Denial of Service Flaws in SDI Software
An Initial Assessment

Karl A. Nyberg
Greby Corporation
karl@greby.com
P. O. Box 1144
Vienna, VA 22180
703-281-2194

ABSTRACT
This paper provides a tutorial and survey into the denial of service aspect of computer security. Definitions from existing literature are presented, and several categorizations of potential denial of service flaws are provided with examples from actual cases. Methods for providing preventive resistance against denial of service threats as well as mechanisms for detection and recovery from denial of service flaws are presented. The application of these methods and mechanisms with respect to software developed for the Strategic Defense Initiative (SDI) program are discussed.

Introduction
The Strategic Defense Initiative (SDI) is a long-range program within the Department of Defense to address the threat of ballistic missiles directed at the United States. The purpose of this paper is to look at one particular area of computer security to the SDI program, namely that of potential denial of service flaws in the SDI software.

A general overview of computer security is first presented, with primary emphasis on denial of service issues. This overview is followed by categorizations of denial of service flaws according to different strategies. The SDI program is then introduced, followed by a discussion concerning mechanisms for coping with the potential denial of service flaws within SDI software.

Throughout this paper specific terms from the Trusted Computer System Evaluation Criteria (TCSEC) [1] and related documents may be used. At their first introduction in this paper, such terms are italicized to indicate that this specific meaning is intended in this report.

Computer Security
Secure operation of computer systems can be attained through the use of a number of security mechanisms. These can include aspects such as physical security, personnel security, information security, communications security, industrial security, and computer security [2]. Through the remainder of this report, the use of the term computer security will more closely relate to the definition of information security, within the context of a computer system.

To date, the major efforts requiring the use of computer security technology have been general purpose computer systems and communications systems. In those environments, there have been three major concerns [3], [4], and [5]:
- unauthorized disclosure of information
- unauthorized modification of information
- unauthorized denial of service

Descriptions of each of these concerns is provided in the following sections. The first two are given only minor discussion here — the references provided should be consulted for additional information.

Unauthorized Information Disclosure
Unauthorized information disclosure is the release of information (also called an object) that is classified at a particular security level to a subject that does not have the corresponding clearance for that level of security. The path via which the information is disclosed is called a channel. Such a disclosure can be categorized according to the method used to make the disclosure, whether it is an overt or a covert disclosure.

Overt Channel. An overt channel is a channel within a system that is used, or intended to be used for information transfer. Overt channels are usually not susceptible to information disclosure in systems designed with security in mind, as checks and balances (such as a security kernel) are designed in to prevent such activities from occurring. Such a channel can be a mechanism for information disclosure when the system's security policy is not adequately enforced.

Covert Channel. A covert channel is one that allows the disclosure of information in an indirect manner, through mechanisms that were not specifically designed for information transfer. Covert channels take advantage of the availability of shared resources (e.g., disk drives, central processing units) to provide an information flow between two (or more) entities that would otherwise not exchange information. In general, covert channels have a communications bandwidth that is significantly lower than that of overt channels because of the indirect nature of the shared resources that provide the communications channel. Development of secure systems often require a covert channel analysis, and a reduction in the bandwidth on identified channels to some specific threshold.

A common opportunity exists for covert communication between processes on a general purpose timesharing computer resulting from modulated requests for system resources. For example, the load average or disk transfer time of such systems could be used as a channel for communication by parties that are in collusion with one another. Access to specific devices that can be used by only one individual at a time, such as modems, can also be used to signal information to others on such systems.

Unauthorized Information Modification
Unauthorized information modification is the modification of information (or objects) by subjects that are not authorized to perform such modification. Like information disclosure, unauthorized information modification occurs when the system security policy is either inadequate or is not adequately enforced. It may occur through an overt action (in the misuse of a service intended for use in the system) or through a covert action due to laxity or inattention on the part of some portion of the security mechanisms of the system.
Denial of Service

Denial of service has been largely out of the mainstream of computer security literature. Articles such as [7], [8], and [9] treat denial of service with only passing interest, while others, such as [10] do attempt to provide examples and preliminary definitions. Some articles that have addressed the issues directly include [11], [12], [13], and [14] with regard to operating systems, and [15] and the Department of Defense's Trusted Network Interpretation (TNI) of the Trusted Computer Security Evaluation Criteria (TCSEC) [16] with regard to communication networks. These articles are limited to a discussion of "authorized" users of the system denying other similarly authorized users service on the particular system.

Denial of service, much like covert channel forms of information disclosure, depends upon the existence of shared resources for its own existence. In environments in which there are no shared resources, there can be no denial of service, just as there can be no covert channels, for there are no other subjects to deny service to or to communicate with. In [13], denial of service has been defined in terms of shared resources and Maximum Waiting Time (MWT).

A group of authorized users of a specified service is said to deny service to another group of authorized users if the former group makes the specified service unavailable to the latter group for a period of time which exceeds the intended (and advertised) service MWT.

This definition appears to have had significant influence upon the discussion provided in [16], which is repeated here

**Denial of Service** -- The prevention of authorized access to system assets or services, or the delaying of time critical operations.

Denial of service has not been used in the broad context to include physical damage to the computer system or the communication medium. The same viewpoint has been taken in this report as was pointed out in [17] that discussion of such threats need only consider authorized users of the system, rather than external non-users, outside of the Trusted Computing Base (TCB). In addition, this report does not address the potential flaws that might occur as a result of the particular processor, instruction set architecture, or other hardware-related aspects. Failure of the system due to its inability to correctly perform its intended function in the face of conditions outside its specification is also not considered denial of service conditions, but rather incompatibilities in the requirements and/or design. A denial of service flaw is a flaw in either hardware or software, (although this paper is concerned only with those flaws evidenced through software) that would allow a denial of service condition to occur in a particular system in contradiction to its specification. Such flaws may actually prevent others from using the shared resource to such an extent that it appears the resource is not available.

The application of this definition is accomplished by enumerating the various shared resources in a system, and determining the MWTs for those resources. [18] presents the concepts of an authorizing agent (known as the Banker) and an arbiter (known as the Bank) for authenticating prospective users and adjudicating the sharing of resources amongst a network of personal computers. It remains for the security policy (or the resource sharing policy in the quoted example) of the system in question to then determine whether such denial of service is appropriate or not.

**Categorization of the Flaws**

This section provides a categorization of potential denial of service flaws according to three categories:

- **Motive**
- **Time of Introduction**
- **Place of Introduction**

This approach combines those given in [19] and [20], as well as [15] for particular application to network denial of service flaws. The categories are not exclusive or exhaustive, and overlap of the categories is recognized. The categories have been selected in the hope that they might be useful for analyses of computer systems to identify denial of service threats and reduce the risk of such flaws. Known examples of the exploitation of flaws are discussed at relevant points.

Some of the indicated flaws may also have potential for disclosure and modification of information, but for the purposes of this paper are not covered in any further detail.

**Motive**

Denial of service flaws may be introduced in one of two manners: they may be intentional flaws, planned in advance and prepared beforehand, or they may be inadvertent, happening only as a result of some other activity or sequence of activities.

**Intentional**. As an example of an intentional application program used to create a denial of service condition, [21] provides a complete program for allocating all available memory in the UNIX operating system, essentially preventing all other users from loading their programs into memory, hence, from being able to run them. Such a tactic would have little utility other than that of denying service to such other users.

In [3], a list of 88 cases is presented in which intentional computer misuse is related to the computer security of the system in question. Many of these cases are not specifically related to computer security, except that a computer was involved way (e.g., theft of computer system equipment).

**Inadvertent**. An inadvertent or unintentional flaw is one that crops up in a system that is performing its normal function, but in a manner that is inconsistent with the anticipated behavior of the system. These flaws can fall into various categories according to their time of introduction, which is discussed in the following sections. Two of the more notable ARPANET examples, include [22], in which the network was flooded with accidentally propagated status messages, and [23], in which the multiple redundant design of the ARPANET's communication system was undone when the lower-level mechanisms supporting that system were routed through a single communications cable.

**Time of Introduction**

Another categorization of denial of service flaws is the time at which the flaw is introduced into the system. Assuming a standard waterfall model of the software development lifecycle, these stages are: requirements, specification/design, implementation, and maintenance.

Although it might seem unlikely that denial of service flaws could be introduced in the requirements phase of the software development, it is certainly possible. In most general purpose operating systems, there are certain trusted subjects that have the ability or privilege to override the existing security policy or resource allocation mechanisms in order to ensure completion of computational objectives. This requirement allows opportunity for flaws in the system when the principle of least privilege is not adequately applied. That is, when a particular subsystem is given more authority or capabilities than required to carry out a particular task, the opportunity for abuse of that additional authority is increased.

Systems can be designed in which denial of service flaws are included, particularly of the inadvertent variety. As an example of a unintentional design flaw, [24] discusses the exploitation of the CPU scheduling algorithm in the UNIX operating system that can be used
to ensure the availability of CPU time among competing long-term, highly CPU-intensive processes. Although the flaw itself is an inadvertent design flaw, exploitation of the flaw is an intentional activity and does not require collusion between the implementer of the scheduling algorithm and the individual taking advantage of the flaw. An early version of the TOPS20 operating system contained a CPU resource-based scheduler that interacted with the virtual memory scheduler in such a manner as to provide all the processing time to processes that used virtual memory. Once such a process was checkpointed waiting for a memory page to be returned, that paging activity took precedence over other activities scheduled by the CPU scheduler.

The implementation phase of the software development lifecycle provides the greatest exposure to introduction of denial of service flaws. The implementation of misunderstood requirements, specification/design, and poor quality software can all provide significant opportunity for such flaws. These errors may be much easier to hide during this phase than during other phases, due to the complexity and volume of software. A trivial example of such a flaw might be the use of a comparison for a “greater than” condition when an “equality” condition is intended. Such a flaw might work for some set of test cases, and therefore not be noticed as an error.

The maintenance phase of the software development lifecycle ought to be no different than the development phase. However, this phase provides greater opportunity for introduction of flaws, both intentional and inadvertent, due to the pressures often associated with maintenance activities. Often during these times standard policies and procedures are bypassed, correlation with requirements and specifications are ignored, and configuration management and exhaustive (or even minimal) testing is not performed.

Place of Introduction

There are a number of places in a system in which denial of service (and other computer security related) flaws can be introduced. In this section, the potential for introduction of such flaws in the layers (or levels) of the system are described.

The operating system and the security kernel of the operating system are the most obvious places in which flaws can be introduced into a system. Flaws at these locations offer the greatest possibility of exploitation, because it is at these locations the greatest amount of privilege exists. Some of the most common software in the system also exists in this area, thus providing the greatest opportunity for misunderstanding of requirements and specification and inadvertent errors in implementation.

System utilities (communications, file backup, etc.) also provide a significant opportunity for introduction of flaws, due to their somewhat trusted nature. Although not containing as high a potential for introduction of flaws or as high a level of privilege as components of the operating system kernel, such corrupted utilities can cause severe damage to operational capability. As an example, if the software performing the backups did not actually write the files to the backup media, then recovery from any system crash that required restoration of the data from such media would be impossible.

Language processing software (e.g., compilers, assemblers, linkers, etc.) provide another opportunity for introduction of flaws. Such software is usually assumed to work “mostly correctly” because of the quantity of use it has had, and to be without malicious logic due to the critical nature of the need to use such software. However, [25] has shown a number of instances in which such software can be used to insert flaws (of any nature) into other software during compilation. The resulting software serves as a Trojan Horse through which denial of service flaws may remain dormant during testing and thus go undetected.

One of the goals of secure kernel technology is the insulation of applications software from security considerations. By ensuring that the security kernel appropriately enforces the security policy, any software that depends upon services provided by the security kernel, and by the security kernel alone, can be presumed are free of constructs that might violate the security policy. Since the security kernel is responsible for the allocation of the resources of the system, any application program would be unable to circumvent the security policy and deny access to those resources.

The Strategic Defense Initiative

The Strategic Defense Initiative (SDI) is a long-range research effort aimed at identification and exploitation of ballistic missile defense technologies in order to nullify potential enemy threats. The relevant technologies, as identified in [26] include sensors, weapons, computing, and communications. Although the major focus of this report is on the computing and communications technologies, an understanding of the total program is necessary in order to perform a proper analysis of the system.

Although decidedly opposed to the SDI program and therefore somewhat controversial, [27] provides an essential overview of the various software technologies within the SDI program. A proposed software technology plan for SDI [28] has been prepared by the Institute for Defense Analyses, and reviews various software technology programs, both within and without the DoD.

The Threat

SDI is intended to provide defense against the basic threat of a ballistic missile attack. The scenarios in [29] should be consulted for additional information and will not be repeated here. [2] delineates the potential threats as:

- Theft
- Fraud
- Destruction
- Inaccurate Results
- Inadvertent Release
- Interruption of Service

The Defense

The SDI system is unique among weapon systems in development today, not only because of its use of high technology in the sensors and weapons arenas, but also because of its requirements for communication, distributed processing, and distributed database management system capabilities. In addition, the inability to adequately specify the intended behavior of the system, due to the uncertainty of potential attack scenarios, make it impossible to state with any significant level of assurance that the system will work correctly when deployed and operational.

The SDI system is complicated not only in its sheer magnitude, but also in its use of leading-edge technologies. One common thread that runs throughout the system is the distributed aspect of all components, an area in which quite a lot of research is being pursued and in which many unanswered issues remain.

The major subsystems in the SDI system are similar to those of the secure computer network described in [30], and consist of:

- operating system
- communications system
- database system
- other application software

Most denial of service conditions result from the exploitation of a flaw in the operating system's ability to adequately manage the sharing of resources by various application software (whether communications, database, or other). [2] advocates the inclusion of controls within application software for ensuring the security of that software. A practical extension to that thesis would be the inclusion of controls within application software for ensuring no denial of service conditions (e.g., no deadlock).

Coping with Denial of Service Flaws
This section presents a number of approaches to coping with denial of service flaws, in general and specifically within the SDI framework. It follows the categorization in [15], in which the following approaches are presented:

- Resistance
- Detection
- Detection and Recovery

These approaches are similar to the steps for resolving the denial of service problem presented in [11] and those related to network presented in [31].

The following assumptions have been made with regard to the software contained within the SDI program:

- software development according to DoD-STD-2167 (and following)
- software implementation in MIL-STD-1815A (Ada)
- SDI program classified at SECRET (or above)

From initial discussions with various contracting agencies, it appears that all of these guidelines have been ignored in the early research and rapid-prototype preparation of systems for learning purposes. This approach, intended to provide early experience with the various technologies and systems, may make evolution into the desired arena and guidelines more difficult in the future.

Resistance

The strongest defense against a denial of service flaw is a system that is specified, designed, and implemented with prevention of a flaw in mind through the entire software development lifecycle. This section describes the techniques that can be applied in different areas of the lifecycle to provide resistance against such flaws.

System Design. The TCSEC requires the identification (and, where possible, elimination or reduction in bandwidth) of covert channels in a system design. Identification of such channels requires the enumeration of all shared resources among users of a system, similar to the enumeration described in [11]. Once the resources and users have been identified, modification of the system design must be effected to preclude such channels.

In [15], [33], and [31] additional criteria are described, which, although particularly directed at network-related security denial of service issues, also have broader applicability, especially in distributed systems.

Software Development. Approach. As mentioned earlier, it is anticipated that the use of the Defense System Software Development Methodology (MIL-STD-2167) [33] will be required for any SDI software development. Modifications to MIL-STD-2167 are specifically allowed by the standard, and some, designed for use within the SDI program, have already been proposed [34] for tailoring the standard to the use of the Ada language. Use of software metrics during the development process, such as [35], can help to reduce the complexity of the resulting software and thus increase assurance that any potential flaws have been found through other mechanisms, (e.g., code inspections, walk-throughs). A commitment to following required development procedures throughout the entire standard software development process (including the maintenance phase) can help to ensure that no unintentional flaws have been introduced.

Verification. The application of verification techniques in the software development approach provides a mechanism whereby proofs of correctness of the software with respect to its specification may be developed in tandem with the software itself. This approach may include rigorous informal verification techniques, such as presented in [36], or the Cleanroom approach in [37], or the more formal techniques, such as those surveyed in [38] and others.

Unfortunately, the application of formal verification techniques in the development of SDI software is hindered by a number of substantial hurdles. Perhaps the most significant hurdles are the lack of technology and tools for performing formal verification of software written in the Ada language and the magnitude of the SDI effort. A number of these issues including recommendations for resolving these difficulties are presented in [39] and [40], while the Institute for Defense Analyses (IDA) recently held a workshop on SDI Verification, Validation and Evaluation that addressed some of these issues.

Personnel. Another area that affects assurance of the correct behavior of operational software is the selection of the development personnel. In [25], a lucid argument is made for placing the trust in the individual(s) developing the software rather than upon any other external measurements. As mentioned previously, it is anticipated that the SDI program will involve access to classified material, and thereby the program (and perhaps the development) will operate in a closed environment, requiring a certain level of trust be placed in the development personnel. The benefit to this approach is that it will presumably help to reduce the potential for insertion of malicious logic into the system by employing only "trusted" personnel.

Detection

In [41] a tutorial on the analysis and testing of software is presented, while [42] discusses the potential for developing a partitioning of the test cases in order to exercise all groups of input classes for complete coverage of a system's input. [43] provides an extensive reference document on various techniques and additional tools available for performing such analyses. Particular techniques and tools identified in that document that have potential for detection of denial of service flaws include:

- algorithm analysis
- analytic modeling of software designs
- execution time analyzer
- software monitors

The result of most of these forms of analysis is usually the detection of flaws related to information disclosure or modification, rather than denial of service. The application of information flow techniques, such as those performed in covert channel analysis [44] (similar to availability analysis in [14]) appears to provide the potential for detection of available shared resources, as described in general in [11].

Some of the literature surrounding the SDI program has included discussion of a "National Test Bed Facility." The use of such a facility can serve to increase the potential for discovery of denial of service flaws by exercising the software in situations that are presumed comparable to those that could arise during deployment. However, the use of testing techniques cannot possibly include all input conditions and could therefore allow many potential flaw situations to go undetected.

Recovery

Flaws uncovered during either type of analysis or testing can be corrected in the software source. Those flaws that are evidenced during program execution, either during the real time analysis or offline, may not be correctable within a time frame to provide an effective opportunity for remedy. [18] describes a system in which recovery is effected by providing authorities capable of revoking access to specific resources. This could be implemented in the SDI software through the command center from which authorizations for specific resources were adjudicated. [15] provides a number of technological countermeasures for dealing with continued denial of service attacks.

Summary

A number of techniques, technologies, tools, procedures, and methods exist for detection and/or prevention of the introduction of denial of service flaws into software. Their use and application throughout the software development lifecycle can provide
assurance of the absence of such flaws.

The greatest difficulty in addressing the issue of denial of service is in categorization of the potential threats and in allocation of available resources. In the corresponding section of [45], it is noted that much remains to be accomplished in the area of problem formulation. This is just as true today, even more so with respect to the SDI program. Without an adequate understanding and statement of the requirements of the system, assurance of the correct functioning of its software is limited, since the potential threats might not be adequately dealt with in the system specification.

Recovery from occurrences of denial of service still requires a significant amount of research, both in general computer systems research and in the actual design of such systems [32]. This is especially relevant to the SDI program because the size and complexity of the system will make any redesign efforts difficult.

The requirements of the SDI program, particularly the real time needs, the distributed nature of the system, use of advanced technologies, and the overall size of the program, still remain as obstacles to ensuring the absence of such denial of service flaws in the system.

**Glossary**

**Access.** (1) A specific type of interaction between a subject and an object that results in the flow of information from one to the other. (2) The ability and the means necessary to approach, to store or retrieve data, to communicate with, or to make use of any resource of an ADP system.

**Application.** Those portions of a system, including portions of the operating system, that are not responsible for enforcing the security policy.

**Automated Data Processing System (ADPS).** A collection of automated methods, procedures, or techniques united by regulated interaction to form an organized whole. An assembly of computer equipment facilities, personnel, software, and procedures. An aggregation of software and the resources required to support it (ADP equipment, manpower, and facilities).

**Automated Data Processing System (ADPS) Security.** All hardware and software functions, characteristics, and features; operational procedures, accountability procedures, and access controls at the central computer facility and remote terminal facilities; the management constraints; the physical environment; emanations security (EMSEC); and personnel and communications security needed to provide an acceptable level of protection for hardware, software, and sensitive data or material in the system.

**Automated Data Processing (ADP).** Data processing largely performed by automatic means (that is, without human intervention). The branch of science and technology concerned with methods and techniques relating to data processing largely performed by automatic means.

**Bandwidth.** A characteristic of a communication channel that is the amount of information that can be passed through it in a given amount of time, usually expressed in bits per second.

**Category.** A grouping of objects to which a non-hierarchical restrictive label is applied (e.g., proprietary, compartmented information). Subjects must be privileged to access a category.

**Channel.** An information transfer path within a system. May also refer to the mechanism by which the path is effected.

**Closed Security Environment.** An environment that includes those systems in which both of the following conditions hold true:

(a) Application developers (including maintainers) have sufficient clearances and authorizations to provide an acceptable presumption that they have not introduced malicious logic. Sufficient clearance is defined as follows: where the maximum classification of data to be processed is Confidential or below, developers are cleared and authorized to the same level as the most sensitive data; where the maximum classification of data to be processed is Secret or above, developers have at least a Secret clearance.

(b) Configuration control provides sufficient assurance that applications are protected against the introduction of malicious logic prior to and during operation of systems applications.

**Compromise.** A violation of the security system such that an unauthorized disclosure of sensitive information may have occurred.

**Computer Security.** Generally used as a synonym for ADPS Security. Sometimes used to denote the protection of sensitive information while it is within a computer (that is, not on an eye-readable or removable storage medium).

**Configuration Control.** Management of changes made to a system's hardware, software, firmware, and documentation throughout the developmental and operational life of the system.

**Confidentiality.** The property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

**Correctness.** The extent to which a program satisfies its specifications.

**Covert Channel.** A communications channel that allows a process to transfer information that violates the system's security policy. A covert channel typically communicates by exploiting a mechanism not intended to be used for communication. See also: Covert Storage Channel, Covert Timing Channel. Compare Overt Channel.

**Covert Storage Channel.** A covert channel that involves the direct or indirect writing of a storage location by one process and the direct or indirect reading of the storage location by another process. Covert storage channels typically involve a finite resource (e.g., sectors on a disk) that is shared by two subjects at different security levels.

**Covert Timing Channel.** A covert channel in which one process signals information to another by modulating its own use of system resources (e.g., CPU time) in such a way that this manipulation affects the real response time observed by the second process.

**Data Integrity.** The state that exists when computerized data is the same as that in the source documents and has not been exposed to accidental or malicious alteration or destruction.

**Denial of Service.** The prevention of authorized access to system assets or services, or the delaying of time critical operations.

**Domain.** The set of objects that a subject has the ability to access.
Environment. The aggregate of external circumstances, conditions, and objects that affect the development, operation, and maintenance of a system. (See Open Security Environment and Closed Security Environment.)

Exploitable Channel. Any channel that is usable or detectable by subjects external to the Trusted Computing Base.

Flaw. An error of commission, omission, or oversight in a system that allows protection mechanisms to be bypassed.

Flaw Hypothesis Methodology. A system analysis and penetration technique where specifications and documentation for the system are analyzed and then flaws in the system are hypothesized. The list of hypothesized flaws is then prioritized on the basis of the estimated probability that a flaw actually exists and, assuming a flaw does exist, on the ease of exploiting it and on the extent of control or compromise it would provide. The prioritized list is used to direct the actual testing of the system.

Formal Proof. A complete and convincing mathematical argument, presenting the full logical justification for each proof step, for the truth of a theorem or set of theorems. The formal verification process uses formal proofs to show the truth of certain properties of formal specification and for showing that computer programs satisfy their specifications.

Formal Verification. The process of using formal proofs to demonstrate the consistency (design verification) between a formal specification of a system and a formal security policy model or (implementation verification) between the formal specification and its program implementation.

Fraud. The intentional misrepresentation of the truth to deceive the owner. (Most computer crimes fall into the fraud category.)

Industrial Security. All aspects of security required by the federal government of contractors in accordance with the applicable contracts. Sometimes the term is limited to the protection of classified information in the hands of U. S. industry.

Information Security. The result of any system of administrative policies and procedures for identifying, controlling, and protecting from unauthorized disclosure, information whose protection is authorized by executive order or statute.

Integrity Policy. A security policy to prevent unauthorized users from modifying, viz., writing, sensitive information. See also Security Policy.

Least Privilege. This principle requires that each subject in a system be granted the most restrictive set of privileges (or lowest clearance) needed for the performance of authorized tasks. The application of this principle limits the damage that can result from accident, error, or unauthorized use.

Malicious Logic. Hardware, software, or firmware that is intentionally included in a system for the purpose of causing loss or harm.

Object. A passive entity that contains or received information. Access to an object potentially implies access to the information it contains.

Open Security Environment. An environment that includes those systems in which one of the following conditions holds true:

(a) Application developers (including maintainers) do not have sufficient clearance or authorization to provide an acceptable presumption that they have not introduced malicious logic. (See the definition of Closed Security Environment for an explanation of sufficient clearance.)

(b) Configuration control does not provide sufficient assurance that applications are protected against the introduction of malicious logic prior to and during the operation of system applications.

Overt Channel. An overt channel is a path within a system that is designed for the authorized transfer of data.

Personnel Security. The process which determines the maximum level of information or equipment to which a person should be allowed access.

Physical Security. The construction and maintenance of secure containers (including entire buildings) for the storage of sensitive information and / or equipment.

Protection Philosophy. An informal description of the overall design of a system that delineates each of the protection mechanisms employed. A combination (appropriate to the evaluated class) of formal and informal techniques is used to show that the mechanisms are adequate to enforce the security policy.

Process. A program in execution. It is completely characterized by a single current execution point (represented by the machine state) and address space.

Reference Monitor Concept. An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects. See also Security Kernel.

Reliability. The extent to which a system can be expected to perform its intended function with required precision.

Resource. Anything used or consumed while performing a function. The categories of resources are: time, information, objects (information containers), or processors (the ability to use information). Specific examples are: CPU time; terminal connect time; amount of directly-addressable memory; disk space; number of I/O requests per minute, etc.

Security Kernel. The hardware, firmware, and software elements of a Trusted Computing Base (or Network Trusted Computing Base partition) that implement the reference monitor concept. It must mediate all accesses, be protected from modification, and be verifiable as correct.

Security Level. The combination of hierarchical classification and a set of non-hierarchical categories that represents the sensitivity of information.

Security Policy. The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

Sensitive Information. Information that, as determined by a competent authority, must be protected because its unauthorized disclosure, alteration, loss, or destruction will at least cause perceivable damage to someone or something.

Subject. An active entity, generally in the form of a person, process, or device that causes information to flow among objects or changes the system state. Technically, a process / domain pair.

System. An assembly of computer hardware, software, and firmware configured for the purpose of classifying, sorting, calculating, computing, summarizing, transmitting and receiving, storing and retrieving data with a minimum of human intervention.
Trojan Horse. A computer program with an apparently or actually useful function that contains additional (hidden) functions that surreptitiously exploit the legitimate authorizations of the invoking process to the detriment of security. For example, making a "blind copy" of a sensitive file for the creator of the Trojan Horse.

Trusted Computer System. A system that employs sufficient hardware and software integrity measures to allow its use for processing sensitive information.

Trusted Functionality. That which is determined to be correct with respect to some criteria (e.g., as established by a security policy). The functionality shall neither fall short of nor exceed the criteria.

Trusted Computing Base (TCB). The totality of protection mechanisms within a computer system -- including hardware, firmware, and software -- the combination of which is responsible for enforcing a security policy. It creates a basic protection environment and provides additional user services required for a trusted computer system. The ability of a TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct input by system administrative personnel of parameters (e.g., a user's clearance) related to the security policy.

Trusted Software. The software portion of a Trusted Computing Base.

Trusted Subject. A subject that is part of the TCB. It has the ability to violate the security policy, but is trusted not to actually do so.

Verification. The process of comparing two levels of system specification for proper correspondence (e.g., security policy model with top-level specification, TLS with source code, or source code with object code). This process may or may not be automated.

Virus. Malicious software, a form of Trojan Horse, which reproduces itself in other executable code.

Vulnerability. A weakness in the ADPs security procedures, administrative controls, internal controls, etc., that could be exploited to gain unauthorized access to classified / sensitive information.