

# Surviving Information Warfare Attacks

In today's heavily networked environment, you must guard against both obvious and subtle intrusions that can delete or corrupt vital data. Ideally, your security measures will allow critical system operation even when you're under attack.

**Sushil Jajodia**

Mitre Corporation and George Mason University

**Paul Ammann**

George Mason University

**Catherine D. McCollum**

Mitre Corporation

The past few years have seen governmental, military, and commercial organizations widely adopt Web-based commercial technologies because of their convenience, ease of use, and ability to take advantage of rapid advances in the commercial market. With this increasing reliance on internetworked computer resources comes an increasing vulnerability to information warfare. Although it can range from psychological operations to physical attacks on information systems, the aspect of information warfare that most concerns computer professionals is defending against the use of computing technology to disrupt or disable the computerized functions and resources that support an organization's operations.

An information warfare attacker's goal is to damage an organization by disrupting its information systems. The specific target of an attack may be the system itself or its data. Although attacks that bring down the system are severe and dramatic, they must be well timed to achieve the attacker's goal, because immediate and concentrated attention will be applied to restoring system operation. That accomplished, a thorough diagnosis will precede installation of measures designed to prevent further such attacks.

More successful in the long run, then, might be attacks that, undetected, install plausible but incorrect information that mislead an organization into making bad decisions—a technique first identified as *storage jamming*.<sup>1</sup> We must recognize that not all attacks—even obvious ones—can be averted at their outset. Attacks that succeed, to some degree at least, are unavoidable. Thus organizations require comprehensive support for identifying and responding to such attacks, as well as plans for living through and recovering from those that are successful.

Although recovery methods have been studied extensively by researchers in the fault tolerance<sup>2,3</sup> and

database areas,<sup>4</sup> these methods do not defend well against information attack. Because such attacks are likely to be coordinated and malicious in nature, we can assume the attacker is familiar with the intricacies of the target system, which makes a worst-case combination of errors or faults the rule rather than the exception. Recovery in this context requires modifying and extending existing techniques and combining them with techniques designed specifically for surviving information attacks.

## TRADITIONAL APPROACHES AND THEIR LIMITATIONS

Although confidentiality, integrity, and availability—which take on increased prominence in information warfare—have long been considered the domain of information systems security, defensive information warfare places a subtly different emphasis on these topics.

Traditionally, information systems security focuses primarily on prevention: putting controls and mechanisms in place that protect confidentiality, integrity, and availability by stopping users from doing bad things. For the most part, these bad things are activities for which the user is unauthorized. However, experience has shown that we cannot be completely successful in preventing problems.<sup>5</sup> Hackers continually surprise us by finding new ways to break into or interfere with systems. Moreover, most mechanisms are powerless against misbehavior by legitimate users who perform functions for which they are authorized: the so-called "insider threat." This problem is two-pronged: not only because we must guard against malicious actions by true insiders, but also because we cannot always distinguish between insiders and outsiders. Networks have eliminated the isolation that once made this distinction clear. Many network-based attacks, such as password sniffing and session hijacking, allow an attacker to masquerade as a legitimate user.

Most mechanisms are powerless against misbehavior by legitimate users who perform functions for which they are authorized.

### Fault tolerance

Fault tolerance models consider two types of errors: *anticipated* and *unanticipated*.<sup>2,3</sup> Anticipated errors allow accurate damage prediction or assessment and let you design specific protection and recovery mechanisms; unanticipated errors do not allow this.

One example of an anticipated error is the loss or duplication of a message, either due to an unreliable communication link or to an attacker interfering with the link. Anticipating link failures can be accomplished by providing redundant links: You can reliably recover from the lost message by resending the message over a redundant channel. Anticipating link interference can be accomplished by a combination of encryption and embedding additional information in the message being sent.

Another anticipated error type is a value out of range during a type conversion—for example, when converting from floating-point to integer values. Recovery can be achieved through the prudent use of exception handlers. Failure to do so can be costly, as demonstrated by the ill-fated maiden flight of the Ariane 5 rocket, lost shortly after take-off due to events that stemmed from an unprotected type conversion.

To recover from anticipated errors, you can use *forward error recovery* methods. Because the errors have been foreseen, either contingency update instructions have been specified or a means of deriving an acceptably correct value formulated.

An example of an unanticipated error is a software bug. The design or code of the software fails to meet its requirements in an unanticipated manner, since otherwise the fault would be fixed.

Unanticipated errors require *backward error recovery* methods, which replace the entire state with a consistent prior state. For example, you can achieve backward recovery by taking checkpoints periodically and logging operations between checkpoints. After a crash, you restore the most recent checkpoint and consult the log to include updates made after the checkpoint was taken.

However, neither approach is ideal. Forward recovery methods have two limitations. First, they are usually very system- and error-specific. Second, their success depends on how accurately damage from faults can be predicted and assessed. On the other hand, the backward recovery approach is suboptimal because it requires significant resources to checkpoint and log, and its implementation often requires that the system be halted temporarily.

### Database management systems mechanisms

Information stored in a database management system has some protection inherently. DBMSs already provide a rich set of integrity and recovery features to

keep data accurate and consistent. Beyond the basic access permissions on modification operations, they support entity and referential integrity constraints, range constraints, concurrency controls, and transaction recovery features. Automatic replication functions assist with keeping data available. These features deal well with many kinds of errors and system failures. However, they have limited effectiveness against an information warfare attacker. Information warfare defense must consider the possibility that authorization controls could be defeated; that an authorized user—through greed, disaffection, or ideology—might become an attacker; or that an attacker might gain the use of a legitimate user's identity and corresponding authorizations. Any of these scenarios might result in the introduction of incorrect or misleading data that corrupts the database. Not only are some DBMS controls ineffectual against this problem, but those intended to maintain consistency among related data may actually help spread the contamination.

Entity and range constraints can ensure that individual data values exist and are legal, but cannot verify that they are reasonable or accurate for the particular entity described. An “information tainting” or storage jamming attack disrupts functions that depend on the database, either by inserting wrong values for particularly critical data or by distorting the overall picture to render aggregates or frequency distributions significantly inaccurate via small changes to many individual items. Referential constraints ensure that interrelationships among entities are maintained, but an attacker could easily make corresponding changes in related data entities.

If cascade or delete rules have been specified for the referential integrity constraints, they may actually assist the attacker, spreading the problem by making the corresponding changes automatically. Concurrency controls ensure only that malicious transactions are properly scheduled along with others. Automated replication helps keep data available in a distributed system in the face of individual system failures, but also serves as an efficient means of spreading erroneous data. Automated replication also incurs significant hardware, software, and system-complexity costs.

DBMS transaction management and recovery features are particularly vulnerable. If a transaction fails or is aborted, the transaction-isolation property supports recovery, in a sense, by ensuring that the transaction can be backed out without affecting other transactions. A malicious transaction, however, would appear to the DBMS like any other transaction and would complete normally. Undo/redo logs support recovery when the system fails with uncompleted transactions in progress, but this feature is ineffective when transactions complete successfully but create bad data.

Suppose that at some time after a malicious transaction has completed and been committed, the bad data it created is discovered. Meanwhile, other innocent transactions may have read the bad data, based their computations on it, and unwittingly written bad data of their own to other items. The only general mechanism available to remove the effects of one or more prior, successfully committed transactions is backward recovery, which rolls the database back to a previously established checkpoint. However, the use of this mechanism poses a dilemma, because the penalty for employing it is the loss of all other, valid work accomplished since the checkpoint was taken.

### INFORMATION WARFARE DEFENSE: A REALISTIC ALTERNATIVE

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, follow:

- *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.

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 transactions. Therefore, semantics for countermeasures must differ correspondingly.

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 damaged by an attack. One way to ensure continued service is to explicitly address integrity losses in the models of systems undergoing information warfare attacks. To some degree, real systems lack integrity most, if not all, of the time. These integrity losses do not keep 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.

### CONSISTENCY IN THE PRESENCE OF DAMAGE

The traditional approach to modeling correct states is to associate a set of invariants or integrity constraints with a system; a system state is considered to be correct if it satisfies these invariants. Figure 1a shows this situation. The states inside the circle are consistent, while those outside it are inconsistent. All actions that map one consistent state to another are acceptable. Any action that leads to a state outside the circle is unacceptable; if it occurs, you must perform a recovery to restore the system to a consistent state.

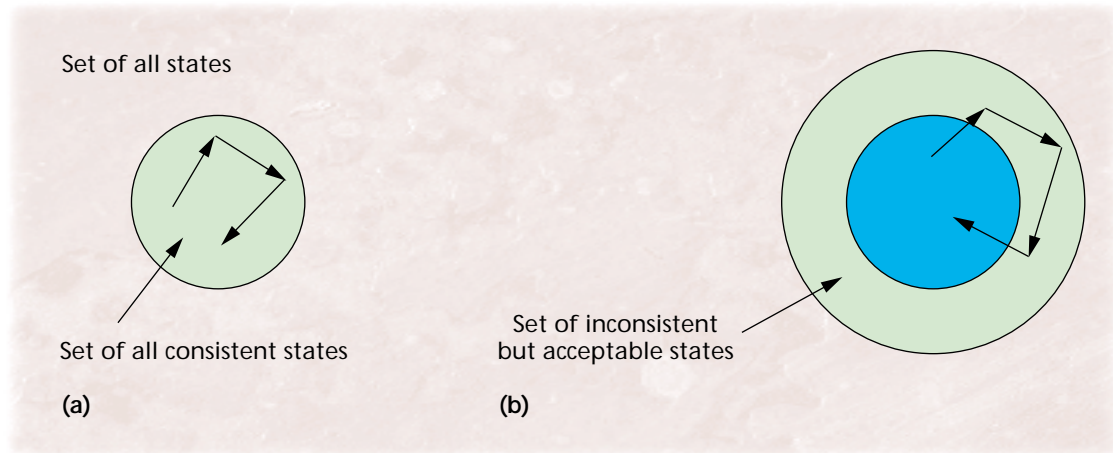
This approach is excessive in environments subject to information attacks. Paul Ammann and colleagues<sup>6</sup> propose that the notion of consistent states needs to be extended so that, when necessary, some but not all deviations from fully consistent states are acceptable, at least temporarily. In the fault tolerance context, Anish Arora and Sandeep Kulkarni propose a closely related model for self-stabilizing systems<sup>7</sup> similar to the model in Figure 1b. As in Figure 1a, the inner circle contains consistent states. States outside the inner circle but within the outer one are “close” to being consistent and thus still acceptable. Only the states that lie outside both circles are always unacceptable.

### Acceptable error example

To see how the extended notion of consistency can increase system availability, consider a database exam-

The information warfare defender's goal is to keep the system operating to support as much critical processing as possible.

Figure 1.  
Classification  
of states.



ple, taken from Ammann's work,<sup>6</sup> that represents aircraft being refueled by aerial tankers. As Table 1 shows, the database consists of the Aircraft Refueling and Tanker Status subtables. For our example, we focus on five aircraft: Aircraft 1, 2, and 3 need refueling, which Tankers 1 and 2 will provide. Tanker 1 is scheduled to fill Aircraft 1's need for 2,000 pounds of fuel and Aircraft 2's need for 3,000 pounds. Aircraft 3, which requires 4,000 pounds of fuel, is scheduled to be refueled by Tanker 2. Tanker 1 has 3,000 pounds of spare capacity; Tanker 2 has 9,000 pounds.

Now suppose we suspect that the pilot name "Cooper" in the Aircraft 3 tuple is incorrect. The incorrect value may actually violate an integrity constraint: Cooper may, for example, be simultaneously assigned to a different task. However, the pilot's name is clearly not critical to the refueling task. The incorrect value may have been entered by a malicious user or by an authorized user who lacks the correct information. Even though such data items must be considered damaged, you may not need or be able to determine the correct value quickly.

If the database is critical and urgently needed, it may be worse to stop and fix the data than to allow its use. That is, we may still wish to allow access to such items, especially if the cost of prohibiting access outweighs that of allowing it, auditing the spread of damage, and subsequently recovering. Allowing use of such values permits transactions that read the damaged field. As Figure 1b shows, this corresponds to permitting executions in states that are known to be inconsistent, but are considered safe.

The acceptable use of damaged values is not always straightforward. In the aircraft refueling example, suppose an aircraft is lost. In this case, it is vital to know exactly which personnel are on the aircraft. Moreover, use of the damaged value clearly requires auditing, because it may be necessary to take some corrective action when the damaged values are finally updated.

As another example, suppose we detect a malicious

change from 9,000 pounds to 50,000 pounds in the Spare Capacity value for Tanker 2. Although the correct value for spare capacity is unknown, an experienced officer may be able to scan the entries of the Aircraft Refueling table for Tanker 2 and use knowledge of the maximum fuel requirements for each aircraft to determine a safe value for this entry. If such information were unavailable or unreliable, entering "0" would be a safe alternative that would prevent additional aircraft being assigned to Tanker 2. Because such changes are not necessarily correct, they may violate some integrity constraint.

The goal of a safe value is to allow on-the-fly repairs where the correct value is not known or available, but an approximate value satisfactory to the application can be derived. Such values are temporary replacements for items damaged by the attacker. Backup versions of items are an example of approximate data.

We can formalize the expanded notion of consistency by using invariants or integrity constraints. We define two sets of integrity constraints: the "normal" set the system state should meet, and a relaxed set that allows additional values that, while acceptable, are not necessarily correct with respect to normal constraints. The values allowed by the normal constraints should be a subset of those allowed by the relaxed constraints, as shown in Figure 1b.

### ATTACKS ON AVAILABILITY, SECURITY, AND INTEGRITY

To be useful, information systems must have three properties: availability, security, and integrity. Availability means that the system's resources are sufficiently functional and accessible to meet mission objectives. Security means that system users are confident that malicious parties have not compromised key resources in the system and that it follows mandatory and discretionary policies. Integrity means that the system is both internally consistent and a valid model of the real world.<sup>8</sup>

These properties can be difficult to achieve in systems under normal conditions; achieving them while experiencing an information attack is clearly more difficult. Further, the three properties often conflict. Consequently, these properties must not be regarded as absolute: Insisting on total availability, security, or integrity is infeasible and unnecessary. We need explicit policies, metrics, and algorithms to manage the conflicts and trade-offs between these three goals.

### Attacking availability

Availability attacks can degrade a communication link by slowing it with additional traffic or disabling it completely. Such attacks can also degrade or disable selected functions in a system by introducing viruses or otherwise altering executable code. Viruses have the additional complication of being infectious: Access to a contaminated site results in further contamination. In designing responses to availability attacks, you must decide whether it is possible to isolate end users from the effects of the attack, and in cases where users must be exposed to partial degradation, how much each user is affected.

### Attacking security

Security attacks include those that cause leakage of sensitive data, authentication attacks in which an adversary poses as a legitimate user, and attacks in which a user is misled to believe that system components have been compromised, even in cases where no actual damage or loss has occurred. For example, an adversary may be able to forge a report that an integrity constraint has been violated without actually gaining write access to the information in question.

Effective countermeasures to security attacks must follow three principles: First, you must have an effective method for assessing the actual damage from the attack. Second, you need some metric for compromise to differentiate especially serious attacks from minor ones. Third, you must understand what degree of compromise is tolerable for each specific mission objective.

### Attacking integrity

Attackers can evade integrity checks by supplying bogus data that coincidentally satisfies system constraints or by suspending the constraints while they execute corrupting transactions. Input streams that provide real-world information can be intercepted, modified, or spoofed. Valid but old data can be substituted for current data. An attacker might introduce executables that appear to offer the same interface to users, but instead introduce inconsistencies. Subtle changes can be difficult to detect but disastrous in consequence.

An effective response to integrity attacks requires methods to detect the damage and track its spread, and also external reference sources to restore correct data.

Table 1. Aircraft refueling and tanker status.

Aircraft refueling				Tanker status	
Aircraft	Pilot	Tanker	Refuel (lbs.)	Tanker	Spare capacity (lbs.)
Aircraft 1	Jones	Tanker 1	2,000	Tanker 1	3,000
Aircraft 2	Smith	Tanker 1	3,000	Tanker 2	9,000
Aircraft 3	Cooper	Tanker 2	4,000		

## IMPLEMENTING COUNTERMEASURES

What can you do to improve your ability to withstand and recover from information attacks? The prevention, intelligence gathering, detection, containment, reconfiguration, and repair phases show the most promise for developing effective information warfare defense mechanisms. The solutions that follow are not panaceas, however—each has drawbacks as well as benefits.

### Prevention techniques

Most information security measures are directed at the prevention phase. This phase includes the definition of both discretionary and mandatory access controls, the establishment of user privileges, appropriate system configuration, and the placement of critical system components within a security architecture. This architecture may include operating-system and network-level controls, strong authentication, and firewalls. The prevention phase also addresses careful administration of the system and adequate back-up and checkpointing practices.

### Deception and hiding techniques

During the intelligence-gathering phase the attacker determines how best to attack the system, what its critical functions and data are, and when to mount an attack. John McDermott and David Goldschlag identify some potential counterintelligence techniques for defending against data jamming that, while primarily intended for detection, could also help deceive the attacker and obscure which data values are critical.<sup>1,9</sup> They propose seeding the database with “bogus” values that are not used in ordinary system processing. Implementing such techniques requires balancing the need to deceive intelligence-gathering attackers against the added DBMS complexity required to ensure that the bogus values are not passed to legitimate users and applications.

### Detection techniques

Detecting an attack is prerequisite to dealing with it. Detection techniques include cryptographic, diagnostic, reasonableness, replication, reversal, structural, and timing checks. Agents can use these techniques to discover the existence of an information

Once bad data has been detected, its further spread must be stemmed while you proceed with repairs and allow applications to continue operating.

attack. Sensors can report that an availability attack is disabling parts of a system. Ideally, detection agents yield clues to the type and scope of damage, which in turn might suggest a recovery plan.

Although a body of work in intrusion detection<sup>10</sup> exists, based on either detecting deviations from expected statistical profiles or pattern-matching against known methods of attack, these sources address detection primarily at the operating-system level, with ongoing work to extend coverage to networks of distributed systems. Such work does not yet provide help with intrusion detection at the level of applications such as database management systems. In a DBMS, the problem can be particularly difficult because it involves determining if data inserted into the database is unreasonable or incorrect. However, the following detection techniques appear promising.

- **Bogus values.** The main purpose of McDermott and Goldschlag's bogus values<sup>1,9</sup> is detection. Since legitimate users and applications would never use them, if the bogus values are ever modified (or are modified at any time other than when preplanned updates occur), we can conclude that the system is under attack. This is similar to the networking concept of "honeypots": decoy systems that exist solely to attract and flush out hackers.
- **Attribute classification.** Attributes could be classified by their source and update characteristics, such as expected update frequency. Although some attributes are independent of other values, some will be derivable either exactly or approximately from other attributes in the database, or perhaps in another component of a distributed database. By extending David Clark and David Wilson's integrity validation procedure concept,<sup>8</sup> values and update histories can be checked periodically. Attributes can be compared to derived expected values or to ground truth, and the frequency of recent updates can be compared against expectations. Care must be taken in interpreting results, however, since an unusually high level of activity might also indicate a crisis in the system's environment.
- **Data integrity constraints.** A variety of integrity constraints beyond those implemented in today's DBMSs could be applied to detect signs of attack. Integrity constraints could be either static or dynamic. Constraint checking could be added through use of active triggers, either at the time of each transaction or periodically. Higher order integrity constraints, such as checks on the distribution of an attribute's data values, could help

identify gradual distortions to a database. Constraints that require significant computation, however, would be run less often. Work in truth-maintenance systems may be applicable to identifying and correcting problems.

- **End-use integrity checking.** If the behavior of a critical database application is considered a function of the database state, periodic testing to determine whether standard applications behave as expected and within established bounds can be a supplementary approach to data integrity checking. This approach would be most useful in a mission-critical application environment.

### Containment, reconfiguration, and repair techniques

The information warfare defender's goal is to keep the database operating, supporting as much of the system's critical processing as possible even after an attack has damaged it. Thus it's undesirable to use recovery techniques that require halting database operations for repair, because this response may be the attacker's real objective, particularly if the attack causes an interruption at a critical time. Once bad data has been detected, its further spread must be stemmed while you simultaneously proceed with repairs and allow applications to continue operating on unaffected data. Here are some possible techniques for achieving these objectives:

- **Static partitioning.** Designing the database and its applications so that transactions can touch data only in a single region limits the extent to which damage can spread and lets applications that use other partitions proceed normally while one partition is under repair. Since this solution may be impractical for many databases, a more flexible alternative—borrowed from concurrent engineering—is to define regional boundaries, identify triggers or propagated updates that cross those boundaries, and limit the bandwidth or conditions under which data may flow across them.
- **Forward error recovery.** Where the semantics of the application are well defined, compensating transactions can implement forward error recovery by anticipating error scenarios. For items that are wholly replaced periodically through normal processing, the error may be corrected merely by waiting until the next replacement occurs.
- **Backward error recovery.** This technique uses database mechanisms such as the undo/redo log to erase recent transactions and restore the database to a prior consistent state.<sup>4</sup> If particular malicious transactions can be identified, you can define algorithms that trace their descendants, roll them back, and then reapply innocent transactions to reach a state that would have existed

had the malicious transactions never executed. However, this type of algorithm is suboptimal because it requires halting database operations temporarily and logging additional information in the DBMS's transaction processing.

- **Versioning.** Using another concept borrowed from concurrent engineering, you can maintain version trees in which versions serve as inter-transaction checkpoints. This technique allows a relatively graceful restoration of a consistent state. If the current database state were found to be unsound, for example, a different branch could be followed. This type of versioning ties closely to the states of database applications and is, thus far, unproven in an information warfare context.
- **Dynamic containment.** This technique's goal is to identify the damage to be contained and alter access patterns dynamically to achieve the containment.<sup>6</sup> Data are marked when damage to them is detected, with the markings determining how other transactions use the data, which integrity constraints hold, and whether damage propagates.

To mount an effective information warfare defense, you will need to tailor some combination of these techniques to fit your particular system and security needs.

**A**lthough ultimately we must prevent malicious attacks from succeeding, not all attacks can be averted at the outset. Our goal therefore should be to develop an adaptable system that maintains maximum availability even when under attack. At any time, healthy components of the system remain available while damaged components have either limited or no availability. ❖

---

#### Acknowledgment

We thank Joseph V. Giordano of the Air Force Research Laboratory/Rome for his support of this work.

---

#### References

1. J. McDermott and D. Goldschlag, "Storage Jamming," *Database Security IX: Status and Prospects*, D.L. Spooner, S.A. Demurjian, and J.E. Dobson, eds., Chapman & Hall, London, 1996, pp. 365-381.
2. P.A. Lee and T. Anderson, *Fault Tolerance: Principles and Practice, 2nd Ed.*, Springer-Verlag, Vienna, 1990.
3. B. Randell et al., eds., *Predictably Dependable Computing Systems*, Springer-Verlag, Berlin, 1995.
4. J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, San Mateo, Calif., 1993.

5. P.G. Neumann, *Computer-Related Risks*, Addison-Wesley, Reading, Mass., 1995.
6. P. Ammann et al., "Surviving Information Warfare Attacks on Databases," *IEEE Symp. Security and Privacy (SP 97)*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 164-174.
7. A. Arora and S.S. Kulkarni, "Designing Masking Fault-Tolerance via Nonmasking Fault-Tolerance," *IEEE Trans. Software Eng.*, Vol. 24, No. 6, 1998, pp. 435-450.
8. D.D. Clark and D.R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *IEEE Symp. Security and Privacy (SP 87)*, IEEE CS Press, Los Alamitos, Calif., 1987, pp. 184-194.
9. J. McDermott and D. Goldschlag, "Towards a Model of Storage Jamming," *Proc. IEEE Computer Security Foundations Workshop*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 176-185.
10. T.F. Lunt, "A Survey of Intrusion Detection Techniques," *Computers & Security*, Vol. 12, No. 4, 1993, pp. 405-418.

**Sushil Jajodia** is a principal scientist at the Mitre Corp., and professor and chair of the Department of Information and Software Engineering and director of the Center for Secure Information Systems at George Mason University in Fairfax, Virginia. His research interests include information security, temporal databases, and replicated databases. He is the founding co-editor-in-chief of the *Journal of Computer Security* and serves on the editorial boards of *IEEE Concurrency*, *ACM Transactions on Information and Systems Security*, and *International Journal of Cooperative Information Systems*. Jajodia is a senior member of the IEEE and a member of the IEEE Computer Society and the ACM. Jajodia received a PhD in mathematics from the University of Oregon at Eugene.

**Paul Ammann** is an associate professor of information and software engineering at George Mason University. His research interests center on why computing systems fail and what can be done about it. Ammann received an AB in computer science from Dartmouth College, and an MS and a PhD in computer science from the University of Virginia.

**Catherine D. McCollum** is a principal engineer at the Mitre Corp. Over the past decade, she has worked in database and distributed database security, security policies, integrity models, and security of distributed object systems and middleware. Her current research interest is information survivability. McCollum received a BS in applied mathematics from Carnegie Mellon University.

Contact Jajodia at [jajodia@gmu.edu](mailto:jajodia@gmu.edu).