the need to determine the information assurance posture for a given organization under given conditions, users in the field require a means to determine the relative degree of assurance associated with the information assets under their control. Likewise, developers of survivable systems require metrics to measure the degree to which they are employing engineering practices during the system development process. The use of IA metrics would permit establishing trust in a system built from untrusted components, determining sufficient levels of security for the specific tactical situation and condition, and assessing system vulnerabilities. IA metrics enable quantitative tradeoffs between security and performance (degradation).
- *Service survivability*. Existing IA technologies largely focus on system survivability, but in many cases system survivability does not imply service survivability, and additional service survivability facilities and controls are needed. Service survivability are application oriented and at a higher level than system survivability.
- Wireless Information Assurance. A key piece of the large-scale information enterprise is the wireless information assurance segment. Wireless networks must exhibit the same functional and IA attributes as wired networks. They must be protected; attacks against these networks must be detected; specifics of successful attacks must be assessed and finally appropriate responses must be carried out. As we move to more and more wireless components becoming a part of the larger network and as wireless networks proliferate, we need to be aware that these networks, if improperly understood and configured, could provide a "back-door" into our

protected wired enterprise. Intrusion detection for wireless networks must be addressed as well as recovery of wireless services after adversary disruption/denial destruction of friendly networks.

- Using intrusion tolerant middleware [6,22,26] to facilitate the development of intrusion tolerant applications. In this way, developers may be relieved from substantial IA design and development issues.

Acknowledgement

This work was supported by NSF ANI-0335241, NSF CCR-0233324, and Department of Energy Early Career PI Award.

Glossary

Availability (a) The ability to access a specific resource within a specific time frame as defined within the IT product specification. (b) The ability to use or access objects and resources as required. The property relates to the concern that information objects and other system resources are accessible when needed and without undue delay. (c) The prevention of the unauthorized withholding of information or resources.

Assurance (a) The degree of confidence that a target of evaluation adequately fulfills the security requirements. (b) A measure of confidence that the security features and architecture of an automated information system accurately mediate and enforce the security policy. Note: The two main aspects of assurance are effectiveness and correctness or development and evaluation assurance.

Authenticity The ability (a) to establish the validity of a claimed identity; (b) to provide protection against fraudulent transactions by establishing the validity of a message, station, individual, or originator.

Byzantine Fault Tolerance A Byzantine fault is one in which a component of some system not only behaves erroneously, but also fails to behave consistently when interacting with other components. Correctly functioning components of a Byzantine fault tolerant system will be able to reach the same group decisions regardless of Byzantine faulty components.

Certificate A certificate is a document that attests to the truth or ownership of something. A digital certificate is a digital document that serves the same purpose. Most specifically, it attests to the truth that you are who you say you are, and that you own the particular public key specified in the certificate.

Certification Authority (CA) A trusted third party who confirms the identity of an organization or individual (an entity).

Confidentiality Assurance that information is not disclosed to inappropriate entities or processes.

Fault Tolerance The ability of a system or component to continue normal operation despite the presence of hardware or software faults.

Integrity (a) Correctness and appropriateness of the content and/or source of a piece of information. (b) The prevention of the unauthorized modification of information. (c) Sound, unimpaired, or perfect condition.

Intrusion Detection A security service that monitors and analyzes system events to find and provide real-time or near real-time attempt warnings to access system resources in an unauthorized manner. This is the detection of break-ins or break-in attempts, by reviewing logs or other information available on a network.

Non-repudiation An attribute of communications that seeks to prevent future false denial of involvement by either party. *Non-repudiation with proof of origin* provides the recipient of data with evidence that proves the origin of the data.

Survivability The ability of a network computing system to provide essential services in the presence of attacks and failures, and recover full services in a timely manner.

Vulnerability A hardware, firmware, communication, or software flaw that leaves a computer processing system open for potential exploitation, either externally or internally, thereby resulting in risk for the owner, user, or manager of the system.

Cross References

References

[1] P. Ammann, S. Jajodia, C.D. McCollum, B.T. Blaustein, "Surviving Information Warfare Attacks on Databases", *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1997, pages 164-174.

[2] P. Ammann, S. Jajodia, P. Liu, "Recovery from Malicious Transactions", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 5, September 2002, pages 1167-1185.

[3] D. G. Anderson, H. Balakrishnan, M. F. Kaashoek, R. Morris, "Resilient Overlay Networks", *Proc. 18th ACM Symposium on Operating Systems Principles*, 2001.

[4] M. Castro and B. Liskov, "Practical Byzatine Fault Tolerance", Proc. OSDI, 1999.

[5] Yves Deswarte, Laurent Blain, Jean-Charles Fabre, "Intrustion Tolerance in Distributed Computing Systems", *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1991, pages 110-121.

[6] T. Courtney, J. Lyons, H. V. Ramasamy, W. H. Sanders, M. Seri, M. Atighetchi, P. Rubel, C. Jones, F. Webber, P. Pal, R. Watro, M. Cukier, J. Gossett, "Providing Intrusion Tolerance with ITUA", *Supplemental Volume of the 2002 International Conference on Dependable Systems & Networks (DSN-2002)*, Washington, DC, June 23-26, 2002, pp. C-5-1 to C-5-3.

[7] J. Jing, P. Liu, D. G. Feng, J. Xiang, N. Gao, J. Q. Lin, "ARECA: A Highly Attack Resilient Certification Authority", *Proc. ACM First Workshop on Survivable and Self-Regenerative Systems*, 2003.

[8] Wenke Lee, Sal Stolfo, Kui Mok, "A Data Mining Framework for Building Intrusion Detection Models", *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1999

[9] P. Liu, J. Jing, P. Luenam, Y. Wang, L. Li, S. Ingsriswang, "The Design and Implementation of a Self-Healing Database System**",** *Journal of Intelligent Information Systems*, Vol. 23, No. 3, 2004, pages 247-269.

[10] P. Liu, W. Zang, "Incentive-Based Modeling and Inference of Attacker Intent, Objectives and Strategies", *Proc. ACM CCS 2003*, Oct. 28-31, Washington DC, 2003.

[11] P. Liu, "Architectures for Intrusion Tolerant Database Systems", *Proc. 2002 Annual Computer Security Applications Conference*, Dec 2002, pages 311-320.

[12] P. Liu, S. Jajodia, "Multi-Phase Damage Confinement in Database Systems for Intrusion Tolerance", *Proc. 14th IEEE Computer Security Foundations Workshop*, June 11-13, 2001, Nova Scotia, Canada. Pages 191-205.

[13] P. Liu, S. Jajodia, C. D. McCollum, "Intrusion Confinement by Isolation in Information Systems", *Journal of Computer Security*, Vol. 8, No. 4, 2000, pages 243-279.

[14] P. Liu, P. Ammann, S. Jajodia, "Rewriting Histories: Recovery from Malicious Transactions", *Distributed and Parallel Databases, 8(1), 2000, pages 7-40.*

[15] T.F. Lunt, "A Survey of Intrusion Detection Techniques", *Computers & Security, 12(4):405-418, June, 1993.*

[16] P. Luenam, P. Liu, "The Design of an Adaptive Intrusion Tolerant Database System", *Proc. IEEE Workshop on Intrusion Tolerant Systems*, June 2002.

[17] D. Malkhi, M. Reiter, "Byzantine Quorum Systems", *Distributed Computing*, 11:203-213, 1998.

[18] D. Malkhi and M. Reiter, "Secure Execution of Java Applets Using a Remote Playground", *IEEE Transactions on Software Engineering, 26(12), 2000*

[19] D. Malkhi, M. Merritt, M. K. Reiter, G. Taubenfeld, "Objects shared by Byzantine processes", *Distributed Computing* 16(1) 37–48, 2003.

[20] B. Mukherjee, L. T. Heberlein, K.N. Levitt, "Network Intrusion Detection", *IEEE Network*, June 1994, pages 26-41.

[21] P. Ning, Y. Cui, D. S. Reeves, "Constructing Attack Scenarios through Correlation of Intrusion Alerts", *Proc ACM Int'l Conf. on Computer and Communications Security*, 2002.

[22] H. V. Ramasamy, P. Pandey, J. Lyons, M. Cukier, W. H. Sanders, "Quantifying the Cost of Providing Intrusion Tolerance in Group Communication Systems", *Proceedings of the International Conference on Dependable Systems and Networks (DSN-2002)*, Washington, DC, June 23-26, 2002, pp. 229-238.

[23] M. Reiter, M. K. Franklin, J. B. Lacy, R. N. Wright, "The Ω Key Management Service", *Proc. ACM CCS, 1996*.

[24] F. B. Schneider, "Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial", *ACM Computing Surveys, 22(4), 1990*

[25] S. Sekar, M. Bendre, P. Bollineni, "A Fast Automaton-based Method for Detecting Anomalous Program Behaviors", *Proc. IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 2001.

[26] S. Singh, M. Cukier, W. H. Sanders, "Probabilistic Validation of an Intrusion-Tolerant Replication System", *Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN-2003)*, San Francisco, CA, June 22-25, 2003, pp. 615-624.

[27] S. Vutukury, J. J. Garcia-Luna-Aceves, "MDVA: A Distance-Vector Multipath Routing Protocol", *Proc. IEEE INFOCOM, 2001*.

[28] M. Yu, P. Liu, W. Zang, "Self Healing Workflow Systems under Attacks**",** *Proc. 24th IEEE International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[29] M. Yu, P. Liu, W. Zhang, "Intrusion Masking for Distributed Atomic Operations**",** *Proc 2003 IFIP International Information Security Conference* (SEC 03), May 2003

[30] L. Zhou, F. B. Schneider, R. V. Renesse, "COCA: A Secure Distributed Online Certification Authority", *ACM Transactions on Computer Systems*, 20(4), 2002.

[31] Jay Wylie, Michael Bigrigg, John Strunk, Gregory Ganger, Han Kiliccote, Pradeep Khosla, "Survivable Information Storage Systems", *IEEE Computer*, August 2000.

[32] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. A. Longstaff, N. R. Mead, "Survivability: protecting your critical systems", *IEEE Internet Computing*, Volume 3, Issue 6, 1999, pages:55 – 63.

[33] M. Reiter, "Secure agreement protocols: Reliable and atomic group multicast in rampart", In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 60-80, Fairfax, VA, November 1994.

[34] http://www.maftia.org/ (MAFTIA project)

[35] http://www.darpa.mil/ipto/programs/oasis_demval/

[36] H. Debar, M. Dacier, A. Wespi, "Towards a taxonomy of intrusion detection systems", *Computer Network*, 31 (1999) 805-822.

[37] A. Shamir, "How to share a secret," *Communications of the ACM*, Nov. 1979, pp. 612-613.

[38] A. Smirnov, T. Chiueh, "A portable implementation framework for intrusion-resilient database management systems", *Proceedings of the 2004 IEEE International Conference on Dependable Systems and Networks*, 2004.

[39] B. Panda, J. Giordano, "Reconstructing the database after electronic attacks", *Proceedings of the 1998 IFIP WG11.3 Working Conference on Database and Applications Security*, 1998, pages 143-156.