# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

**A DEMONSTRATION OF THE SUBVERSION THREAT: FACING A CRITICAL RESPONSIBILITY IN THE DEFENSE OF CYBERSPACE**

by

Emory A. Anderson, III

March 2002

| | |
|---|---|
| Thesis Advisor: | Dr. Cynthia Irvine |
| Co-Advisor: | Dr. Roger Schell |

**This thesis was completed in cooperation with the Institute for Information Superiority and Innovation.**

THIS PAGE INTENTIONALLY LEFT BLANK

| REPORT DOCUMENTATION PAGE | | | Form Approved OMB No. 0704-0188 |
|---|---|---|---|
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | |
| **1. AGENCY USE ONLY** *(Leave blank)* | **2. REPORT DATE** March 2002 | **3. REPORT TYPE AND DATES COVERED** Master's Thesis | |
| **4. TITLE AND SUBTITLE**: A Demonstration of the Subversion Threat: Facing a Critical Responsibility in the Defense of Cyberspace | | | **5. FUNDING NUMBERS** |
| **6. AUTHOR(S)** Anderson, Emory A., III | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Naval Postgraduate School Monterey, CA 93943-5000 | | | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)** N/A | | | **10. SPONSORING / MONITORING AGENCY REPORT NUMBER** |
| **11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. | | | |
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT** Approved for public release; distribution is unlimited | | | **12b. DISTRIBUTION CODE** |


**13. ABSTRACT** ***(maximum 200 words)***

This thesis demonstrates that it is reasonably easy to subvert an information system by inserting software artifices that would enable a knowledgeable attacker to obtain total and virtually undetectable control of the system. Recent security incidents are used to show that means, motive, and opportunity exist for an attack of this nature. Subversion is the most attractive option to the professional attacker willing to invest significant time and money to avoid detection and obtain a significant payoff.

The objective here is to raise awareness of the risk posed by subversion so that the decision makers responsible for the security of information systems can make informed decisions. To this end, this work provides a complete demonstration of a subverted system. It is shown how a few lines of code can result in a very significant vulnerability. The responsibility to defend information systems cannot adequately be met without considering this threat.

Addressing this threat gets to the very nature of the security problem, which requires proving the absence of something – namely, a malicious artifice. Several techniques for demonstrating security are shown to be inadequate in the face of this threat. Finally, a solution is presented with a proposal for future work.


| **14. SUBJECT TERMS** System Subversion, Computer Security, Artifice, Verifiable Protection | | | **15. NUMBER OF PAGES** 71 |
|---|---|---|---|
| | | | **16. PRICE CODE** |
| **17. SECURITY CLASSIFICATION OF REPORT** Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified | **19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified | **20. LIMITATION OF ABSTRACT** UL |

**THIS PAGE INTENTIONALLY LEFT BLANK**

**Approved for public release; distribution is unlimited**
**This thesis was completed in cooperation with the Institute for Information Superiority and Innovation.**

**A DEMONSTRATION OF THE SUBVERSION THREAT: FACING A CRITICAL RESPONSIBILITY IN THE DEFENSE OF CYBERSPACE**

Emory A. Anderson, III
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

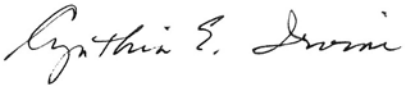
from the

**NAVAL POSTGRADUATE SCHOOL**
**March 2002**

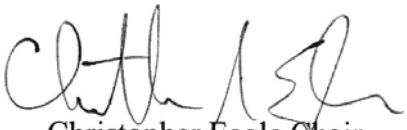
Author:


Emory A. Anderson, III

Approved by:


Dr. Cynthia Irvine, Thesis Advisor


Dr. Roger Schell, Co-Advisor


Christopher Eagle, Chair
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

# ABSTRACT

This thesis demonstrates that it is reasonably easy to subvert an information system by inserting software artifices that would enable a knowledgeable attacker to obtain total and virtually undetectable control of the system. Recent security incidents are used to show that means, motive, and opportunity exist for an attack of this nature. Subversion is the most attractive option to the professional attacker willing to invest significant time and money to avoid detection and obtain a significant payoff.

The objective here is to raise awareness of the risk posed by subversion so that the decision makers responsible for the security of information systems can make informed decisions. To this end, this work provides a complete demonstration of a subverted system. It is shown how a few lines of code can result in a very significant vulnerability. The responsibility to defend information systems cannot adequately be met without considering this threat.

Addressing this threat gets to the very nature of the security problem, which requires proving the absence of something – namely, a malicious artifice. Several techniques for demonstrating security are shown to be inadequate in the face of this threat. Finally, a solution is presented with a proposal for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

## TABLE OF CONTENTS

# LIST OF ACRONYMS

| | |
|---|---|
| CC | Common Criteria |
| COTS | Commercial off-the-shelf |
| CVE | Common Vulnerabilities and Exposures |
| EAL | Evaluated Assurance Level |
| euid | effective user indentifier |
| fsuid | file system user identifier |
| FTLS | Formal Top Level Specification |
| GNU | GNU's Not UNIX |
| GPG | GNU Privacy Guard |
| IP | Internet Protocol |
| IT | Information Technology |
| MD5 | Message Digest 5 hash algorithm |
| NCSC | National Computer Security Center |
| NFS | Network File Server |
| NTFS | Windows NT File System |
| PGP | Pretty Good Privacy |
| PKI | Public Key Infrastructure |
| RM | Reference Monitor |
| RPC | Remote Procedure Call |
| RVM | Reference Validation Mechanism |
| SMB | Server Message Block |
| SSE-CMM | System Security Engineering – Capability and Maturity Model |
| SSL | Secure Sockets Layer |
| ST&E | Security Test and Evaluation |
| TCP | Transmission Control Protocol |
| TCSEC | Trusted Computer Security Evaluation Criteria |
| UDP | User Datagram Protocol |
| UID | User Identifier |
| VFAT | Virtual File Allocation Table |
| VFS | Virtual File System |

THIS PAGE INTENTIONALLY LEFT BLANK

# EXECUTIVE SUMMARY

*"Aesop, who used animals to illustrate human foibles, told of the eagle that died from an arrow, the shaft of which had been feathered with one of its own plumes, to make the point that 'We often give our enemies the means of our own destruction.' "*[1]

Information technology has made possible some of the most astonishing achievements in the history of the modern world. Its usefulness has paved the way for its introduction into nearly every facet of society to the point that we are now inescapably dependent upon it. The ironic fact that that which gives us a significant edge over our adversaries also carries within it one of our most ominous vulnerabilities is now widely accepted and has resulted in an effort known as Critical Infrastructure Protection (CIP). Information security plays an important role in CIP but one significant threat that has been ignored for nearly twenty years is the threat of information system subversion. System subversion involves the clandestine and methodical undermining of a system by planting artifices (trap doors) in it that bypass its security controls. This method of attack is the choice of the professional attacker (such as a nation state or a large company engaging in corporate espionage) who gains access at some point in the system lifecycle and plants the artifice. This artifice grants only those with knowledge of both its presence and how to trigger it, virtually undetectable and potentially unrestricted access to the system. The professional attacker is motivated by the prospect of significant payoff that may not be realized until several years in the future – often at a time when the victim's dependence upon the proper functioning of the system is crucial.

Perhaps the closest relative to subversion is the Trojan Horse attack in which the attacker sends the victim a program in the hopes that the victim will run it on his system. When he does so, the program covertly performs some additional malicious function. There are three primary factors that distinguish this from subversion. First, the Trojan Horse requires a legitimate user to run it while the artifice in subversion does not. Second, the Trojan Horse program exploits the level of privilege associated with the individual that runs it whereas the artifice (typically embedded in the system) simply

[1] From Grant, P. & Riche, R. (1983). The Eagle's Own Plume. Proceeding of the U. S. Naval Institute, July, 1983.

bypasses these privilege checking controls. Finally, the artifice in subversion typically has some type of activation and deactivation mechanism.

Recent trends indicate recognition of the importance of assurance in information systems, but none of the commonly available commercial systems or solutions even approach the level of assurance required to address the subversion threat. Firewalls, intrusion detection systems, encryption, and other similar defense layers all fail in the face of subversion. Moreover, methods used to assess the security of systems such as security test and evaluation (ST&E), red teams (tiger teams), and penetration testing are unable to protect against this threat. The only known way to protect against the subversion threat is with high assurance systems that offer *verifiable* protection. This is not the type of assurance being associated with the majority of modern commercial "high assurance" systems, but the type found in systems designed formally and evaluated through formal audits of the system against a sound set of criteria. Systems built in such a way offer the unique ability to verify that the security mechanisms are both sound and complete even after they have been built.

Press reports over the past year provide ample evidence that the means, motive, and opportunity for an attack of this nature exist. Certainly, decision makers could be considered negligent if they ignore the threat of subversion in systems upon which the defense and critical infrastructure of the United States depend. The computer security community understood the subversion threat as early as the 1970's and by the mid 1980's, vendors were building systems with assurances to defend against it. However, these systems saw little application (even in the Department of Defense) as the society was not yet so inescapably dependent on information technology, the threat was not so mature, and there were strong pressures to use products of the commercial market that favored functionality and features over security and assurance. There has been very little work in this area since this time.

This thesis demonstrates that it is relatively easy to subvert a system (given access at some point during its lifecycle) in a way that can cause significant harm and includes two example artifices (one of which is implemented). Finally, it points out that lessons learned in designing systems to withstand subversion are in danger of being lost and proposes a project to educate and train a new generation of security professionals.

# I. INTRODUCTION

## A. PURPOSE OF STUDY

The purpose of this paper is to heighten awareness of the threat of system subversion and demonstrate that, unless this threat is addressed where appropriate, the responsibility to provide proper security cannot adequately be met. To this end, a brief examination of the threat is provided followed by several real world and publicly reported incidents that together demonstrate the means, motive, and opportunity for an adversary to conduct this type of attack against the United States. Finally, a working example of a subverted operating system is constructed by placing an artifice in the kernel. The effect that this might have on the larger system is demonstrated in the form of a Network File Server, which will allow the attacker who has knowledge of the artifice full access to the server.

The security problem is unique and difficult. It involves proving not only the presence of proper functional security controls, but also proving the absence of anything that can undermine these controls. After-the-fact testing and certification cannot provide the level of protection require to counter subversion. The only way to counter this threat is to build systems in such a way that the insertion of an artifice is both detectable and is guaranteed to be discovered during formal audits of the system against a sound set of criteria. This approach will be presented following the examples.

## B. DEFINITION

Meyers (1980) defines subversion of a computer system as "… the covert and methodical undermining of internal and external controls over a system lifetime to allow unauthorized or undetected access to system resources and/or information." At the time of this writing, an increase in reliance on commercial off-the-shelf (COTS) automated data processing systems was cited as cause for concern over the protection of information of varying degrees of sensitivity in the presence of multiple users with varying levels of trustworthiness. The contemporaneous solution to this "classical security problem" was the security kernel – a small analyzable portion of a trusted operating system that was protected and responsible for enforcing security policy on the system. It was noted that

unless the threat of subversion was addressed, this security kernel technology could never hope to provide a solution to the security problem.

The environment has changed significantly since 1980. In many ways, Meyers foretold the environment that exists today. Products are vastly more complex and interrelated making it much easier to hide artifices (Anderson, 1972). Systems (such as web servers) now provide simultaneous service to large numbers of completely unknown and untrusted users. Reliance on information technology (IT) has skyrocketed - every individual and almost every aspect of modern life are affected by computers in some way even if not in a direct relationship. There is so much that must be considered when building a system that it is easy to overlook the most fundamental issues.

For instance, lessons had been learned in the early 1970's that no ad hoc fix or security patch could provide security if the operating system could not be trusted (Anderson, 1972 and Schell, Downey, and Popek, 1973). While it is uncommon today for a vendor to be foolish enough to claim a silver bullet solution, there are a significant number of these ad hoc security products available to throw at the problem. One can easily be lulled into a false sense of security by employing an assortment of products in a so-called "layered defense." However, the lessons of the past must not be forgotten. High assurance systems (i.e. systems with "verifiable protection") are the only known technology that can address the threat of subversion and subversion is a threat that deserves attention.

## C. HISTORICAL BACKGROUND

The focus of concern in the field of computer and information security has meandered through a seemingly infinite list of threats. In the beginning, computers ran a single program and the threat could be addressed with physical controls. Operating systems brought capabilities to reduce the time that these expensive machines stood idle by enabling them to share resources and run several programs "simultaneously." Protecting data and resources became a much more difficult problem in these shared-use systems as indicated in the Air Force Computer Technology Planning Study (Anderson, 1972):

> ... The nature of shared-use multilevel computer security systems presents to a malicious user a unique opportunity for attempting to subvert through programming the mechanism upon which security depends (i.e., the control of the computer vested in the operating system)…The threat that a single user of a system operating as a hostile agent can simply modify an operating system to by-pass or suspend security controls, and the fact that the operating system controlling the computer application(s) is developed outside of USAF control, contribute to the reluctance to certify (i.e., be convinced) that contemporary systems are secure or even can be secured.

The security kernel was offered as a proposed solution by providing domain separation and addressing the issues of object reuse, covert channels, and other avenues of attack on confidentiality and integrity that resulted from this development. Anderson (1972) describes the security kernel as follows:

> The objective for a security kernel design is to integrate in one part of an operating system all security related functions. This is for the purpose of being able to protect all parts of the security mechanism, and to apply certification techniques to the design.

This report also introduced the concept of the *Reference Monitor* (RM) the function of which is to validate that all references made by subjects (active entities in the system) to objects (passive entities) are authorized for the specific mode of access requested. In other words, whenever a process requests access of a particular mode (e.g. read/write) the RM must validate that the subject is authorized to reference the object in that mode. The report introduced the term *Reference Validation Mechanism* (RVM) as the implementation of the concept and stipulated the following three enduring requirements:

- The RVM must be tamper proof;
- The RVM must always be invoked; and
- The RVM must be small enough to be the subject of analysis and tests to assure that it is correct.

The RVM would be one component of the security kernel.

In 1979, Air University Review published an article (Schell, 1979), which presented evidence of weaknesses in the security controls of contemporaneous systems that were assumed to be secure by operators. The author compared the use of these systems under the false assumption that they were secure to Japan and Germany's confidence that their encryption systems could not be broken in World War II.

In the 1980's, the DoD conducted more studies to address the computer security problem. In 1983, "The Eagle's Own Plume" (Grant and Riche), illustrated the threat in the context of the increased use of electronics and computers in Naval Weapon's systems. The article provided hypothetical examples of how artifices inserted into weapons or weapon support systems by U.S. adversaries might cause failed communications and erroneous presentation of information and how this could negatively impact a conflict scenario. At approximately the same time, criteria for evaluating the security assurances of computer systems were being developed. These criteria were published in 1985 in the Department of Defense Trusted Computer Evaluation Criteria (TCSEC) or "Orange Book" (DOD 5200.28-STD, 1985). The RVM discussed above was incorporated into the criteria beginning at Division B (Mandatory Protection). Subsequent work resulted in an entire series of publications (commonly called the "Rainbow Series") which extended the criteria for application to networks and databases and amplified various aspects of functional and assurance requirements.

Increasingly, systems were linked together into networks and ultimately the Internet in the 1990's. With the Internet in particular, the number and sources of threats soared as hardware decreased in cost, the pace of software product development accelerated, and the number of users with the ability to connect within the network increased. To counter these new threats, cryptography, intrusion detection systems, virtual private networks, and firewalls took the forefront as the must-have tools for protecting information systems.

The end of this cycle appears to be nowhere in sight; new technologies and products constantly emerge to address the new threats that appear in the rear-view mirror only to be followed by yet another newly discovered vulnerability or exploitation technique. The recent creation of the Common Vulnerabilities and Exposures (CVE) List (http://www.cve.org) is testament to the magnitude of the problem – vulnerabilities are so prevalent, that a vulnerability taxonomy and management system is required just for the security community to communicate effectively about the problem.

Yet, with all this attention to computer and network security, with all the various security technology available today, the threat of subversion remains unchecked and

largely ignored. These new technologies simply cannot counter the threat of subversion. Loscocco, *et al* (1998) provide excellent examples of how these technologies fail to provide protection when it is falsely assumed that the underlying system is secure. It is not the case that we do not know how to address this threat. In fact, since the early 1970's, we have had available technology and procedures that, if applied properly, will eliminate the most serious threats of subversion. These will be discussed later.

## D. THE PROFESSIONAL ATTACKER

Subversion is the technique of choice for the professional attacker. In order to fully understand the subversion threat and appreciate why it deserves attention, the distinction between the professional and amateur attacker must be understood. The professional attacker is distinguished from the amateur by objectives, resources, access, and time. He is very concerned about avoiding detection and this will therefore be one of his primary objectives. Amateur attackers are often motivated by a desire for notoriety or simple curiosity as much as for gaining access. The problem for them becomes maintaining selective anonymity relative to the observers – allowing some observers to attribute the attack to them while denying the same ability to law enforcement or other authorities.

The professional will often be well funded and have adequate resources to research and test the attack in a closed environment to make its execution flawless. A flawless attack will attract less attention than one that must be mounted numerous times due to errors or bugs.

As described by Meyers (1980), the professional attacker is one who understands the system lifecycle. Using this knowledge, he may construct a subverted environment by controlling the efforts of a development team without the individuals realizing that they are involved in the subversion activity. In a large system with complex dependencies between modules (as is common in today's operating systems) the opportunities for this approach are clearly evident.

Coordinated attacks are also the mark of a professional. These types of attacks are launched from multiple systems simultaneously at a single target and can achieve a big pay-off in a short period of time. They also complicate efforts to discover the origin

of the attack, helping to conceal the identity of the attacker. The distributed denial of service (DDOS) attacks in which large numbers of vulnerable system are used by an attacker as "zombie" platforms to attack a common target, while not a professional attack, illustrate this concept.

Finally, the professional is willing to invest a significant amount of time in both the development of the artifice as well as its use – possibly waiting years before reaping the benefits of the act of subversion. This final characteristic demonstrates another facet of the professional attack. The subverter (who plants the artifice) may be – in fact, usually will be - a different individual than the attacker. In this scenario an attacker may have paid someone else to perform the subversion and will at some point in the future, activate the artifice and attack the system. The artifice may be designed in a general enough way that an arbitrary attack is possible.

An important note is that skill level is an independent consideration. The professional does not necessarily possess more skill than the amateur attacker. As will be shown by the example in this paper, the technical skill level required to plant an artifice may be comparable to anyone with a basic ability to understand and write code.

## E. CONTRAST WITH OTHER ATTACK METHODS

Meyers (1980) classifies attacks into three categories: Inadvertent Disclosure, Penetration, and Subversion. This taxonomy is presented with the assumption that the goal of each attack is bypassing system controls for the purpose of unauthorized disclosure of information. Distinctions are made between them based on the skill level of the attacker and the level of control he has on the system under attack. Inadvertent disclosures predominantly result from accidentally occurring states in the system or simply when a human error allows unauthorized entities the ability to observe information in an unintended way. Penetration is a more deliberate attempt to exploit a flaw that *already exists* in the system to bypass security controls. The penetrator must be content with working under the constraints imposed upon him by a system over which he has no control. He will exploit bugs or other types of flaws to bypass or disable security mechanisms on the system. In contrast, the subverter is skilled and knowledgeable and has sufficient access to the system at one or more points in its life cycle to exert influence on its design, implementation, distribution, installation, and/or production in a way that

can be later used to bypass the protection mechanisms. He will not rely on the presence of an accidental bug in the system (which may be corrected at any time), but will favor instead a carefully hidden mechanism that he can be relatively sure will persist through new product versions and upgrades.

The distinction between the so-called Trojan Horse and an artifice as used in system subversion is important to understand. The Trojan Horse is a piece of software that provides two distinct functions. One is observable and is a function that entices an individual to use the software. The other function is hidden and carries out the malicious intentions of its designer. Implicit in this description is that the Trojan Horse requires actions (but not the active cooperation) of a legitimate user on the system. It will therefore be constrained by the level of access that this user has on the system. With this technique, system security mechanisms are still in place and functioning properly – the attacker's code is executed with a surrogate's permissions. If the surrogate has a restricted domain of control, the Trojan Horse software will be limited in its utility to the attacker. Subversion on the other hand, does not require action by a legitimate user. It simply bypasses any security mechanisms that the subverter chooses to bypass.

Additionally, an artifice will typically include the capability to be remotely activated and deactivated. The normal state for the artifice is deactivated. In this way, the artifice is less likely to be observed by the users and operators of the system. The mechanism that activates the artifice waits for some unique trigger to exist in the system. Examples are a particular stack state, an unlikely sequence of system calls or signals, or codes hidden in unused portions of data structures passed into the kernel via system calls. The possibilities are endless. This trigger can be thought of as a key that can be made arbitrarily long from a cryptographic standpoint. Other examples from Grant and Richie (1983) include triggers such as geographic position or keywords observed in messages processed by communication systems. As a result, no amount of security test and evaluation (ST&E) can confirm the presence or absence of the artifice. This will be explored later in Chapter VI. "Limiting the Risk of Subversion in Information Systems."

Brinkley and Schell (1994) classify misuse techniques similarly into human error, user abuse of authority, direct probing, probing with malicious software, direct

penetration, and subversion of security mechanism. In this taxonomy, subversion stands apart as well for the same reasons described above.

## F. WHY DECISION MAKERS SHOULD SERIOUSLY CONSIDER THE THREAT OF SUBVERSION

That subversion is a threat is even truer today than it was in the past. The sheer size and complexity of today's systems alone make the insertion and hiding of an artifice much easier than it was at the time of Meyers' thesis. Additionally, design and development efforts are more divided among many individuals or departments which makes it possible to divide the artifice into separate modules further obscuring its true nature.

As the market for information technology gradually shifted from the government and military to the private sector, the government's ability to keep pace with industry diminished. For the government to continue developing systems in-house would hinder its ability to do business with private industry. As a result, the government shifted to the use of commercial products and in so doing resigned itself to accepting whatever direction products were taken by the larger market forces. It is now clearly evident that security was unimportant to the broader market and therefore had little impact on an IT company's bottom line. As competition among vendors increased, the pressure to release products by deadlines intensified. As a result, a subverter may stand a better chance of getting his artifice through quality control (if it is noticeable at all) in the rush to get the product to market. This is the environment in which the products we rely upon for our information technology needs are developed.

One does not need to look far to find reports of security critical bugs and system intrusions that could just as easily been the result of subversion as error. Recent press reports illustrate the opportunity and motive to conduct a subversive attack in the Microsoft Windows operating system. As this is a major vendor with a tremendous market share, any such subversive attack would have far reaching impact. During the month of October 2000, an individual gained access to systems at Microsoft and had access to the source code for a future release of the Windows operating system and

Office Suite[2]. The company of course dismissed the possibility that the code was tampered with, but it would be impossible to provide any real level of assurance that this is true. In January 2001, Verisign Corporation (http://www.verisign.com) erroneously issued Microsoft certificates to individuals who falsely claimed to be Microsoft employees[3]. As a result, these individuals had the opportunity to publish code that would appear to be certified by Microsoft Corporation. Hence, many users (even administrators) might associate a false level of trust with malicious software (operating system upgrades, etc.) written by these individuals. On Dec 17, 2001, shortly after the United States' war on terrorism began, Newsbytes reported[4] on claims by a captured member of the Al Qaeda regime in Afghanistan that members of a terrorist organization had infiltrated the Microsoft Corporation as programmers for the expressed purpose of subverting the Windows XP operating system. While there was a lack of corroborating reports, this indicates that subversion is considered as a valid attack technique.

Taken together, these cases demonstrate that both opportunity and motive exist for carrying out a subversive attack. By the example presented later in this paper, one may conclude that the means to mount such an attack would not be hard to attain. Therefore, in order to fulfill the responsibility to protect information systems, decision makers should be aware of this threat and consider its significance in all deployments of information technology. Certainly, decision makers could be considered negligent if subversion is ignored in a wide range of military systems or in the systems upon which the critical infrastructure of the United States Depends.

## G. OUTLINE

The remainder of this paper is organized as follows. Chapter II. "High Level Discussion of the Network File Server (NFS) Experiment" describes the experiment from a high level, covering the goals and objectives and how these affected the design of the artifice.

---

[2] See http://money.cnn.com/2000/10/27/technology/microsoft/

[3] See http://www.verisign.com/developer/notice/authenticode/

[4] See http://www.newsbytes.com/news/01/173039.html

Chapter III. "High Level Discussion of SSL Subversion" describes an attack, which was not implemented in this project, by which an attacker can observe the SSL encrypted network communications between two systems with little to no risk of being discovered. It is shown that no amount of assurance on the server side alone can prevent such an occurrence if the client is subverted.

Chapter IV. "Detailed Description of the NFS Example" describes the implementation of the NFS experiment in detail.

Chapter V. "Evaluating System Security in the Face of Artifices" debunks the prospect of discovering the presence of artifices or proving their absence in a system after it has been developed either through code inspection or by security test and evaluation.

Chapter VI. "Limiting the Risk of Subversion in Information Systems" presents the only known solution for eliminating the threat of subversion.

Chapter VII. "Conclusions and Future Work" ties all of these points together and proposes future direction to address the threat of subversion.

# II. HIGH LEVEL DISCUSSION OF THE NETWORK FILE SERVER (NFS) EXPERIMENT

## A. HIGH LEVEL CONSIDERATIONS

### 1. General

The primary purpose of this thesis is to highlight the risk of the threat posed to an information system by a professional attacker mounting a subversion attack. A demonstration of a working example was decided to be the best option for making this point. To achieve maximum impact, not only did an artifice need to be constructed, but a full demonstration of an attack had to be developed as well. As a result, the experiment diverges from a true reflection of the professional attack in several respects. First, there were significant restrictions on the time available to construct the artifice. This would rarely be the case in a well crafted artifice such as might be constructed by a professional attacker (e.g. a nation-state, organized crime group, or company engaged in corporate espionage). Second, the author was free to openly plant the artifice and did not need to be concerned that his actions might be discovered. Obfuscation of the artifice was not given serious attention. In a professional attack, obfuscation would be of the utmost importance and would add to the time required. Finally, there was essentially no long-term motivation in the development of this attack. The professional would be motivated by the prospect of a very significant payoff potentially far in the future while the author was motivated by a short-term payoff goal. In spite of these constraints, this experiment demonstrates the inability of contemporary approaches (such as application level security, encryption, security test and evaluation (ST&E), and perimeter defenses) to provide for security against subversion.

### 2. Selecting the Method of Subversion and the Target System

Subversion may occur at any point in the lifecycle of a system (Meyers, 1980). However, since access to the lifecycle of a major operating system is quite limited for a student, the installation and maintenance phases present greater opportunity for insertion of an artifice. Also, in the case of open source systems, a distribution phase attack could be mounted. Several Linux vendors offer no-cost versions of their operating system for download from their websites. MD5 hashes of the CDROM images are posted to provide

customers with the ability to check the integrity of the downloaded images. Often, third parties provide mirror sites to reduce the load on a single server, which opens the possibility for a malicious mirror site offering subverted versions of the software.

For the reasons stated above, the Linux operating system was chosen as the platform on which to build the demonstration. The availability of source code and the abundance of documentation make the Linux kernel fit well into the constraints of the experiment.

Choosing an open source operating system for this experiment runs the risk that some readers will be lead to believe that closed source products are immune to the attack as demonstrated. While it is arguably more difficult, the task is not particularly daunting for the professional attacker described above. As this example is carried out late in the lifecycle of the product, there would be significantly more work involved in subverting a closed source product. Planting an artifice during the distribution or installation phase would involve reverse engineering the application and creating a binary patch to insert the artifice at the appropriate location in the product binaries. However, during the earlier phases of the lifecycle, the level of difficulty from a programming standpoint is the same for both closed and open source products. Then there is the question of access. Here again, the professional is one who will either have the access or have the resources to obtain it regardless of the open or proprietary nature of the system.

### 3. Selecting a Suitable Attack Demonstration

In order to achieve the desired impact, an attack had to be selected such that its significance was readily apparent to the average observer. Moreover, the activation mechanism needed to be commonly available and understood. To this end, the author chose a Network File Server (NFS) as the application on which to demonstrate the attack. The file server is general enough and common enough that the typical user of information technology will immediately understand its function as well as the notion that one user's data should be protected in some way from access by other users who lack permission to do so. A demonstration in which an attacker is able to obtain access to information which he should not have should be comprehendible to readers with diverse levels of technical sophistication.

Likewise, the activation must be fairly straightforward. After investigating several options, the author settled on using the network interface for activation. Again, network communications are easily explainable to most audiences. They are also ubiquitous enough that the activation interface will likely be available on any target system as well. The Internet is one of the most widely used technologies available today. For an overview of the aspects of Internet networking that apply to the following discussion, see the APPENDIX.

The specific form of activation in this experiment involves sending a malformed packet, which contains some additional information to the target. The target's networking implementation (having been subverted) will drop the malformed packed as it normally would, but in addition would recognize the packet as being a trigger to activate the artifice that will then grant the attacker access. Hence, the artifice on any system connected to a network can be activated remotely from any other system connected on the same network (e.g. the Internet, Wide Area Network or Local Area Network). Additionally, the artifice can be activated locally by sending the activation packed to the host's loopback address (an address that loops back to the sender of the packet).

For the reader who does not have the opportunity to observe a demonstration of the NFS example, the attack (shown in Figure 1.) proceeds as follows. A client is shown to have the file system of a remote network file server mounted locally. The demonstrator shows how he is denied permission to access a certain portion of the file system. An activation packet is sent to the subverted server. Then the demonstrator shows how he subsequently has access to read, write, and modify file objects on the server. Finally, a packet is sent to deactivate the artifice and the normal functioning of the system is confirmed by showing a denied attempt to access another user's file or directory on the server.

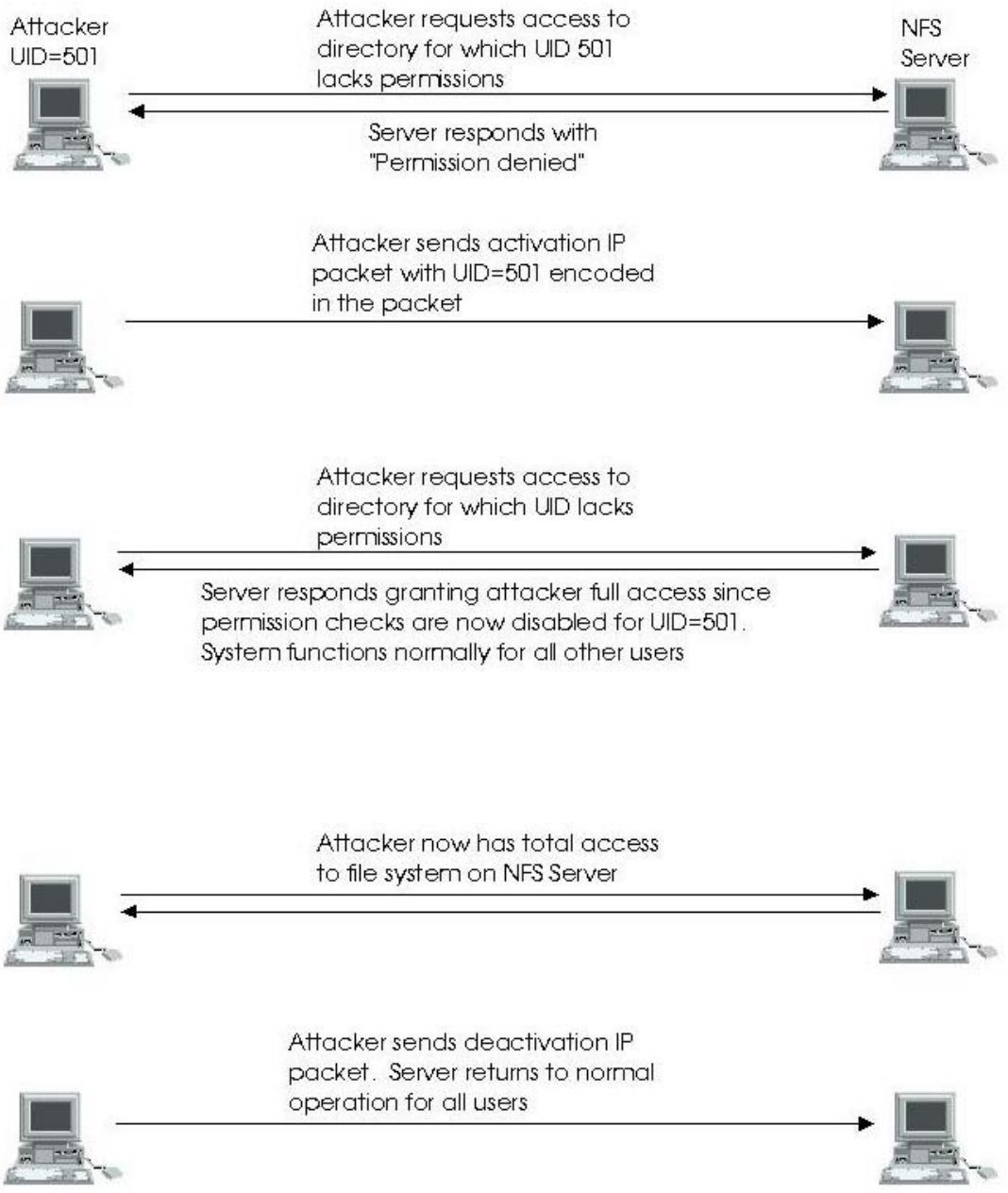



Figure 1. The NFS Attack Scenario

## B. ARTIFICE DESIGN AND INTEGRATION INTO THE OPERATING SYSTEM

The purpose of the artifice used in this example is to provide a means to bypass file permission checks for a specified user at will and then to re-enable the normal system operation. The Linux kernel uses a virtual file system that abstracts all of the various types of files systems and devices into a logical, consistent interface.

The example presented here shows how a few lines of code can result in a significant vulnerability. It exhibits all of the characteristics of subversion except that no real attempt has been made to hide or obfuscate the artifice. First, the artifice is small. In all, eleven statements in the C programming language are needed for this example. This small size in relation to the millions of lines of code in the Linux Kernel makes it highly unlikely that any but those involved in the development of the kernel would notice. The artifice itself is composed of two parts located in two unrelated areas of the kernel.

A second characteristic is that it can be activated and deactivated. As a result, the functionality exists only when the attacker wills it. This will further complicate any attempt to discover the existence of the artifice. Unlike some Trojan Horse attacks, there will be no suspicious processes running on the system. Even when activated, the functionality is embedded in the kernel and not in a user-space process so its observability is limited. Under these conditions, no amount of testing is likely to show the presence or absence of a well-hidden artifice.

Finally, it does not depend on the activities of any user on the system. The attacker can activate and deactivate the artifice at will as long as the system will process IP packets. He is therefore not subject to any permission constraints of a particular user on the system. Moreover, the fact that all users and administrators of the system may be trusted (for example in an environment where all users are cleared to the same level and the system is operated in system-high mode), has no effect on the attacker's ability to exploit the system. Administrators make the system vulnerable simply by connecting it to a network.[5]

[5] A closed system can be vulnerable as well by using triggers based on other conditions in the system such as geographic position, system load, etc. For examples, see Grant and Riche (1983).

### 1. Artifice Function

The artifice in this example subverts the Linux file system permission checks. When in the activated state, the artifice grants the attacker access to any file on the system by causing the file permission check in the kernel to be bypassed. This behavior is limited to a specific user ID that the attacker specifies at activation time. The system functions normally for all other users. In fact, the attacker can even use a userID that is unused (i.e. one for which no account exists) on the target system. Activation and deactivation is accomplished by sending a single User Datagram Packet (UDP) to the target system. The portion of the kernel that receives network communications has also been subverted to recognize a packet that has a distinguishing characteristic (a trigger known only to those involved in the subversion and the attack) that activates and deactivates the artifice in the file system. The trigger can be made to be arbitrarily unique to avoid not only accidental activation and deactivation, but also make it difficult to guess the activation code (this will be discussed in greater detail later).

The operation of the file permission check mechanism in the subverted system is shown in Figure 2. The comparisons that are made in the decision branches reference a global variable in the kernel. Since the variable is global, any portion of the kernel can have access to this value. Global variables are used quite often in the Linux kernel.

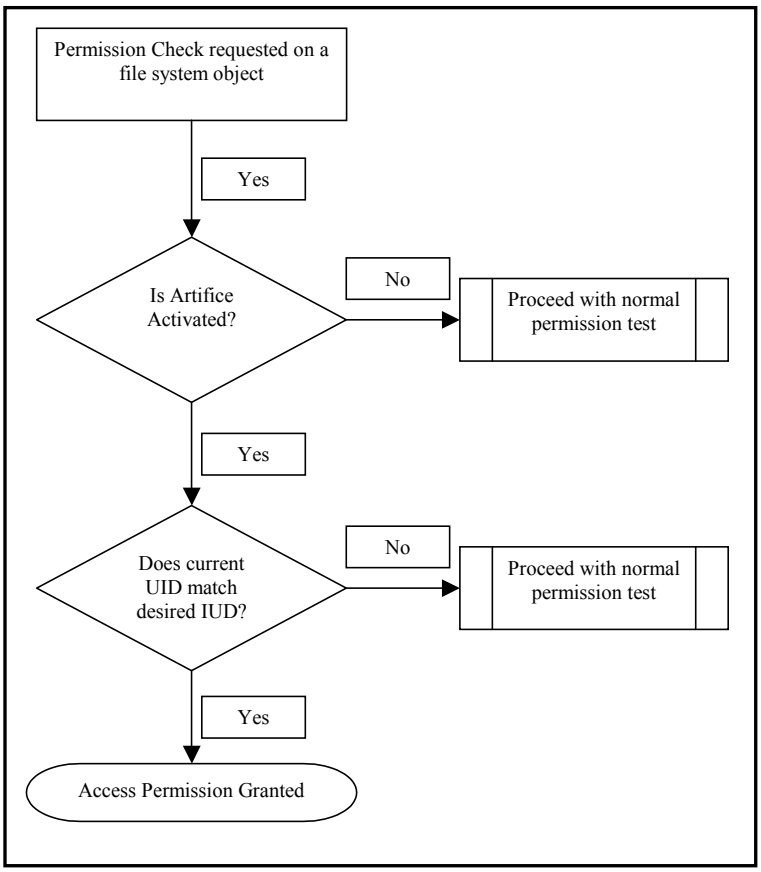



Figure 2. File Permission Checks in a Subverted System

The artifices' global variables are set and cleared in the activation and deactivation mechanism. This occurs in the portion of the kernel that implement the network protocol stack. In the Transmission Control Protocol/Internet Protocol (TCP/IP), headers contain checksum values to ensure integrity of the received data. The example uses a User Datagram Packet (UDP) to activate and deactivate the artifice. In the UDP handling code of the kernel, every UDP packet is tested for an invalid checksum. If one is detected, control passes to a portion of the kernel that logs an error and drops the packet (removes it from the list of packets in the queue). In the subverted kernel, the checksum error code looks for some predefined unique values. If this condition is met, it sets the artifice state to "On" and the user who is to receive privilege to the value in the source port field of the UDP packet. This control flow is shown graphically in Figure 3.

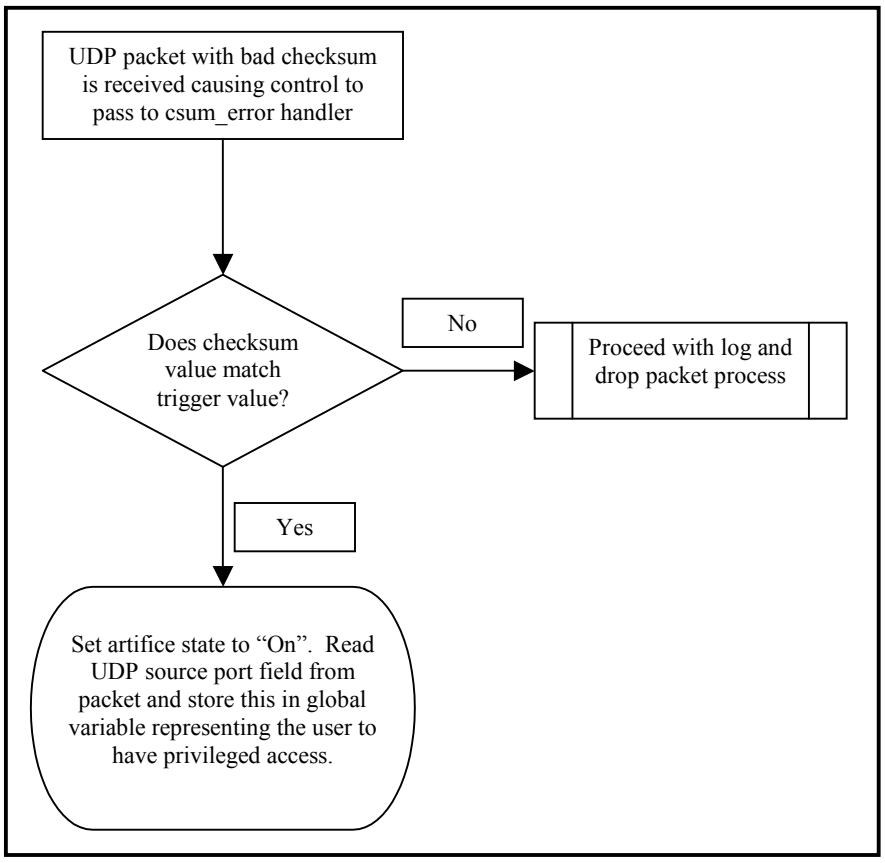



Figure 3. Artifice Activation

Deactivation occurs the same way except that the global variables are set so that the artifice is deactivated and the system returns to its original operation.

## C. THE SUBVERTED SYSTEM IN OPERATION

A system subverted in this way is vulnerable simply by connecting it to a network that can be reached by the attacker. In fact, the *only* action required on the part of an administrator of the NFS system is to connect it. Furthermore, the only action he can take to deny access to the attacker is to disconnect it (making it useless as an NFS server). While it is true that there are additional measures that can be taken to enhance the security of NFS over the configuration used here, they are largely ineffective, as the subverter would simply bypass these security controls as well. For example, NFS servers register clients that are permitted to use them. If the attacker works from one of these machines, the attack is trivial. If the attacker does not have access to one of these

machines, he merely needs to be in a position to observe traffic on the network and spoof his identity as that of a legitimate NFS client.

## D. CHAPTER SUMMARY

We have described an example of subversion, which is simple but effective for illustrating the risk of subversion. While it is possible to secure an NFS server in ways that will render this artifice ineffective, to dwell on such details would distract from the point. The main point here is that any protection mechanism can be rendered ineffective by modifying it so that the protection mechanism is bypassed. With the risk that the artifice can be planted in the kernel, no application layer security solution can be counted on to provide the protection for which it is designed.

THIS PAGE INTENTIONALLY LEFT BLANK

# III. HIGH LEVEL DISCUSSION OF SSL SUBVERSION

As a second example, a high level discussion is presented here of a subverted Secure Sockets Layer (SSL). SSL is commonly used to provide secure communications between web servers and web clients (browsers) so that internet banking transactions and credit card purchases for web commerce applications are protected from unauthorized observation or modification. The client and server negotiate session keys that are used to encrypt traffic between them using some symmetric encryption algorithm. Linux systems commonly use OpenSSL (http://www.openssl.org) to implement SSL. The Apache Web Server[6] (http://www.apache.org) has a mod_ssl package available that acts as an interface between the web server and OpenSSL. Therefore the Apache web server relies on OpenSSL to perform the encryption and decryption of the web traffic.

## A. OVERVIEW OF A POSSIBLE SSL SUBVERSION

SSL works in the following manner. The client and server use symmetric session keys to encrypt and decrypt traffic sent between them. The session keys are generated by the sender and then encrypted with either a secret shared by both systems or by using some public/private key asymmetric encryption. When the encrypted session key is received, it is decrypted by the receiving system and then used to decrypt the data received from the sender. There are a number of options available to the attacker for subverting this mechanism. The most obvious is to simply duplicate all communications in an unencrypted form and send them to a system of the attacker's choosing. This option might cause far too much observable system activity to be considered viable by the professional attacker. Another option could be to weaken the key generation mechanism by limiting the amount of entropy used to generate random numbers. The attacker could then capture the network transmissions and the task of "breaking the code" would be sufficiently constrained so as to make it computationally practical.

A much better option than either of these two would be to simply have one of the systems participating in the communication send the session keys out in an unencrypted form. The attacker could then position himself between the two systems, gather the

[6] According to the Netcraft Survey (http://www.netcraft.com/survey), over 65% of the active web servers in February 2002 were running the Apache web server.

encrypted traffic and at the same time, watch for the session keys. Having both, he could then decrypt and read the data. Due to the fact that the session keys are unknown on either system outside of the SSL implementation, plus the fact that they are a 'random' sequence of bits, it is unlikely that any observation of them would lead to the conclusion that they are keys that have recently been used to encrypt SSL traffic. As an added precaution however, the session keys could be transmitted in a way that draws little attention to the fact that they are being sent as described below.

Web communications use Transmission Control Protocol (TCP), which has provisions to guarantee the connection's integrity. One of the ways it does this is by providing sequence numbers in each packet. The sequence numbers give the receiver the ability to ensure that it received all of the packets (by checking for a missing number in the sequence) as well as to put the packets back together in the correct order. In much the same way that the trigger is set in the NFS example, the session keys could be embedded in a packet that has an invalid checksum. The receiver would therefore ignore and drop the packet. Since the checksum is bad, the receiver would normally request that the sender retransmit it. However, the artifice would simply cause the packet to be dropped silently in this case. The logging of the bad packet could be bypassed as well. To the attacker eavesdropping on the communication, these packets would be tested for a characteristic chosen by the attacker to indicate the presence of the session key. In order to know which data packets are associated with which key, the artifice could reuse one of the legitimate TCP sequence numbers that was used to transmit the SSL encrypted data.

This approach has some distinct advantages. First, the attacker can be totally passive and maintain anonymity. No packet needs to be sent to him directly by either system. Another advantage is that a subverted system on either end would provide all required information for decrypting the traffic. This is significant in that it invalidates any assumption that using high assurance systems or additional security products at the server end only will protect the data stored at the server. Since both ends must use the same session keys, any client that is permitted to access the server can potentially undermine the security in place on the server end.

For instance, one might provide additional protection of the server's private keys by storing them on a smart card. With this design, the session keys are decrypted on the smart card, which adds some level of protection over decrypting the keys on the host operating system since the private key is not exposed. However, the computationally intensive decryption of the *traffic* is most often performed off of the card due to smart card processor and I/O limitations. As a result, the session key (once decrypted on the smart card) will be sent to the host (i.e. server) system's SSL implementation that will then use it to decrypt the traffic. If the server's SSL implementation has been subverted to send the session key to the attacker, the presence of the smart card is irrelevant to the level of confidentiality realized. Additionally, since the same session key is used on both the server and client sides, the security is only as strong as the weaker of the two sides. The client may just as effectively transmit the keys as the server.[7]

## B. CHAPTER SUMMARY

We have provided an additional high-level discussion of a potential subversion of SSL, which is relied upon for protecting confidential communications on the internet. In this example, subversion of either the web server or the web client can result in a compromise of this communication. As proposed here, placing the artifice in the SSL implementation will affect not only web traffic, but also any application that relies on SSL. We have also presented a case against investing significant amounts of resources in placing strong security measures at the server side only. Before making such an investment, the prospect of subversion and its potential consequences should be thoroughly considered.

[7] This is not to imply that every component in the network must be high assurance. Architectures such as those presented in Weissman (1992) and Fellows, *et. al.* (1987) provide sufficient security at both the client and the server without requiring the client to give up "typical commercial functionality."

THIS PAGE INTENTIONALLY LEFT BLANK

# IV. DETAILED DESCRIPTION OF THE NFS EXAMPLE

The details of the example are presented here for those who are interested. The discussion begins with a description of the normal functioning of file permission checks. It must be remembered however, that the details of what happens in these tests were rendered totally irrelevant in the experiment. None of these details matter since the artifice simply bypasses the permission checks.

## A. LINUX IMPLEMENTATION OVERVIEW

Linux implements a Virtual File System (VFS), which is a software layer in the kernel that abstracts the system calls for various types of file systems into a single common interface. The VFS provides access to disk-based filesystems (Unix, ext2, MS-DOS, VFAT, etc) and special filesystems (`/proc` and `/dev`). In addition, VFS sits above the implementations for several network files systems such as the Network File Server (NFS), Coda, SMB (Microsoft's Windows and IBM's OS/2 LAN Manager), and Novell's NetWare Core Protocol (NCP). A diagram of this relationship is shown below in Figure 4.

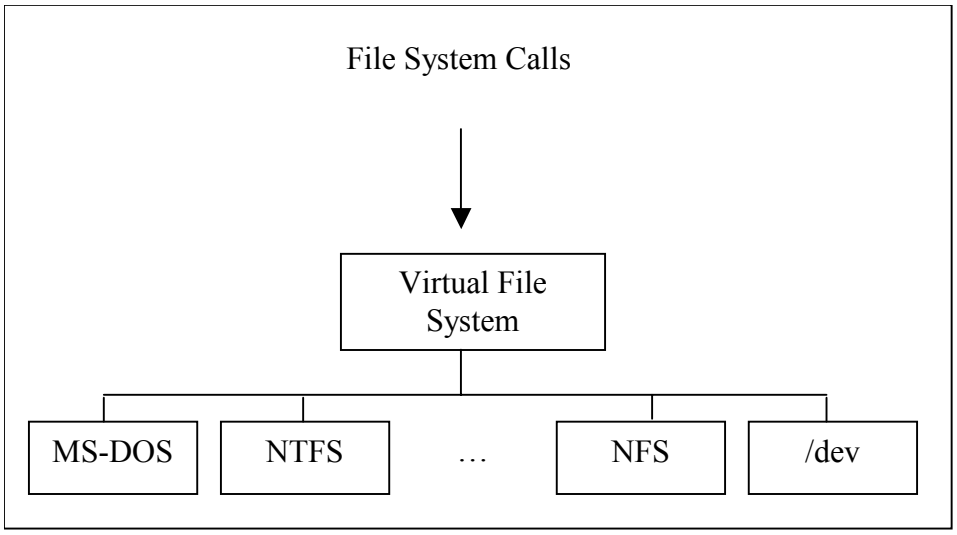



Figure 4. The Linux Virtual File System

File object access permissions in Linux are implemented using the standard Unix file permission bits for read, write, and execute as granted to owner, group and others. All requests for access to a file flow through the function `permission()` located in the kernel source file `fs/namei.c`. The file's `inode` structure is first checked for a

pointer to a file system-specific `permission()` function. If there is none, `permission()` calls the `vfs_permission()` function. Permission bits are checked against the user specified in the current process' `task_struct` (defined in `include/linux/sched.h`) which is stored in the process descriptor in the kernel-space stack as shown in Figure 5. below. The file system user ID (`fsuid`) is used by this function in the tests for permissions. Having a separate file system userID (`fsuid`) enables the kernel to handle requests from a process that is acting on behalf of another process without modifying the normal `uid` or effective user ID (`euid`) which may be root or 0. The `permission()` function returns a zero if the permission test succeeds and the `-EACCESS` constant if it fails.

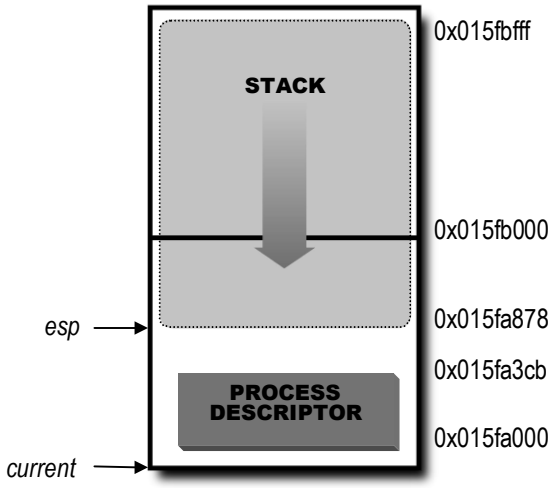



Figure 5. The Process Kernel Stack (From Bovet & Cesati, 2001)

### 1. Artifice Implementation

The VFS system is an attractive location for the artifice since it can affect numerous file systems types. Simply returning a zero from the `permission()` function will result in approval of the requested access. This alone, however, is far too general, as all users on the system will be granted access to any file. To avoid such obvious behavior, a mechanism must be built through which a particular user can be specified leaving all other users on the system subject to the normal permission checking. This functionality is implemented in the activation mechanism. Any activation mechanism should be flexible enough to be used in a variety of situations and ubiquitous

enough that the attacker can be relatively sure it will be available on his target. In meeting these design objectives, the networking protocol is a suitable location for the activation portion of the artifice.

The portion of the artifice that grants access is located in the `fs/namei.c` source file of the Linux 2.4.3 kernel. Two global variables, `activated` and `userid`, are used to store the state of the artifice (deactivated=0, activated=1) and the `uid` for which to grant access respectively. Immediately upon entering the `permission()` function, a test is conducted by an '`if`' statement that evaluates (1) whether or not the artifice has been activated and (2) whether the process' `fsuid` is equal to the global `userid` variable. If so, the permission check returns success (access granted) regardless of the file permissions on the requested file. Otherwise, the permission tests proceeds as normal.

The activation and deactivation artifice is planted in the Linux 2.4.3 kernel source file `net/ipv4/udp.c`. Any error detected in the checksum of a UPD header will result in execution branching to code that logs the event and silently discards the packet. The artifice is planted in the portion of code that provides this function. The artifice is embodied in two '`if`' statements. The first statement activates the artifice. It first checks to see that (1) the artifice has not already been activated and (2) whether or not the bad checksum matches the activation codes of decimal 213 or 2113 (chosen arbitrarily). If so, it activates the artifice by setting the global variable `activated` to 1 and the `userid` variable to the user ID that has been specified in the UDP source port.

As designed in this example, activation proceeds in the following manner: a UDP packet is constructed with the following characteristics:

| Field | Value |
|---|---|
| Checksum | Invalid and equal to one of two predefined values (213 or 2113) |
| Source Port | The userID corresponding to the `uid` for which the attacker wishes to grant access. |
| Destination port | Irrelevant |

Table 1. Activation Packet Characteristics

Two predefined values (decimal 213 and 2113) for the checksum are used in the event that our 'secret' checksum (the activation code) turns out to be valid for the packet we construct. In the normal kernel, a packet with a bad checksum would simply be discarded. Since UDP is a stateless protocol, the packet will just be forgotten. Since userIDs are commonly small numbers (most Linux implementations begin UIDs for normal users close to 500), these packets will appear to have originated from a privileged (<1024) port on the host and will more likely be permitted through a firewall than those with higher port numbers. TCP could be used as well but would require a slightly more complex implementation. Once a UDP packet with these characteristics has been received, the artifice will be in the activated state.

Deactivation proceeds in a similar fashion. A packet with the following characteristics is sent to the target host:

| Field | Value |
|---|---|
| Checksum | Invalid and equal to one of two predefined values (312 or 3112) |
| Source Port | Equal to the current value stored in the `userid` variable |
| Destination port | Irrelevant |

Table 2. Deactivation Packet Characteristics

When a packet with these characteristics is processed, the artifice will set the `activated` variable to zero which deactivates the artifice.

There are a few things worth pointing out about this artifice. First, it gives the subverter even more power with respect to the file system than the root user has. Even

files for which the root user does not have current access[8] will be available to the attacker. Second, not only can this attack can be mounted remotely via any network path available, but it can also be launched locally by sending the activation and deactivation packets to the localhost (127.0.0.1) address.

### 2. The Network File Server as a Target

The Network File Server (NFS) allows a remote file system to be mounted locally and appear as though it were located on a local disk drive. NFS runs on top of the RPC protocol and uses AUTH_SYS as its default security mechanism. AUTH_SYS is a weak security protocol but is common in NFS implementations. In this protocol, the standard set of user credentials (the UID and associated Group ID's) are used for determining access. Every NFS request will contain these credentials. When a request comes into the server, the NFS Daemon maps the `uid` of the requestor into the `fsuid` of the NFS daemon process. After some initial checks, the request will be passed through the VFS and into the `permission()` function. As specified earlier, the access request will be evaluated based on the `fsuid`.

## B. CHAPTER SUMMARY

We have provided the details of the NFS subversion example. While the implementation is small (about 11 C language statements), the fact that it is in the source code listing makes it vulnerable to detection. Again, obfuscation was not one of the goals of this project. Given more time, the artifice could be hidden much better. The critical observation is that the technical skills required to subvert a system are relatively commonplace.

[8] Of course the root user could simply give himself access to the file using the UNIX `chmod` command.

THIS PAGE INTENTIONALLY LEFT BLANK

# V. EVALUATING SYSTEM SECURITY IN THE FACE OF ARTIFICES

As stated earlier, no real attempt was made to hide the artifice in the Network File Server example. In contrast, the professional attacker will most definitely spend significant effort on this aspect of the attack. This section will begin with a discussion of the various techniques that might be employed to hide an artifice followed by a discussion of the ways in which an artifice might be detected. Apart from catching the subverter *in the act* of subversion, discovering the presence of a particular artifice can be made extremely difficult.

## A. TECHNIQUES FOR FINDING AN ARTIFICE

The manner in which one may find an artifice depends upon the way in which it was hidden and the point in the product lifecycle when the subversion occurred.

### 1. Design and Implementation Phase Subversion

Meyers (1980) describes several examples of artifices that could be implemented in a system's design phase. Design phase subversion has the least chance of being "patched" away. Of course the artifice will be visible at some level in the design documentation but there is ample opportunity for a subverter to propose to a design team a seemingly sound alternative that offers subtle vulnerabilities. One subtle example can be found in asymmetric encryption protocols which is discussed below as an analogy.

Jim Alves-Foss (1998) presented a very interesting example in the context of public-key based authentication protocols. In this attack, an otherwise secure protocol is subverted in an indirect way by the existence of another independent protocol (possibly devised by an attacker) that is permitted to use the same key pair as the original protocol. By interleaving messages between the two protocols, the attacker can masquerade as a legitimate user to other entities. He demonstrated how a Needham Schroeder Public Key Protocol (which had been hardened against a previously known attack) could be spoofed if it used the same keys that are used in another protocol designed for a different purpose. Hence a subverter developing an application that uses public key certificates could devise his own protocol (possibly secure enough for the new application) that provides the

necessary pieces to masquerade as other entities in previously established protocols and applications.

What makes this example of subversion interesting is its circuitous nature. The target of the attack is not the authentication protocol but the public key infrastructure. The authentication protocol is not altered in any way by the subverter. To accomplish his goal, the subverter merely needs to influence policy so that his new application and protocol is permitted to use the same keys as the existing ones.

Such subtle interactions often occur accidentally and are discovered later with widespread use. However, as stated earlier, the well-designed artifice will have some form of unique trigger mechanism to hide its presence in normal operation modes. In this way accidental discovery from general use is nearly impossible.

The artifice in the NFS example could potentially be inserted during the implementation phase of the system lifecycle. In this mode of operation, the subverter will attempt to insert the artifice during the implementation or coding of the system. He can accomplish this either by infiltrating the development team as an agent or by exploiting weaknesses at the development facility (either in its physical security or in its configuration management system).

In a given system, the subverter often has a vast number of options for hiding an artifice. The use of low level languages such as assembly (common in the Linux kernel to enhance speed) can be used to code the artifice in a way that make understanding its nature difficult. For example, in the Linux operating system, every process acts on behalf of a user (possibly the system) who is identified by a userID stored in the process descriptor that the kernel maintains for each process. When a process requests access to a file, the kernel checks the userID of the requestor by inspecting this data structure. Since this data structure is used frequently and for many purposes, it must be fast. Linux achieves speed in functions like this by using low-level assembly language to program them. To keep the difficulty of writing the code at a minimum, a macro called `current` has been implemented that will return the starting address of the process descriptor belonging to the current process. Individual members of the process descriptor (e.g. `fsuid`) are referenced relative to this base address. Prior to compilation, every instance

of `current` in the source code will be replaced with the assembly language statements. So, one way to hide a portion of the NFS artifice, would be to replace the macro `current` with a modified version of the assembly code that would return the address of the `fsuid` data member rather that the beginning of the data structure. It therefore would be less likely to be associated with the original macro's function.

Any artifice placed in the system's source code is visible in the source listing. A developer with understanding of the system of sufficient breadth and depth might discover the artifice through inspection. This depends on how well the subverter has hidden it. The subverter might also remove the artifice from the source listing after it has been compiled. Another technique was presented in Ken Thompson's 1984 Turing Award acceptance speech (Thompson, 1984). In this speech, he described a subverted compiler that would recognize when it was compiling the UNIX source code and insert a trap door. In this way, the trap door could not be discovered from inspection of the source code. This speech sent shock waves through the audience at the time, but this technique was described more than a decade earlier in Karger and Schell (1974).

**2. Distribution, Maintenance, and Support**

With these attacks, the subverter can choose to affect all delivered systems or focus the artifice with respect to a particular target. In these attacks, the system has been through testing for quality assurance and a binary has been produced that is ready to ship to customers. The subverter gains access at some point in the distribution system. He then modifies the system by applying a patch to the binaries or by replacing the binaries entirely.

Similar methods are used in maintenance and support phases. Most vendors have patches and updates available for download via the Internet. Others ship them through the mail or send them with maintenance personnel. This artifice is perhaps the easiest to find if a known good copy of the system can be obtained for a byte-by-byte comparison. The difficult task is establishing that you have a good copy. Some measure of assurance is provided by digitally signing the binaries. The vendor runs a cryptographic one-way hash function over the binaries and then signs it using a private key. To confirm that the software has not been modified, the customer uses the vendor's public key to verify the

signature. In some cases, a GNU Privacy Guard (GPG) key may be posted on the website with the software. GPG is an open source equivalent to Pretty Good Privacy (PGP). The trust model of these schemes is such that each user determines whom to trust individually as contrasted with the model used in the commercial Public Key Infrastructure (PKI) and Department of Defense models in which a trusted root authority (e.g. the DoD, Verisign, etc.) binds identities to public keys and signs a certificate that provides this binding. Providing the public keys through the same distribution channel as the software leaves customers with little reason to trust that the software, public keys, and hash values have not all been modified. The PKI trust model has been compromised in the past as well as described in the Verisign certificates case in Section I.F. "Why Decision Makers Should Seriously Consider the Threat of Subversion."

## B. PROVING THE PRESENCE OR ABSENCE OF AN ARTIFICE

The question to ask given an example of an artifice such as the one presented here is not "How can this artifice be found" but "How can one be sure that there are no artifices in the system?" The answer to the first question is of little value. The artifice presented here can be found since it appears in the source listing and two new global variables are present in the system. It is also relatively overt in that it operates in a rather direct manner inside a security critical function. However, without controls as described in Chapter VI. "Limiting the Risk of Subversion in Information Systems", gaining the assurance that a system is free of artifices is an impossible task. Two possible ways of "finding" an artifice once it has been planted have been suggested. The use of these techniques to provide protection from a subversion attack are refuted below.

### 1. Source Code Inspection Will Fail to Reveal an Artifice

Having the system's source code available will not give an inspector the ability to ensure that it is free of artifices. As stated earlier, the professional will have hidden the artifice in a way that makes it exceedingly difficult if not impossible to identify by inspection of the source code. In fact, it is possible for the subverter to plant an artifice in a way that it never appears in the source listing of the system, for example by subverting the compiler used in developing the system, or planting the artifice in the object or binary code directly.

Even without going to the trouble of subverting the compiler, the subverter can hide an artifice in a way that it is likely to never be found. Today's systems are much larger and more complex than ever before. The more complex a system is, the harder it is to understand its overall function. The attacker can use this fact to make his artifice hard to understand. Programmers will often leave code alone if they do not understand what it does and it is not known to cause any problems in the system.

**2. Security Test and Evaluation (ST&E) Will Fail to Reveal an Artifice**

Security tests and penetration testing are worthless tools for assuring that a system is free of artifices. Any artifice that incorporates a well-designed trigger (one which is unique) will never be found because the testing will occur when the artifice is disabled. The function of the artifice will never be noticed through this type of testing. Code testing involves testing for the documented features of the system to make sure they function properly. It does not consider what other undocumented functions may exist in the system. To do so by testing would require that all possible input be sent to all possible interfaces. This exhaustive approach is infeasible if not impossible for most systems.

An informal argument supporting this assertion can be made by comparing the task of finding an artifice to finding a software bug in a system. It is an indisputable fact that certain bugs may appear in software that are particularly difficult to track down. These bugs may appear and disappear as a result of a rare combination of conditions in the system. Finding them is exceedingly difficult. Now imagine that a similar function is built intentionally. The feature manifests itself only when a rare combination of system conditions exists and furthermore, it can be triggered to occur at the will of the attacker. If finding the source of such a condition that has manifested itself in a noticeable way is a hard problem, how much harder would it be to find one that will manifested itself only when triggered? It would be impossible to devise a procedure to test for clandestine code that operates this way.

Given a black box (i.e. one for which the internal operations are unknown), it is impossible to determine whether or not it contains an artifice. As stated in Section A.1. of this chapter "Design and Implementation Phase Subversion," the attacker can ensure that nothing more than a black box exists with respect to the artifice, for example by

removing the artifice from the source listing after compilation or by modifying the compiler to perpetually insert it. In this way, the source code for the artifice will cease to exist yet the functionality will remain in the system.

Edsger Dijkstra provided a scenario of testing a 72-bit adder that illustrates the futility of using testing to determine whether or not an artifice exists in the system. In this scenario, the tester is given a black box that adds two 72-bit values with carry[9]. His task is to determine if the adder performs correctly for any given input. He must be able to state one of two results: (1) the adder performs correctly for every possible input or (2) the adder does not perform correctly for every possible input. If the result is (2), the tester must provide the input conditions (i.e. the two numbers) for which the adder fails. Obviously, the only way to complete this task under these conditions is to test all possible inputs and validate the output. The number of inputs that would need to be tested is $2^{72}$ X $2^{72}$ or $2^{144}$ possibilities. Consider the strength of a 128 bit cryptographic key and the amount of work required to mount a brute force attack against it. The work involved in cracking a 128-bit key pales in comparison to the work required to test all inputs to the adder. Conducting this test is computationally infeasible.

Compare this scenario to the task of determining whether or not an artifice exists in a system. As stated earlier, the subverter can use an arbitrarily long value as the activation key. Suppose the tester or certifier knows how to tell if the artifice is active as well as how to try guesses of the trigger value (i.e. he knows how to send an arbitrary value into the trigger mechanism). This is essentially the same scenario as Dijkstra's 72-bit adder with carry problem. Since the activation key can be made arbitrarily long, it is computationally infeasible (i.e. impossible) to test whether or not the artifice is present in the system. Now, suppose the task is to test for the presence of an *unknown* artifice. In this case, the tester does not know how to send in a guess at the key. Even if he stumbled across it and somehow accidentally activates the artifice (though he would likely not realize he has done so), what would he look for in the system? What does he test for after he makes an attempt at activation? The problem space is now the product of key space,

[9] A binary add with carry results in the following operations: two '0's added results in a '0', a '1' and a '0' added results in a '1'. Two '1's added will result in a '0' with a '1' carried over as input to the add operation performed on the next higher order bit.

potential activation mechanisms, and artifice function. The added difficulty of this task is obvious. These testing arguments can be extended to cover so-called "active defenses" built upon the use of intelligent agents or network monitoring to ensure system self-protection, or to the use of cryptographic techniques for system protection. Thease are worthless in the face of subversion.

## C. CHAPTER SUMMARY

The obvious conclusion from these examples is that no amount of code review or ST&E can provide even the most basic level of assurance that a system is free of artifices. How then can one ever have enough confidence in an information system to use it for critical applications? This question will be addressed in Chapter VI "Limiting the Risk of Subversion in Information Systems."

THIS PAGE INTENTIONALLY LEFT BLANK

# VI. LIMITING THE RISK OF SUBVERSION IN INFORMATION SYSTEMS

It is impossible to protect information in a system that contains an effective artifice since the mechanisms relied upon to provide protection have been "programmed to fail" under specific conditions. If it is impossible to prove by inspection or testing that a system is free of artifices and effectively impossible to find a known artifice after the system has been built, how can one ever obtain a level of assurance that would protect us from system subversion? To answer this question, we must return to the conditions that make it possible for subversion to exist as a threat in the first place. In the end, we present a viable solution to the subversion threat.

## A. ANALYZING THE THREAT MODEL

We have shown that subversion is a real threat due to the existence of means, motive, and opportunity to conduct subversive attacks in the current environment. Specifically, we have shown that opportunity and motive to mount such an attack exist through presentation of real-world events covered in the press. We have demonstrated by an example of a simple attack that the means to plant an artifice is present in anyone with a modest understanding of the target system and intermediate programming skills.

So, to counter the threat of subversion, we must eliminate at least one of the three conditions that make it possible. Obviously, it would be impossible to reduce or eliminate the subverter's means to subvert the system. The skills required to carry out this type of attack are far too common today. Eliminating the motive is likewise a loosing proposition. To do so, one must either reduce the benefit of mounting the attack or raise the cost of the attack to the point that it becomes prohibitive. Since subversion bypasses the security controls of the system, the attacker gains at least as much benefit from it as do legitimate users. Therefore, reducing the benefit for the attacker would reduce the benefit to legitimate users in equal measure. Recall that subversion is the choice of a professional attacker, who was characterized by a willingness to incur substantial costs in resources and time in the first place. The professional attacker could easily come in the form of a nation state with virtually unlimited resources. Attempting

to raise the costs would likely have limited effect on the motivation of a professional attacker.

One might also attempt to reduce motive through deterrence. Laws combined with a strong law enforcement system discourage bad societal behavior in this way. However, with system subversion, avoiding detection is the overriding concern of the attacker and one source of his willingness to incur significant costs. Any hypothetical law enforcement effort would be mired in its inability to detect the activity. In fact, in some of the early demonstrations by Air Force tiger teams in the early 1970's, some artifices were so undetectable, that the system manufacturer was unable to locate them even when told of their presence and given details on how they worked (Brinkley and Schell, 1995). Moreover, the source of the attack is equally liable to come from outside the jurisdiction of law enforcement as it is to come from within. System subversion is a type of attack that would be considered by a nation state (or state-sponsored organization), organized crime group, or large corporation involved in corporate espionage. Therefore, laws would likely have little to no effect on deterring this threat due to jurisdictional limitations and difficulties in establishing attribution.

We are left then with the task of reducing or eliminating any opportunity to subvert the system. At one level, this involves denying the subverter access to the system at all phases in its lifecycle. Physical security of the development environment, background checks for employees and protected distribution and upgrade channels provide some measure of defense at this level. However, these measures alone are not sufficient to provide assurance that subversion has not occurred in a system. Espionage cases have shown that we cannot solely rely on background checks and security clearances. To do so in the context of system subversion would require us to be certain of the trustworthiness of *all* of the many individuals involved in the design, development, maintenance, etc. of a system over the course of its *entire* lifecycle. Some other control or controls must be put in place such that it would be impossible for the subverter to insert an artifice without it being detected.

## B. SOFTWARE ENGINEERING, OBEJCT-ORIENTED PROGRAMMING (OOP), AND DEVELOPMENTAL ASSURANCE APPROACHES

Software engineering has gained significant ground as a discipline since the time that subversion was first discussed. It attempts to provide a sound methodology for carefully breaking a system down into highly cohesive modules and reduce the amount of reckless programming that occurs within software projects. OOP languages have made this task easier by providing language constructs that enable software reuse as well as data hiding and encapsulation. However, while these measures admittedly bring tremendous benefit to system development, they do not guarantee secure systems. Parnas (1972) provides two modularized designs for the same hypothetical system to demonstrate good and bad examples of modularization.

Likewise, development assurance approaches such as the System Security Engineering – Capability and Maturity Model (SSE-CMM http://www.sse-cmm.org) attempt to address the problem by enhancing the quality of the system that produces the software. The following quote is extracted from the SSE-CMM Vision statement (available at http://www.sse-cmm.org/vision.htm)

> The SSE-CMM, applied to the entire life cycle of products and systems, will maximize synergy between life cycle phases, minimize effort within each phase, eliminate duplication of effort, and result in more secure solutions, with greater assurance and at much lower cost.

These are certainly desirable qualities for any development effort. However, as in most of the quality-driven models, the SSE-CMM cannot reliably address the threat of subversion because the model assumes that all members of the team share the goal of producing secure software. If a subverter is operating covertly on the development team, the model breaks down.

## C. VERIFIABLE PROTECTION

As stated earlier, we have known for a long time how to build systems in a way that allows us to convince ourselves that it is free of artifices. The highest assurance rating (TCSEC A1) is based on criteria that address these issues. However, we must be careful in the use of the term *high assurance* as there are many vendors making this claim of products that simply do not come close to these standards. The line between the colloquial use of the terms high, medium and low assurance are vague and the terms are

easily tossed about in product sales literature and at demonstration booths for nearly every product aimed at security. What are required are systems that not only meet sound security criteria, but that are also built in such a way that we can verify the protection mechanisms they provide. Accordingly, we shall use the term *verifiable protection* from the TCSEC (DoD 5200.28-STD) to describe a system that is resilient to subversion. We will discuss verifiable protection in detail in the following sections.

Take for example the 72-bit adder with carry example (see Section V.B.2 "Security Test and Evaluation (ST&E) Will Fail to Reveal an Artifice"). Dijkstra used this example to illustrate the absurdity of building a 72-bit adder with carry as a monolithic piece of hardware. In practice, designers would build this device by constructing 72 one-bit adders with carry and then linking them together properly. Then, testing of the entire system amounts to checking each of the 72 individual components for proper handling of the four possible inputs and properly passing any overflow condition to the next higher order component. From this perspective, the system has been reduced to modules that can be completely tested. As a result, the operation of the device as a whole has become *verifiable* and we can claim with *high assurance* that it will properly handle *every possible input* without actually testing every possible input.

The simplicity of this example may leave the reader with doubts that such an approach can scale to a large information system. A slightly more complex example is presented in Dijkstra (1968). In the design of the "THE"-Multiprogramming System (a small operating system), he showed that "…it is possible to design a refined multiprogramming system in such a way that its logical soundness can be proved *a priori* and its implementation can admit exhaustive testing." The THE system and the 72-bit adder example are alike in that to test the THE system (or any general purpose computer) at the system interface with no knowledge of how it was constructed would require feeding all possible programs into the system. He goes on to state that testing must only involve what is relevant and what is relevant can be decided only if the internal structure of the system is known. The fact that the set of relevant test cases was small enough to be exhaustively tested was a result of the modular nature of the design. In the context of our problem, recall that one requirement of the Reference Monitor Concept (Section I.C. "Historical Background"), is that security kernel design must be small enough to be the

subject of analysis and tests to assure that it is correct. The exhaustive testing of the security kernel should be possible if it is built with a modular architecture.

### 1. Properties of Verifiable Protection

A system offering verifiable protection will provide a high assurance that its security properties are correct, complete, and allow nothing beyond their specification. Additionally, it will have the following properties:

- Designed to have no exploitable security flaws
- Enforce security policies on information flow, thereby bounding the damage of malicious software (e.g., Trojan Horses).
- Built to be subject to third party inspection and analysis to confirm the protections are correct, complete and do nothing more than advertised (i.e., no trap doors).

One of the primary risks that is addressed by verifiable protection is the risk that trap doors may be present which give an attacker the ability to bypass the system's normal security controls. As stated earlier, the entire system cannot be considered *verifiable*. The size and complexity of the typical system today prohibit such an undertaking. The properties of the system that must be verifiable should be limited to the interface at the Reference Validation Mechanism (RVM) (or the boundary of the security kernel).

There must be a formal model describing these properties, which can be mathematically proven to be sound and correct. An example of such a model is the Bell-LaPadula access control policy model (Bell and LaPadula, 1975). With verifiable protection, all of the operations at the boundary of the RVM must correspond to one or more rules that represent the security policy. Figure 6. shows the work that must be performed in building a system that provides verifiable protection. The formal model describes the security policy. This is mapped to a Formal Top Level Specification (FTLS) which is mathematically proven to be sound. The mapping continues down to any engineering specifications and down through the implementation of the hardware and software through code correspondence. Finally, the executable code (binaries) are shown to map to the source code by analysis of the tools used to generate them. Once this correspondence has been shown, the soundness and correctness of the system's security controls follows by transitivity.

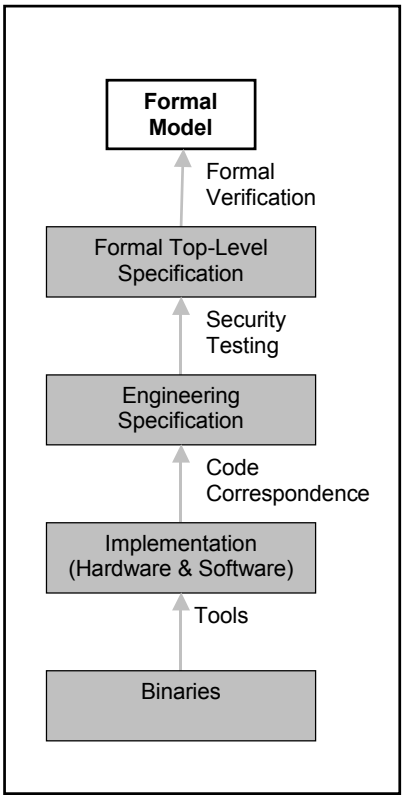



Figure 6. Correspondence of Security Policy to Implementation (After NCSC-TG-010)

A system designed in this way will eliminate any opportunity for the attacker to plant his artifice as any function that does not relate back to the policy model will be discovered in the correspondence mapping.

### 2. Requirements for Verifiable Protection

In order to obtain verifiable protection in a system, it must not only be verifiable, but it also must be possible to audit the correct functioning of the system after the fact. The TCSEC (DoD 52100.28-STD) requires the following for a system to be verifiable and auditable:

> In order for a design and development to result in technology that is "*verifiable*" … it must include at least:
>
> - An information security policy
> - A formal mathematical model of the security policy
> - A formal top level specification (FTLS), having a precise syntax and well defined semantics, that accurately and completely

describes the boundary of the RVM in terms of exceptions, error messages and effects
- A design that uses a complete, conceptually simple protection mechanism with precisely defined semantics
- An implementation that makes effective use of abstraction, layering and information hiding.

For a system's security to be "*auditable*", the development process provides evidence necessary for an effective after-the-fact security assessment including at least:

- A formal proof that the FTLS implements the security policy model. Third parties can repeat such a proof. This is in contrast to a descriptive top-level specification (DTLS), which would require third party participation in the design process to conclude the specification implements the model.
- A mapping of all source code within the Reference Validation Mechanism (RVM) to the FTLS. It is the design process, particularly the production of a formal specification and a layered implementation that incorporates information hiding, that makes this possible. This provides evidence that the implementation is free errors, including trap doors.
- A demonstration that the implementation is consistent with the FTLS
- Functional testing in which the advertised features of a system are tested for correct operation, and it is confirmed
- An information flow analysis of the FTLS
- Configuration management supporting a reliable rebuilding of the security mechanisms. This requires configuration management for hardware, software, firmware, formal specifications and all tools used to rebuild the system. There must exist a protected master copy of all material used to generate the RVM.
- Trusted distribution allowing confirmation that a given instance of the security mechanisms matches an authoritative reference point

## D. CHAPTER SUMMARY

We have shown that the only viable way to mitigate or eliminate the risk of subversion is by removing the opportunity that a would-be subverter might have to plant an artifice during all phases of the system lifecycle. Through proper physical and other security controls, unauthorized individuals can be denied this opportunity to some extent. However, a much more practical and effective approach is verifiable protection. This is the only way one can be sure that the system correctly and completely implements the

protection mechanisms and does nothing beyond its specification. Furthermore, verifiable protection provides the ability to audit the system and ensure that a delivered system has not been altered at some post-production phase of the system lifecycle.

# VII. CONCLUSIONS AND FUTURE WORK

We have shown that the risk of subversion is one that must be addressed in order to have any justification for trust in our information systems. Decision makers responsible for security of information technology should consider this threat when deploying systems. We have also shown that the current trends in approaches to "proving" security are inadequate at best. Penetration tests, add-on third party products, layered defenses and security patches are largely accepted as a means to providing proper security – a practice known long ago to be irrational. This results in a situation that may be even more dangerous than having poor security in the first place, as decision makers operate under the flawed belief that their system provides adequate security or that any breach will be discovered through layered defenses.

We have also known for some time how to address the threat of subversion. Evaluation criteria tried and tested over the past 15 years have been applied to successfully provide appropriate security from a technological standpoint. That these approaches have fallen into disfavor was foreshadowed in an early work by Karger and Schell (1973):

> We are confident that from the standpoint of technology there is a good chance for secure shared systems in the next few years. However, from a practical standpoint the security problem will remain as long as manufacturers remain committed to current system architectures, produced without a firm requirement for security. As long as there is support for ad hoc fixes and security packages for these inadequate designs, and as long as the illusory results of penetration teams are accepted as a demonstration of computer security, proper security will not be a reality.

Much has changed since these early days of computer security. The source of threats has multiplied significantly with the advent of the Internet. Furthermore, our "…reliance on … technology is increasing much more quickly than our ability to deal with the also increasing threats to information security" (Landoll, Schaefer, and Williams, 1995). We continue to nurture a vast industry that provides security as an afterthought in the form of add-on applications that offer little or no assurance. We spend unquantifiable resources reacting to the latest known vulnerability by applying hastily

developed patches, which are tested predominantly by placing them in operation. What we really need is assurance. Not the assurance one hears about at trade shows or in product sales literature, but a verifiable level of protection, which offers not only protection from the amateur, everyday exploit but protection from the professional attacker as well. As it has been attested throughout this paper, we have known how to offer this level of protection for more than thirty years. However, these techniques have seen little application. The few systems that were built were proprietary and lessons learned during their development and in evaluating them is largely undocumented. Knowledge of how to build such systems exists for the most part in a select few individuals who are now scattered among various other pursuits. We must begin to look at these techniques again now - not only for the undeniable fact that the current security posture of the United States demands it, but also for the fact that if we do not begin soon, we may loose much of the costly knowledge we attained in those early years.

Already, there are trends that indicate that this expertise is slipping away from the United States. Criteria-based security evaluations are now conducted against criteria specifications written in the language of the Common Criteria for Information Security Evaluation (CC) (see http://www.commoncriteria.org). Under the current scheme, a Mutual Recognition Arrangement exists between the United States, Canada, Great Britain, and other European member nations to recognize evaluations conducted at the lower assurance levels. This was in part an effort to appease vendors who did not want to spend resources having their products evaluated multiple times in each nation. However, inspection of the evaluated products list (http://www.commoncriteria.org/epl/ProductType/all.html) indicates that the majority of the evaluations above EAL4 are being conducted in Europe. In fact, while the original mutual recognition arrangement extends up to and including Evaluated Assurance Level 4 (EAL4), a set of the European nations have extended this arrangement between themselves, such that evaluations are recognized up to and including EAL7.[10] Therefore, a vendor wishing to have a product certified at EAL7 will get more market recognition for the evaluation by having the evaluation conducted in Europe. To obtain certification

[10] See http://www.cesg.gov.uk/assurance/iacs/itsec/index.htm.

in the United States, he must complete a separate evaluation in one of the United States Labs.

In an attempt to rescue the lessons learned in past high assurance development projects, the Naval Postgraduate School is embarking on an effort to develop an open source high assurance security kernel that offers verifiable protection. This effort will be completely open to observation and participation and will hopefully result in an open source product made available to be used as the foundation for any product that can be built on top of it.

The time for this type of project is now. We cannot afford to wait until a devastating attack occurs at which time, the corporate knowledge gained in the building of products that address the subversion threat may no longer exist. It is critical that those responsible for the security of information technology systems give this threat due consideration.

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES


Alves-Foss, J., (1998), *Multi-Protocol Attacks and the Public Key Infrastructure*. Proceedings of the 21st National Information Systems Security Conference, USA, 1998, (pp. 566-576).

James P. Anderson, (1972), *Computer Security Technology Planning Study* Volume II, ESD-TR-73-51, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA, (Oct, 1972) (NTIS No. AD-758 206).

Bell, D. & LaPadula, L. (1975), *Secure Computer System: Unified Exposition and MULTICS Interpretation*, ESD-TR-75-306, rev. 1, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA (DTIC AD-A023588); also MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA 01730 (Mar. 1976).

Brinkley D., & Schell R. (1995) *What is There to Worry About? An Introduction to the Computer Security Problem.* In M. D. Abrams, S. Jajodia, & H. J. Podell, (Eds.), Information Security: An Integrated Collection of Essays (pp. 11-39). Los Alamitos: IEEE Computer Society Press.

DOD 5200.28-STD. (1985) Department of Defense Trusted Computer Evaluation Criteria (TCSEC).

Fellows, J., Hemenway, J., Kelem, N., & Romero, S. (1987). *The Architecture of a Distributed Trusted Computing Base*. Proceedings of the 10th National Information Systems Security Conference, USA, 1987, (pp. 68-77).

Grant, P. & Riche, R. (1983). *The Eagle's Own Plume*. Proceedings of the United States Naval Institute, July, 1983.

Karger, P. A., & Schell, R. R., (1974). *MULTICS Security Evaluation: Vulnerability Analysis,* ESD-TR-74-193, Vol. II, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA, (June 1974).

Landoll, D., Schaefer, M., & Williams, J. (1995) *Pretty Good Assurance*. Available on-line at http://www.sse-cmm.org/Papers/Pretty.pdf.

Peter A. Loscocco, Stephen D. Smalley, Patrick A. Muckelbauer, Ruth C. Taylor, S. Jeff Turner, & John F. Farrell (1998). *The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments*. Proceedings of the 21st National Information Systems Security Conference, USA, 1998, (pp. 303-314).

Meyers, P. A. (1980) *Subversion: The Neglected Aspect of Computer Security.* Master's Thesis, Naval Postgraduate School, Monterey, California, June 1980.

NCSC-TG-010 Version-1. (1992) *A Guide to Understanding Security Modeling in Trusted Systems.*

Parnas, D. (1972). *On the Criteria to be Used in Decomposing Systems into Modules.* Communications of the ACM, 15(12), 1053-1058.

Schell, R., (1979). *Computer Security: The Achilles' Heel of the Electronic Air Force*. Air University Review, XXX(2), 16-33.

Schell, R., Downey, P., & Popek, G. (1973). *Preliminary Notes on the design of Secure Military Computer Systems*, MCI-73-1, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA (January 1973).

Stevens, R. (1994) TCP/IP Illustrated, Volume 1. Boston: Addison-Wesley.

Weissman, C. (1992). BLACKER: *Security for the DDN Examples of A1 Security Engineering Trades.* Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, USA, 1992, 286-292.

# APPENDIX

This appendix provides a basic introduction to Internet protocols to aid readers in understanding the subversion examples presented in Chapters II-IV.

The Internet is one of the most widely used and widely understood forms of network communication. It is often described as a massive "cloud" through which computers pass data to each other in the form of binary 1's and 0's modulated into an electrical signal. The Internet uses pre-defined protocols to make communication on the Internet proceed smoothly. The most common is the Transmission Control Protocol/Internet Protocol (TCP/IP), which defines how these connections get established, maintained, and torn down. Data are passed around the network encapsulated in *packets*. These packets contain the data (e-mail, documents, pictures, etc.) as well as information about the data (its origin, destination, size, etc.). This data about the data or *metadata* are formed into well-defined *headers*. Packets are sent from one computer to another along a path in the network. Intelligent nodes analyze information contained in the header and decide where to send the packet next thus determining its path through the network. Eventually, the packet arrives at a node that knows where the destination is and the path is complete. This method of connecting one end to another is known as *packet switching*. It is distinguished from *circuit switching* in which a physical end-to-end connection is built by electronic circuits and remains connected for the duration of the communication. Circuit switching is the method used in telephone communications.

The network protocols are organized into a hierarchical layering as shown in Figure 1. Layers of the TCP/IP Protocol. An application (such as an e-mail program generates an e-mail message). The data that comprise this message are formed in such a way that the e-mail program on the receiving side understands how to interpret the data and display the message. Before the message can be sent out however, it must be formed into a packet. So the e-mail program passes it down to the next layer which is either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). The difference between TCP and UPD is not critical here, but this layer is responsible for defining which

application is responsible for the datagram portion of the packet and for guaranteeing that the information is received correctly. The packet now contains the data generated by the e-mail program and the TCP or UDP header. This then becomes the datagram for the next level down – the Internet Protocol IP layer, which will add the IP header. The IP header contains, among other things, identifying information for the origin and destination computers. After the IP layer is the Link Layer. While the IP layer gets the packet from the origin to the ultimate destination, the link layer gets the packet from node to node. Finally, the Physical Layer deals with the physical medium and how signals travel through it.

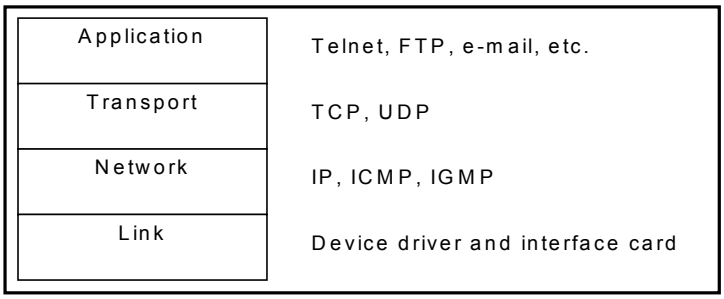



**Figure 1. Layers of the TCP/IP Protocol Suite (from Stevens, 1994)**

On the receiving end, the packet is passed up the stack in reverse order. At each level, the header is stripped off and analyzed until the data is passed to the application (e.g. the e-mail program that will display the e-mail message). As the packet is passed up this stack, it is checked for integrity (in the event that a glitch causes data modification) by comparing a filled-in the header called the *checksum* with the datagram portion of the packet. If the checksum is invalid, the packet must be retransmitted (in the case of TCP), or just discarded and forgotten about (as in the case of UDP).

The trigger in the NFS example (see Chapter II. "High Level Discussion of the Network File Server (NFS) Experiment") is a UDP packet with an invalid checksum of a predefined value. When a UDP packet with a bad checksum is received by the subverted system, it discards the packet normally, but also checks to see if the value in the checksum field is one of the predefined triggers. If so, it triggers the artifice and then discards the packet.

# INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Mr. Richard Clarke
   President's Critical Infrastructure Protection Board
   The White House
   Washington, D.C. 20504

4. Mr. Paul Kurtz
   President's Critical Infrastructure Protection Board
   The White House
   Washington, D.C. 20504

5. Mr. Keith Schwalm
   President's Critical Infrastructure Protection Board
   The White House
   Washington, D.C. 20504

6. Mr. Dan Porter
   SECNAV DON CIO
   1000 Navy Pentagon
   Washington, D.C. 20350-1000

7. Mr. Dave Wennergren
   SECNAV DON CIO
   1000 Navy Pentagon
   Washington, D.C. 20350-1000

8. VADM Richard Mayo
   Chief of Naval Operations
   2000 Navy, Pentagon, N6
   Washington, DC 20350-2000

9. Commander, Naval Security Group Command
   Naval Security Group Headquarters
   9800 Savage Road
   Suite 6585
   Fort Meade, MD 20755-6585
   San Diego, CA 92110-3127

10. Ms. Deborah M. Cooper
    Deborah M. Cooper Company
    P.O. Box 17753
    Arlington, VA 22216

11. Ms. Louise Davidson
    N643
    Presidential Tower 1
    2511 South Jefferson Davis Highway
    Arlington, VA 22202

12. Ms. Elaine S. Cassara
    Branch Head, Information Assurance
    United States Marine Corps
    HQMC, C4
    2 Navy Annex
    Washington, DC 20380

13. Mr. William Dawson
    Community CIO Office
    Washington, DC 20505

14. Ms. Deborah Phillips
    Community Management Staff
    Community CIO Office
    Washington DC 20505
    deborlp@odci.gov

15. CAPT Robert A. Zellman
    CNO N6
    Presidential Tower 1
    2511 South Jefferson Davis Highway
    Arlington, VA 22202

16. Dr. Ralph Wachter
    Office of Naval Research
    Ballston Tower One
    800 North Quincy Street
    Arlington, VA 22217-5660

17. Major Dan Morris
    HQMC
    C4IA Branch
    TO: Navy Annex
    Washington, DC 20380

18. Mr. Richard Hale
    Defense Information Systems Agency, Suite 400
    5600 Columbia Pike
    Falls Church, VA 22041-3230

19. James P. Anderson
    James P. Anderson Co.
    140 Morris Drive
    Ambler, PA 19002

20. Dr. Cynthia E. Irvine
    Computer Science Department
    Code CS/IC
    Naval Postgraduate School
    Monterey, CA 93943

21. Dr. Roger R. Schell
    Aesec Corporation
    25580 Via Cazador
    Carmel, CA 93923