Automated Tools for

Testing Computer System Vulnerability

W. Timothy Polk

December 3, 1992

Contents

1 Introduction **1**

1.1 Intended Audience 2

1.2 How To Use This Document 3

2 Vulnerability Testing Objectives **5**

2.1 Stand-Alone Systems 6

2.1.1 Password Mechanisms 7

2.1.2 User Files 8

2.1.3 System Files 8

2.2 Network Hosts 8

2.3 Summary 10

3 Vulnerability Testing Methods **11**

3.1 Active and Passive Testing 11

3.2 Scope 12

3.3 Local, Network, and Distributed Testing 13

3.4 Reporting Methodology 13

3.5 Summary 14

4 Vulnerability Testing Techniques **15**

4.1 Configuration Review Tests 15

4.2 File Content and Protection 15

4.3 Bug Fixes 16

4.4 Change Detection Tests 17

4.5 System-Specific Testing 18
4.6 Distributed Communications 18
4.7 Artificial Intelligence 19
4.8 Summary 19

5 Policy and Procedures **21**

5.1 Testing Procedures and Responsibilities 21
5.2 Developing a Toolkit 23
5.3 Distribution of Tools 25
5.4 Summary 27

A References **29**

B Primary Tools Reviewed **31**

Abstract

Computer security "incidents" occur with alarming frequency. The incidents range from direct attacks by both hackers and insiders to automated attacks such as network worms. Weak system controls are frequently cited as the cause, but many of these incidents are the result of improper use of existing control mechanisms. For example, improper access control specifications for key system files could open the entire system to unauthorized access. Moreover, many computer systems are delivered with default settings that, if left unchanged, leave the system exposed.

This document discusses automated tools for testing computer system vulnerability. By analyzing factors affecting the security of a computer system, a system manager can identify common vulnerabilities stemming from administrative errors. Using automated tools, this process may examine the content and protections of hundreds of files on a multi-user system and identify subtle vulnerabilities. By acting on this information, system administrators can significantly reduce their systems' security exposure.

Automated vulnerability testing tools are available for a wide variety of systems. Some tools are commercially available; others are available from other system administrators. Additional tools may be developed to address specific concerns for an organization's computer systems. This document examines basic requirements for vulnerability testing tools and describes the different functional classes of tools. Finally, the document offers general recommendations about the selection and distribution of such tools.

1 Introduction

Most modern computer systems have effective controls for implementing computer security. However, many systems achieve considerably less security than their controls could offer due to improper use of those controls or errors in system configuration. The existence of these controls presents an illusion of security to management and users who assume the controls are properly configured. Thus, they maintain sensitive data and applications on the systems as if it offered real security.

Many computer security incidents result directly from such improper use. A General Accounting Office report describes one example:

> The hackers exploited well-known security weaknesses – many of which were exploited in the past by other hacker groups. These weaknesses persist because of inadequate attention to computer security, such as password management, and the lack of expertise on the part of some system administrators... [1]

Other highly publicized incidents, such as the Internet worm in November, 1988 [2], and the DECnet worms (four cases in 1988 and 1989,) [3] [4] [5], exploited similar weaknesses. [1]

This problem is not necessarily due to incompetence; even the most expert administrator may make errors due to the size and complexity of computer systems. The average system supports a wide range of services and a large number of files. The security mechanisms used to control access to services and files must be flexible to address a wide variety of requirements. This flexibility enables users and administrators alike to heighten *or degrade* the security of the computer system.

To ensure that an acceptable level of security is achieved, the administrator should utilize automated tools to regularly perform system vulnerability tests. The tests examine a system for vulnerabilities that can result from improper use of controls or mismanagement. Examples of such vulnerabilities include:

- easily guessed passwords;
- improperly protected system files;
- opportunities for planting Trojan horses; and
- failure to install security-relevant bug fixes.

[1] The Internet worm also exploited errors in the code of the operating system itself.

To identify such vulnerabilities, the testing process analyzes the content of various files in the system and the attributes associated with those files. The number of programs and sheer magnitude of data make it difficult for a system administrator to assess a system's security. An extremely large number of tests and checks may be required. As a result, that review may be feasible only with the assistance of the computer itself.

Software tools are available to aid the system administrator in this task. These tools use the power of the system itself to perform the large number of tests required. These tools will be referred to as automated vulnerability testing tools.

The goal of vulnerability testing is to achieve the greatest degree of security possible, given a particular system. This process focuses on the current state of that system to determine if common vulnerabilities exist.

1.1 Intended Audience

This document addresses concerns of system administrators, security practitioners and information resource managers. It provides guidance on the implementation, selection, utilization, and distribution of vulnerability testing tools.

The primary audience for this document is composed of system administrators and system auditors who are responsible for evaluating the security of systems. These tools provide the means to perform that task. This document assists this audience by providing guidance in the selection of appropriate tools and the analysis of the output.

The secondary audience for this document includes security officers and ADP managers who are responsible for implementing organizational security policy. For this audience, computer system vulnerability testing may be a facet of organizational policy, and a means to verify compliance with policy. The information contained in this document will assist them in the development and evaluation of organizational policy based on these tools.

Finally, this document may prove useful to programmers who are developing vulnerability testing software. It includes a basic list of *objects*[2] to review and some hints about applying the list to particular systems. A number of common techniques for implementing vulnerability testing are also described.

[2] An object is an abstraction for anything that holds data. The most common object is the file. Other examples can be directories, devices, etc.

1.2 How To Use This Document

This document provides guidance on how to:

- determine the types of vulnerabilities that should be considered;
- determine what objects on a system should be reviewed;
- determine how to test those objects; and
- implement a vulnerability testing policy for an organization.

Section 2, Vulnerability Testing Objectives, describes the types of vulnerabilities that can be addressed. Applying these objectives to a particular system is reasonably straightforward; each objective will relate to a set of programs or configuration files.

Section 3, Vulnerability Testing Methods, describes how vulnerability testing may be performed. The appropriate methodology depends upon who performs the testing and the test objectives. Testing a single computer is different from testing a network of computers. Testing performed by the system administrator will also differ from tests performed by the organization's security officer. There are several general vulnerability testing "methodologies"; each applies to certain scenarios.

Section 4, Vulnerability Testing Techniques, describes and classifies common techniques for the implementation of computer system vulnerability tests. These techniques can implement a variety of testing methodologies. These techniques use general computing concepts and apply to a wide variety of systems.

Section 5, Policy and Procedures, includes a variety of recommendations regarding the implementation of a vulnerability testing program within an organization. Recommendations focus upon the selection, distribution, and use of computer system vulnerability tests.

The document is best read in its entirety. However, the relative importance of certain sections will depend upon the reader. The organizational security officer or information resource manager who is developing policy may wish to skip the implementation details in Section 4. Auditors and system administrators will find the policy discussions in Section 5 less relevant than the remainder of the document. Programmers will find Sections 3 and 4 most informative.

2 Vulnerability Testing Objectives

The technical strength of the security in a computer system is a function of the design of its hardware and software. However, the actual security achieved is a function of the way the machine is used. Security is affected by the actions of both the users and the system administrators. Users may leave their files open to attack; the system administrator may leave the system open to attack by insiders or outsiders.

The features used (or misused) frequently involve system or user environment configuration. Two examples are:

- A password system provides some degree of potential security. Users may negatively affect security by using a null password, selecting an easily guessed password, or taping the password to their terminals.
- The discretionary access controls associated with a typical operating system provide some degree of potential security. For convenience, configuration files set system and user defaults for the file protection attributes. This frees users from specifying the protections assigned for every file created. However, the security achieved will be minimal if a user's default file protections are "read/write/execute by ANYONE."

In each of these cases, little actual security is achieved. If a user makes these mistakes, the damage is confined to portions of the system that the user can access. If that user is the system administrator, the entire system is at risk.

In combination, these errors place the system at greater risk than either error alone. If the system administrator's default access control settings allow anyone to alter files, non-privileged users can replace common system executables with Trojan horses. If a user also has a "joe account" (where the userid and password are identical), an unauthorized person might guess the password and gain access to the system. The unauthorized user could install a Trojan horse and gain system administrator privileges.

In each of these cases, the system was vulnerable due to misuse of the system's features. These mistakes occur with alarming frequency. Fortunately, it is simple to identify many common errors using vulnerability testing tools. These tools search for vulnerabilities that arise from common administrator and user errors.

Vulnerability testing tools analyze the *current state* of the system. This is different from activity monitoring or intrusion detection. Monitors and intrusion detection systems analyze *events* as they occur. Vulnerability testing tools review the objects

in a system, searching for anomalies that might indicate vulnerabilities which could allow an attacker to:

- plant Trojan horses;
- masquerade as another user; or
- circumvent organizational security policy.

Anomalies might be the unexpected modification of files, "suspicious" content in certain files, or successful performance of forbidden operations. These anomalies may indicate the presence of a Trojan horse or an opportunity to plant one. The anomalies may also indicate an opportunity to masquerade as another user.

There are basic rules for system security that address these concerns. These rules apply to most computer systems. Automated tools can review the system to verify compliance to these rules. The following sub-sections describe some basic rules. The first proposes rules for stand-alone systems; the second identifies additional rules for systems connected to networks.

2.1 Stand-Alone Systems

To identify vulnerabilities on a stand-alone system, vulnerability testing tools review executables shared among users, and security controls that:

- restrict system access (passwords, smart cards, etc.);
- set the system configuration; or
- set a user's configuration.

Vulnerabilities in the system access controls may allow one user to masquerade as another. The configuration files and shared binaries are attractive ways to install a Trojan horse. Finally, the configuration files set default mechanisms that should reflect your organization's security policy.

It is difficult and time-consuming to review all of these objects by hand. However, a vulnerability testing package can quickly and accurately perform such a review. The typical system has several areas where vulnerability testing tools may be applied. These include:

- identification and authentication systems (especially password systems);
- content and protection of critical system files, such as system configuration files;
- content and protection of critical user files, such as session start-up and configuration files; and
- prevention and detection of changes in system binaries.

In this document, the term *critical file* refers to files whose modification or disclosure could result in circumventing system controls. That is, their modification may allow a user to gain unauthorized access to a system (or resource) or plant Trojan horses.

The following sections develop more specific objectives for the application of vulnerability testing tools to password mechanisms, user files, and system files respectively. These can be applied to a particular system to identify specific vulnerabilities for review.

2.1.1 Password Mechanisms

FIPS Pub 112, *Password Usage*, contains a basic set of rules for password-based identification and authentication systems. [6] FIPS Pub 112 describes ten factors which "must be considered, specified and controlled when ... operating a password system." Of these factors, four are candidates for automated vulnerability testing:

- length: Short passwords are easily broken by exhaustive attempts.
- lifetime: Passwords have a limited lifetime. They should be changed regularly or whenever they may have been compromised.
- source: Passwords that are not randomly selected may be guessed or discovered by a dictionary attack.
- storage: Passwords stored in a computer should be protected to prevent disclosure or unauthorized modification.

Note that the six remaining factors are *not* candidates for vulnerability testing. For instance, ownership is the set of individuals who are authorized to use a password. FIPS Pub 112 states that "Personal passwords used to authenticate identity shall be owned (i.e., known) only by the individual having that identity." Vulnerability testing cannot ensure that individuals have not disclosed their passwords.

2.1.2 User Files

The basic rules for the content and protection of *User Files* are derived by considering the testing objectives. User files must not permit the installation of Trojan horse programs. Users must restrict access to objects they create according to the organization's security policy. The following rules support these goals:

- Protect personal start-up files from modification by others. (These files are ideal candidates for planting Trojan horses since they are ALWAYS executed.)
- Do not specify personal or shared directories before system-provided directories in executable search paths. (This invites the installation of Trojan horses.)
- Default protections assigned at file creation should meet system standards.
- Limit write access in a user's personal file space (by appropriate protection of user directories).

2.1.3 System Files

The rules for *System Files* are developed in a similar fashion. The system configuration files and shared binaries must be protected against Trojan horses and audit trails must be protected against undesired modification. The following rules support these goals:

- Restrict modification privileges for system binaries to systems staff.
- Review the content of system binaries for unexpected changes.
- Restrict modification of systems start-up scripts to systems staff.
- Review content of systems start-up scripts to ensure that secure defaults are specified and programs executed are not candidates for Trojan horse conversion.
- Protect audit trail log files from unauthorized modification.

2.2 Network Hosts

In computer networks, systems typically share data and other resources. This complicates the problem of unauthorized access by adding two new variables: the identity of the remote system, and the relationship between the identities of the users of the

two systems. The networking software creates additional avenues for access to the system. The security mechanisms controlling these access paths must be reviewed for vulnerabilities.

Networking software can allow a user *or system* to access a system or its resources. The original list of vulnerability test objectives must be enhanced to reflect the additional threats. On a network host, the test objectives are to identify vulnerabilities which would allow:

- a user to masquerade as another user *or a system to masquerade as another system*;
- installation of Trojan horses *or penetration by network worms*; and
- circumvention of security policy *by users of remote systems*.

A network host will have all of the potential vulnerabilities of a stand-alone system, as well as the vulnerabilities contained in the network services. Reviewing the configuration of the network will require additional tests, but these tests address the same issues as an audit of a stand-alone system. To extend the stand-alone rules for network hosts, examine the added services with these concerns in mind. The particular additions will depend upon the types of services offered and used by the system.

For instance, a network host may perform all authentications locally. (In this case, the password is transmitted across the network.) Then the testing would include all the identification and authentication rules for the stand-alone system. One additional requirement is needed: network access should not allow users access to the password file beyond that provided in stand-alone mode.

In network environments, many systems rely upon the remote authentication of a user. In this case, the problem is entirely different. The local host relies upon the remote system to authenticate users. The result of the authentication is only reliable if:

- the remote system is known to the system; and
- the remote system's identification and authentication database correlates accurately with the local system's database.

That is, it is important to know the remote users *and* their system. [3]

[3] This does not rule out vulnerabilities related to spoofing of network hosts; this is a function of the network protocols employed.

The concerns for user files are similarly augmented according to services provided. If the users can define remote access capabilities for themselves, the content and protection of those files should be reviewed. [4] For system files, the critical files are any file modifying remote access capabilities and system binaries used in network communication. Vulnerability testing software can review the configuration files, verify that binaries have not been modified, and may even verify the correlation of identification databases.

2.3 Summary

The generic rules presented in Section 2.1 provide a basis for testing any system. The particular rules applied in the testing process reflect the specific features of the system in question. There may be vulnerabilities associated with every resource (such as electronic mail or virtual disk) that the system provides. Examine each resource to determine if additional identification and authentication tests are required or if critical user or system files exist.

These rules extend the basic vulnerability testing objectives to address specific features of the system. If the host supports additional resources, such as a database management system, or maintains a relationship of trust with connected devices, the testing rules must be enhanced to reflect this.

The absence of controls on a system can reduce the set of testing rules. For example, personal computers frequently lack identification and authentication mechanisms or file-level access control. For those systems, the identification and authentication rules do not apply.

Note that many additional security rules are not candidates for review by vulnerability testing tools. For example, a software audit can not detect passwords taped to terminals; this is external to the system. However, software *can* determine if a user can copy the password file.

[4] This is true for certain implementations of the Berkeley Software Distribution (BSD) network utilities, such as *rsh*, *rlogin*, and *rcp*.

3 Vulnerability Testing Methods

Depending upon the objective, vulnerability tests may implement a variety of methods to assess security. Tests may mimic an attacker or simply browse through the system in more typical auditing fashion. Tests may run on the system undergoing audit or may execute on a remote system. Tests may view the system narrowly or broadly.

This section describes several different classifications for vulnerability test programs. Tests are classified according to:

- passive or active testing;
- scope;
- local, network, or distributed testing; and
- reporting methodology.

The following sections define and describe these classifications and suggest appropriate applications.[5]

3.1 Active and Passive Testing

Tests may be classified as passive or active. Active tests are intrusive in nature; they identify vulnerabilities by exploiting them. Passive tests only examine the system; they infer the existence of vulnerabilities from the state of the system.

Consider the example of a password-based identification and authentication system. A password testing program might actually attempt to login with a small set of "easy" passwords. When successful, the program might mail or write a notification of this success to the system administrator. This is an active test. A passive test might involve checking the password file protection, or copying the file and performing off-line encryption and comparison of encrypted strings.

Both types of tests are useful. Where the password file is unprotected, an off-line test is more efficient, more realistic, and more thorough. Active testing may be the only possible method if the test program cannot gain access to the encrypted data.

[5] Note that these classification are not mutually exclusive. For example, passive tests may also be local tests.

However, active tests are more dangerous than passive tests. Active tests can frequently be transformed into a Trojan horse (or network worm) with only minor modifications. Passive tests are usually less volatile. However, both types are useful to an attacker, as in the two types of password tests.

3.2 Scope

Test programs may be classified according to scope. Test programs may examine a single vulnerability or examine the vulnerability of an entire system. The single vulnerability tests have a narrow scope; the system vulnerability tests exhibit a broad scope.

The simplest vulnerability testing programs test for a single specific vulnerability. For example, a test might simply check for unprotected start-up files. By using a series of such tests, it is possible to identify common vulnerabilities. However, such tests do not consider the complete ramifications of the vulnerabilities.

The cumulative effect of a vulnerability may be far greater than it appears. For example, unprotected start-up files allow users to plant Trojan horses. If user X's start-up files are unprotected, and X can modify the password file, any user may masquerade as any other user. This is a simple example; more realistic scenarios can become much more complex.

A single vulnerability test would identify the unprotected start-up files. Another single vulnerability test might report that X was among users who could modify the password file. A system vulnerability test performs many single vulnerability tests, considers the system's access control rules, and determines the complete ramifications for systems security.

System vulnerability testing is more useful than a collection of single vulnerability tests. It is not always possible to correct every specific item flagged by vulnerability testing. A system vulnerability test will assist the administrator in determining the total risk (to the system) posed by a specific vulnerability.[6]

[6] For example, an application might require permission to access sensitive files. A system vulnerability test could help the system administrator evaluate and limit the vulnerability of the system. If the vulnerability was too great, the administrator might wish to disable the application or isolate it on another host.

3.3 Local, Network, and Distributed Testing

Tests may be designed for local testing of a single system, network testing, or distributed testing. Local tests examine the system where they execute. Network tests use communication links to examine the state of a remote system. Distributed tests execute different tasks on each system, according to the system's role.

Most tests are designed for local execution on a single machine. These tests are restricted to the examination of the (virtual) system. They can examine the content and protection of local objects, and remote objects that are available on virtual devices. They cannot examine objects strictly local to remote systems.

Network tests examine the state of remote systems, using communication links to access various services and objects. This type of test permits network security managers to assess compliance with security directives. For example, a network test could determine if insecure network services were enabled by actively probing systems.

This may be sufficient for network hosts that do not trust other network hosts. However, if the host is a member of a distributed system, a remote system performs authentication or access control of local objects. In this case, security-relevant controls and information are distributed among the systems. The testing must analyze components from each host to adequately assess the vulnerability of the distributed system.

To ensure synchronization of controls, distributed tests are needed to compare the configurations of the "related" hosts. Tests that perform this task must consider each host's role in the system and analyze the appropriate components. Accessing the appropriate components often requires local execution, so the tests themselves must be distributed in nature.

3.4 Reporting Methodology

In most cases, test reports are generated for the local system administrators. Test reports might also be returned to a central site for auditing purposes. There is a great difference in the two methods. In the former, the test is a tool for the system administrator. In the latter, the tests are intended to identify systems that pose an unacceptable risk to the network.

As an example, an international network was attacked several times by network worms that exploited the same vulnerability. The network security administrator had issued an edict requiring correction of this vulnerability after the first incident. A network test with centralized reporting would have assisted the network administration in the

identification of non-compliant systems. In combination with administrative procedures (to disable network connections of non-compliant systems), such testing might have reduced the networks vulnerability to subsequent attacks.

3.5 Summary

To summarize the most important points of this section:

- Restrict the use of active tests to circumstances requiring their unique characteristics. Active tests can test any vulnerability, but are dangerously close to a Trojan horse or worm. Passive test programs effectively perform most vulnerability testing tasks and are relatively difficult to convert to Trojan horses or worms.
- Active testing techniques are closely coupled with the specific system. A variety of passive techniques are available; they may be applied to any system.
- System vulnerability tests analyze multiple vulnerabilities and attempt to determine the cumulative effect. This is superior to a battery of single vulnerability tests in which each addresses a specific vulnerability.
- Local testing is employed to determine the vulnerability of a stand-alone system, or network hosts that do not "trust" other hosts.
- Network testing is employed by network security officers to examine the use of "dangerous" services. These are often active tests.
- Distributed testing is required for systems that "trust" other hosts to enforce access control or perform identification and authentication.
- Vulnerability testing tools use local reporting when employed to assist a local system administrator. The tools may use remote reporting when someone other than the local system administrator (e.g., the network security administrator or an auditor) is analyzing security. Remote reporting is also useful when a single system administrator runs multiple machines on a single network.

4 Vulnerability Testing Techniques

This section describes the various techniques which may be used to ensure conformance with the generic testing rules. A single rule may require several different tests, regardless of the testing methodology employed. Each test would employ a different technique to examine a particular aspect of the problem.

4.1 Configuration Review Tests

Modern computer systems are highly configurable, reflecting the flexibility of controls and range of security policies that must be implemented. The relative security of the different configurations can also vary widely. In many cases, a system runs in an undesired and insecure system configuration. This often occurs because that configuration may be the default or the simplest to implement. In other cases, the complexity of the configuration file results in an unintended (and insecure) configuration. Configuration review tests read and interpret the files which represent system configuration information, searching for evidence of vulnerabilities.

Insecure configurations may exist for longer periods of time than most system configuration errors. If a system configuration error results in system failure or impacts performance, users will insist that the error be identified and corrected in a reasonable time frame. Configuration errors degrading security may not be identified until the machine is successfully attacked. Configuration review tests are a reliable method for detecting these errors before a system is attacked.

An example of an insecure configuration is uncontrolled sharing of resources by a network host (e.g., sharing disk partitions with any system on the network). Few scenarios justify uncontrolled remote disk access. However, if the installation scripts default to "export to world," many systems' configuration will never be corrected. This type of configuration problem can be detected by analyzing the content of network configuration files with configuration review tests.

4.2 File Content and Protection

Command files (especially start-up scripts) and system utilities are attractive targets for the insertion of Trojan horses. The integrity of these processes must be protected. A test is required to ensure that no one but the rightful owner can modify the start-up procedure. This is not a simple task, since each process may execute others and these must be protected as well.

Verification of access control settings is the first step in assessing the security of these files. This process is often complicated by the fact that many systems support several access control systems and their interactions are not always clear. In addition, testing the permissions associated with a file may be insufficient.

To be more complete, it is necessary to verify that all programs the command file executes are also appropriately protected. As an example, a UNIX [7] test program might confirm that *rc.boot* is owned by root and protected from modification by all other users. This is insufficient; if any program executed by *rc.boot* is not owned by root and similarly protected from modification, that program is a potential Trojan horse. This type of testing requires reviewing the content of the file, and identifying called programs.

Users can also create vulnerabilities by assigning inappropriate values to user-controlled configuration parameters or environmental variables. In these cases, the content is examined to determine the values assigned to user-controlled variables. This value is evaluated and flagged if insecure.

A simple example is the default file protection attribute that is assigned when a user creates a file. If this value is weak, such as "modify by *anyone*," confidentiality and integrity may suffer. A program could trace through the start-up procedures to determine the value assigned to this variable.

In a more complex example, UNIX systems typically specify the set of trusted hosts for accessing BSD networking utilities (rlogin, rsh, and rcp) in a configuration file. Users can modify this list by placing entries in their own network configuration files. The degree of security provided by the system is directly affected by these settings. A program could read and analyze the content to evaluate the security level.

4.3 Bug Fixes

Operating system bugs can also be a security vulnerability in a system. Some highly publicized attacks, including the Internet worm, have exploited these vulnerabilities to gain access to systems. As a result of these incidents, vendors are making a concerted effort to develop and distribute patches to correct these bugs. However, many system administrators do not keep their systems current by installing the appropriate patches.

The security advisories which announce the availability of the security patch describe simple procedures to determine if the patch is required. These procedures usually

[7] UNIX is a registered trademark of AT&T.

involve verifying sizes, version numbers, or checksums associated with the executables. The appropriate patch(es) can be obtained from the vendor. [8]

This process can be automated with a program that reviews system binaries to verify the installation of security bug fixes. These tests may be active or passive in nature. Active tests attempt to exploit the bug; passive tests use checksums, file sizes, and version numbers to determine if the patch is in place.

The passive tests are extremely limited in nature; they apply to a particular hardware platform and a range of software versions. For example, a passive program which confirms that the UNIX *fingerd* binary is secure might only work on SunOS version 3.X for the SUN-3 series of computers.

The active tests are more flexible. An active program testing the *fingerd* bug could be recompiled and executed on any UNIX system. However, the test program could also be transformed into a worm or Trojan horse with only minor modifications. As with all active tests, distribution must be carefully considered.

4.4 Change Detection Tests

Change detection tests are a class of passive audit tests. These tests are performed to verify that files have not changed since some baseline was established. These tests ignore date and time stamps, which may be faked or corrupted, and rely on cyclic redundancy checks (CRCs) or encryption-based algorithms such as the Data Authentication Algorithm (DAA) [7], which utilizes the Data Encryption Standard (DES) [8]. These tests cannot prevent change or determine what has changed, but are reliable for determining if change has occurred. This sort of test may be performed to ensure that a system program has not been replaced with a Trojan horse.

Change detection tests are sometimes used in a reverse fashion to verify that an update has been installed; if the CRC generated is X, the patch has not been installed. Note that the result of change detection tests will be different for binaries of the same program on different hardware platforms. Application of such tests will be limited to particular hardware/software combinations.

Enhanced techniques are required to review self-modifying executables. In such a case, a "map" specifying the constant portions of the program and the corresponding checksums are required.

Change detection testing can be very effective, but is more demanding procedurally than file content or configuration review tests. The degree of assurance corresponds

[8] Several vendors actually distribute patches via anonymous ftp on the Internet.

directly to the protection of the baseline. It is best to store checksums off-line. Another option is to store the checksums in an encrypted form.

4.5 System-Specific Testing

In some cases, generic testing must be discarded in favor of system-specific testing. Tests must be designed to target specific features of a system when generic techniques are not useful. For example, a review of "industry-standard" passwords would be targeted towards specific account-password combinations unique to each particular operating system.

As a general rule, active tests will be system-specific. They will attempt to exploit specific vulnerabilities by executing system-level or resource-level commands. Configuration review tests are also system-specific. They will test for particular insecure configurations of this particular operating system.

In contrast, change detection tests are entirely generic. The algorithm utilized to create checksums is not related to the system.

4.6 Distributed Communications

Centrally reported tests are a managerial device, designed to assist personnel who manage or audit a large number of systems. However, this device is also vulnerable to eavesdropping when executed on typical networks. Eavesdroppers can "listen promiscuously" and obtain access to confidential vulnerability test reports. These reports may go so far as identifying the vulnerabilities for the attacker.

Results of vulnerability testing should always be secured if central reporting is performed. If the system has any vulnerabilities, the complete results of vulnerability testing will describe them in detail. There are two basic methods for protecting this information.

First, reports can be "sanitized" by reducing the information reported to a simple numeric score. Secondly, public-key encryption techniques can be used to assure confidentiality. These techniques can, of course, be used in combination.

4.7 Artificial Intelligence

Artificial intelligence techniques can be employed when evaluating system vulnerability. Identifying a basic vulnerability in a system is relatively easy. The ramifications of the exploitation of a particular vulnerability are not always so obvious, though.

For example, suppose a user has a null password. Clearly, anyone can now masquerade as that user. The threat is even greater if that user can modify system configuration files. That would let anyone plant a Trojan horse, and so execute programs with other users' authorizations. That makes the null password much more serious. This is a relatively simple example, requiring only two steps. More realistic examples might require a longer series of steps.

To recognize the full ramifications of a vulnerability, a system must be able to identify a series of actions that could be exploited to obtain access or information that could not be achieved by exploiting any single vulnerability. This task is well suited to rule-based artificial intelligence tools. Given the rules for system access, such a tool can quickly determine the "maximum" vulnerability.

4.8 Summary

To summarize the most important points of this section:

- Passive tests are usually sufficient; active tests are dangerously close to a Trojan horse and should be used only in special circumstances.
- Network testing is useful when configuration files on more than one machine must be synchronized or for active testing of critical system access problems.
- Tests which analyze multiple specific vulnerabilities and attempt to determine the cumulative effect are superior to a large number of tests which address only a specific vulnerability.
- Local reporting is used when auditing to assist a local system administrator; remote reporting is used when the network security administrator wishes to analyze the security of the network. (Remote reporting is also useful when a single system administrator runs multiple machines on a single network.)
- There are many techniques for auditing; these techniques may be used in concert to address all facets of a potential security flaw. Most techniques look for a clear "problem"; others look for unexpected change in a system.

- When passive testing is insufficient, generic techniques must be abandoned as well. The basic nature of active testing targets system-specific vulnerabilities. As a result, all active tests will be custom software.
- Centrally reported tests are vulnerable to eavesdropping when executed on public networks. Developers should draw upon the field of secure communications to ensure confidentiality. Failing that, remove sensitive information so that the eavesdropper does not learn of specific vulnerabilities.
- System vulnerability testing is a complex task. It can be implemented by augmenting other testing techniques with rule-based analysis techniques. (Other artificial intelligence techniques, such as neural nets, may also be applicable.)

5 Policy and Procedures

An effective vulnerability testing program can increase the level of computer security throughout an organization. Vulnerability testing is intended primarily to help system managers achieve the maximum security with available tools. Vulnerability testing is also a management tool, underscoring the management commitment to security. Realizing the potential requires that guidelines are in place and adequate tools are provided to the appropriate personnel.

The formulation of guidelines is generally the responsibility of the organization's security officer. Guidelines should specify the procedures for use and distribution of vulnerability testing tools and the responsibilities of organization personnel in the program.

Development or procurement of appropriate tools must also fall to the security officer, perhaps with the assistance of system managers. This process begins by reviewing the organization's systems and developing vulnerability testing requirements in accordance with system functionality. The next step is to develop specific requirements for each type of system. From the specific requirements, the available software can be analyzed for suitability. Remaining holes may be addressed by custom software.

Ultimate success or failure will rest with the system administrator. A vulnerability testing program's primary goal is getting the most security from the available controls. The system administrator must perform the tests and address the indicated vulnerabilities.

5.1 Testing Procedures and Responsibilities

Management should establish systems procedures to ensure that:

- vulnerability testing is a regular procedure;
- vulnerability testing tools are available and complete; and
- vulnerability testing tools that pose a risk to the system are adequately protected from misuse.

This section presents basic concepts for the formulation of vulnerability testing guidelines.

Require regular vulnerability testing of systems.

System managers should perform vulnerability testing on a regular basis (monthly or weekly) and at several critical milestones. The critical moments are: installation or upgrade of system software; modification of user privileges; and an attack (or suspected attack) on a system. Whenever system software is installed, permissions and contents should be reviewed. Installation of new software will also make the baseline for change detection obsolete. When user privileges are modified (such as introduction of new users or adding users to a new group), the system may be put at risk. Finally, whenever an attack has occurred, there is a chance that Trojan horses have been left behind.

Provide vulnerability testing tools to all appropriate personnel.

System managers are the primary beneficiaries of testing tools. However, other members of an organization may also use these tools. Network managers may benefit from these tools; auditors can also use these tools to judge the security posture of systems. The organization's security officer should identify personnel with security responsibilities and take their needs into account in the toolkit development process.

Ensure that adequate tools are available for the most common systems.

Provide access to adequate software for common agency systems through an organizational vulnerability testing toolkit. This toolkit may include:

- locally developed software;
- public domain tools; and
- commercial vulnerability testing packages.

Internet archive sites and system-specific users' groups are good sources for public domain software. These packages are usually distributed in source code, and can be ported to new OS releases. When commercial packages are selected, the purchase of site licenses is a good plan. In any case, supplying the tools from a central site will encourage use of these tools. Requirements for vulnerability testing may be ignored if tools are expensive or difficult to locate.

Develop checklists where vulnerability testing tools are not applicable.

Some items noted in Section 2 cannot be assessed on all platforms. This may be due to a lack of controls or the variety of hardware platforms. For example, an organization might have a dozen types of PC-compatible computers. Many of these systems will lack identification and authentication; those which support it may do so in a non-standard manner.

Most organizational security guidelines address selection of passwords, and vulnerability testing of multi-user systems should examine compliance to those guidelines. However, PC passwords are an example where compliance cannot be assessed. In this case, the vulnerability testing process would include a checklist that would remind the user of the guidelines. The user would simply check the appropriate boxes to verify compliance. For PC passwords, the checklist might appear as follows:

- ☐ Password is at least six characters in length, and is mixed case or includes a digit.
- ☐ Password is not the name of a person or place.
- ☐ Location is physically secured when authorized personnel are not present.

Increase depth of analysis for critical nodes.

Some systems are more important to the organization than others. The tests should reflect this fact. For example, a UNIX-based network gateway is probably more important than a UNIX-based system configured as a single user workstation. The gateway should be subjected to more intensive vulnerability testing techniques. The organization's security officer should identify these critical nodes and determine the level of testing required.

5.2 Developing a Toolkit

The primary tasks involved in the development of an organization's vulnerability testing toolkit are to ensure that the vulnerability tests are complete and the tests themselves do not pose a risk to the system.

There are a number of concrete steps which may be taken to obtain adequate tests.

- Review the organization's systems and develop vulnerability testing requirements in accordance with system functionality. This process is based upon the generic rules presented in Section 2.
- Select appropriate methodologies for each class of personnel (e.g., system managers and network security managers) that will use these tests. This process is based on the information presented in Section 3.
- Develop specific test requirements for each type of system.
- Analyze the available software against the specific requirements (developed in the previous step) for suitability.
- Address unfilled requirements by developing or procuring custom software. (Developing these tests requires considerable knowledge about security but does not require extraordinarily difficult software techniques.)

The following common-sense points should be considered in this process.

Vulnerability testing tools should be comprehensive.

Any single hole in system security can place an entire system at risk. Thorough testing of identification and authentication controls without testing the content and protection of system configuration files will only perpetuate the illusion of security. Computer system vulnerability testing tools should address every applicable item from the generic rules list presented in Section 2.

Active tools should only be used where passive tools are inadequate.

Passive tools are preferable, because of the similarity of active testing tools and automated attack tools. However, active tools may be required to verify compliance in critical cases. Critical cases would include known vulnerabilities which have been or are currently being exploited.

As an example, a network security administrator might use active tools to ensure that critical security vulnerabilities have been closed. These vulnerabilities might have been exploited by known network worms or hackers currently targeting an organization. In such a case, the network administrator may use an active tool to identify systems that have not closed the security hole.

Note that any active network testing tool must be written to execute locally, "probing" the remote systems. The risks of a "good" worm being trapped and modified by hackers would outweigh any possible improvement in security. The worm could be trapped, and simple modifications to the executable would quickly generate a very dangerous attack mechanism.

Borrow from organizations with an established vulnerability testing program.

If an organization has a small number of systems of a particular type, it should look to organizations with a large computing base of this type for assistance. This assistance may include development of vulnerability testing requirements, analysis of available tools, or sharing agency-developed tools.

5.3 Distribution of Tools

There are two viewpoints on the use and distribution of vulnerability testing software. The software can assist a conscientious system administrator in the maintenance of a secure configuration. It can also be used as a tool to assist to a malicious individual attempting to penetrate a system. Those who emphasize its positive potential tend to support wide distribution of such software. Those who emphasize its negative potential are often proponents of limited distribution.

The potential of vulnerability testing tools cannot be achieved unless they are in the hands of the appropriate personnel. This section provides basic guidelines concerning the distribution of vulnerability testing tools.

Distribute passive tools widely.

Passive tools for system administrators should receive wide distribution within the organization. These tools are worthless to an organization if they are not in the hands of the system administrators. Properly used, these tests guard against the type of common mistakes which can lead to simple manual attacks or worm attacks. Many similar tools, and in some cases these exact tools, are believed to be available in the hacker community. Wide distribution of these tools is necessary to place system administrators on an even footing with their adversaries.

However, active tools developed for network or organizational security administrators should be tightly controlled. Whether in source or binary form, such tests represent a serious threat to the organization. Distribution of active tools is an invitation for automated attacks, especially in networked environments.

Source code distribution is usually preferable for passive tools.

Locally developed or public domain vulnerability testing tools can be distributed as source code or in binary form. (Commercially available tools will usually be available in binary only.) The most appropriate form depends upon the organization's computing systems and the personnel who administer the systems.

Source code distributions allow the system administrator to locally compile or interpret the tests on a wide range of systems. This is the simplest way to address the myriad of hardware and software combinations in open systems. Binary distributions would require too much maintenance for most organizations.

However, source distribution has its drawbacks. Source distributions provide a nice, readable description of the security vulnerabilities reviewed. Source code can potentially be modified for use as an automated attack technique. Finally, the systems administrators need to be able to modify and compile the software.

In an organization with homogeneous computing systems, tools can be distributed in binary form. This may be preferable. If the users will be system administrators and auditors with minimal experience, binary distribution may be required.

Secure the distribution process.

The security of the distribution process itself is another important consideration. The distribution process should ensure both integrity and confidentiality of the delivered tools. The distribution process may involve transfer of physical media or may be performed electronically.

If the distribution process involves physical media, security begins with physical control. Registered mail (or similar) delivery can provide assurance that the media reaches appropriate personnel. This is sufficient for some classified information; it is probably good enough for many types of tools.

If this level of security is insufficient (e.g., active tools), encryption becomes the primary method for securing the distribution process. Encryption can be used to provide both integrity and confidentiality. It is critical that the key and software are delivered via different paths.

If electronic distribution is employed, the transfers should be performed from a single, protected server. Several alternatives are available to secure this type of distribution. The server may require identification and authentication (pre-authorized passwords) of users for access. Encryption can protect end-to-end security. CRC techniques can be used to verify integrity. These methods can be used in combination.

If passwords are to be used for authentication, distribution of passwords with a limited life span to system administrators who request the information via e-mail would provide a limited audit trail. Telephone distribution of passwords (rather than e-mail) would provide additional confidence in the authentication.

Protect the software after distribution.

Security measures should not end with the distribution process. Guidelines for safe use should be provided with the software. Similarly, the possible ramifications of unprotected tests should be explained. System administrators should be required to protect the executable files (binary files or command files) and delete the source programs for compiled code.

Address bug fixes.

Central distribution of security-relevant "bug fixes" is appropriate if the organization has a homogeneous computing base. Bug fixes can be archived or distributed via electronic mailing lists. Distribution of security patches should be protected in the same manner as vulnerability testing tools.

Distribution of relevant "bug fixes" to appropriate systems can be very difficult in a large organization with a mixed computing base. The relevance of the bug fix depends on the system's current hardware and software configuration. If the agency has access to a major network, simply provide information regarding system-specific mailing lists where such patches are regularly announced. The administrators of the systems would be responsible for obtaining the appropriate security patches.

5.4 Summary

The organization's security officer should:

- determine the level of testing required for the "typical" system and the frequency with which it is required;
- determine who should have access to security access tools and the testing methodologies they should employ;
- identify critical systems that will require more rigorous testing;

- ensure that adequate tools are available for the most common systems; and
- ensure that appropriate guidelines are in place where testing is unusable.

Possible sources for toolkits are public domain tools, local development, and procurement of commercial tools. Toolkits should emphasize passive tools; they are adequate for the great majority of testing scenarios. The distribution process should be as secure as possible, but wide distribution is imperative. If active testing tools are required, they must be tightly controlled. Finally, procedures for distribution of security-relevant bug fixes must be developed.

The ultimate success or failure of a vulnerability testing program depends upon the system managers, auditors, and resource managers who receive the tools. They must *use* these tools to realize any benefit. Equally important, managers must act upon the data these tools provide. If they do, the level of computer security achieved by an organization can be increased greatly.

A References

[1] Brock, Jack L. Jr., *Computer Security: Hackers Penetrate DOD Computer Systems*, GAO/T-IMTEC-92-5.

[2] Spafford, Eugene, The Internet Worm Program: An Analysis Technical Report CSD-TR-823. Department of Computer Science, Purdue University, November, 1988.

[3] Green, James L. and Sisson, Patricia L., *The "Father Christmas Worm,"* in the 12th National Computer Security Conference Proceedings, 1989.

[4] Longstaff, Thomas A. and Schultz, Eugene E., *Beyond Preliminary Analysis of the WANK and OILZ Worms: A Case Study of Malicious Code* in the Proceedings of the Third Workshop on Computer Security Incident Handling, 1991.

[5] Stoll, Cliff, *An Epidemiology of Viruses & Network Worms*, in the 12th National Computer Security Conference Proceedings, 1989.

[6] FIPS PUB 112 Password Usage, May 30, 1985.

[7] FIPS PUB 113 Computer Data Authentication, May 30, 1985.

[8] FIPS PUB 46 Data Encryption Standard, January 15, 1977.

B Primary Tools Reviewed

A number of tools were examined while performing research for this paper. [9] The majority of these tools were designed for UNIX or VMS systems, although tools exist for personal computers and mainframes as well. This section provides a brief description and references for the major tools or toolkits reviewed.

Note that newer versions have been released for most of these packages. Many other tools have also been developed, but were not reviewed for this project. Information about vulnerability testing software that is currently available for your system may be obtained from your vendor, user groups, or various electronic mailing lists.

COPS 1.3

COPS, or the Computer Oracle Predictor System, is a collection of configuration review tests, file protection tests, password tests, audit trail analyzers, and a CRC-based checksum program for detection of change. COPS is composed primarily of single vulnerability tests, but it also includes a rule-based system vulnerability analyzer. COPS is in the public domain and source can be obtained from a number of ftp sites on the Internet or comp.sources.unix.

The original COPS paper, "The COPS Security Checker System" by Dan Farmer and Eugene H. Spafford, can be found in the Proceedings of the Summer 1990 USENIX Conference.

Clyde Digital Security Toolkit [10]

Clyde Digital's Security Toolkit is a commercial software security package for VAX systems using VMS. [11] Security Toolkit is designed "to assess the security of VAX/VMS computer systems." Security Toolkit is a collection of locally reported passive audit tests. A variety of targets are audited, including:

- user access authorizations;
- capabilities and rights analysis;
- object access, controls and protections;

[9] Inclusion or omission of a particular tool does not imply endorsement or recommendation by the National Institute of Standards and Technology. These examples are cited to credit developers of these tools and to present examples of actual vulnerability testing tools.

[10] Clyde Digital is now part of RaxCo; the Security Toolkit and Security Baseline products have been renamed appropriately.

[11] VAX and VMS are registered trademarks of Digital Equipment Corporation.

- network security and remote access authorization (proxy log-in; protection of network objects; security of network executor);
- VMS audit and sysgen reporting; and
- prior period comparisons.

Clyde Digital's Security Baseline System is available as an option for the Security Toolkit security package for VAX/VMS. Security Baseline is designed to "locate and report discrepancies between current system characteristics and site-defined security standards."

The system consists of three components: *Templates*; *Tests*; and *Baselines*. The Template defines the correct attributes for a set of system entities. The types of entities are pre-defined. Tests are used for the comparison of system-specific entities against a template. Baselines are logical groupings of tests which may be executed interactively or in batch mode.

UNIX-CAATS

NIST and the Department of Energy developed the UNIX-CAATS software package for use by auditors from the Department of Energy Inspector General's Office. This package included a password checker by Dr. Matt Bishop (formerly of NASA/Ames), and a variety of passive, single vulnerability tests. The single vulnerability tests included system and user configuration review tests, network configuration tests, and access control review tests. It also included a file permission verifier similar to SPI/UNIX's File Permission Inspector.

The NIST/DoE toolkit lacked CRC testing and complex file protection tests (which would read files and determine other programs and files which would be executed). UNIX-CAATS is no longer supported; its functionality has been superseded by COPS.

SPAN Toolkit

The SPAN Toolkit includes NASA-developed software for VMS vulnerability testing. This toolkit is available to all VMS systems connected to the NASA Science Internet (NSI). The primary security test components include:

- captive account auditing;
- file-based checksum calculation and comparison;
- directory-based differencing for on-line backup disks;
- dictionary-based password checking;

- a security alarm extractor;
- a terminal timeout and resource control monitor;
- a utility for identifying high-risk accounts; and a
- pronounceable password generator.

The system lacks tools for examining the contents of system configuration files. It may not be obvious which combinations of privileges present an unacceptable risk. The system does not examine capabilities and rights, or object access authorization. When misused, these VMS features may result in unexpected access privileges.

The Security Profile Inspector for UNIX

The Security Profile Inspector for UNIX (SPI/UNIX) is a software package developed by Lawrence Livermore National Laboratory for vulnerability testing of VMS systems. The package is available for distribution within the Department of Energy. (Other agencies may also be able to obtain the package. Contact the Computer Incident Advisory Capability, or CIAC, for further information.) SPI/UNIX is a set of three passive tests which perform the following functions:

- test for easily guessed passwords;
- generate and verify checksums of critical files; and
- save and verify current access permissions associated with critical files;

SPI/UNIX does not review protections of programs called within critical shell scripts or review the content of configuration files.

The Security Profile Inspector for VMS

The Security Profile Inspector for VMS (SPI/VMS) is a software package developed by Lawrence Livermore National Laboratory for vulnerability testing of VMS systems. The package is available for distribution within DoE, and is similar in nature to the SPI/UNIX package. (Other government agencies may be able to obtain the package as well.)

The SPI/VMS package consists of four programs which:

- check for trivial passwords, such as user names, dictionary words, and null entries;

- store, retrieve and verify checksums and time stamps associated with critical system files;
- identify all users who have access to a specified file or files; and
- check a specified list of files for a particular identifier, or for identifiers in general.

Like SPI/UNIX, the package does not concern itself with the content of files. Errors in network or system configuration are not explicitly identified. The system only identifies modifications in content or access parameters of critical files.

Index

active testing, 11
active tools, 24
    distribution of, 25
artificial intelligence, 19

bug fixes
    verify installation of, 16

central reporting, 13
command files
    content of, 16
    protection of, 16
configuration review, 8, 9, 15
CRC codes, 17, 26

Data Authentication Algorithm, 17
Data Encryption Standard, 17
distributed systems testing, 13

encryption, 17, 26

local reporting, 13
local system testing, 13

network testing, 13

passive testing, 11
passive tools, 24
    distribution of, 25
passwords, 26
policy
    testing procedures, 21
public-key encryption, 18

single vulnerability tests, 12
source code distribution, 26
system bugs
    active testing of, 17
    bug fixes, 27
    passive testing of, 17
system vulnerability tests, 12

Trojan horses, 6, 8