UNIX^{TM} & Security

Eugen Mate Bačić
Sr. InfoSec Research Scientist
Canadian System Security Centre
Communications Security Establishment
P.O. Box 9703, Terminal
Ottawa, CANADA K1G 3Z4
InterNet: ebacic@manitou.cse.dnd.ca

Abstract

UNIX^{TM} and security are usually considered mutually exclusive terms, an oxymoron. This paper shows that even the most basic UNIX system can provide effective security and, with modifications, can provide security at the C2 Class as defined in the US TCSEC, or Orange Book. What follows is a description of UNIX security features as well as those features which, if added, would result in an evaluatable system at Class C2.

1 Introduction

This paper is a sort of smorgasbord. It contains information ranging from the every day usage of UNIX[1] to how to take a standard UNIX system and make it C2 compliant.

In writing this paper an attempt was made to convey as much information, and dispel much of the myth, on the security aspects of UNIX. To date many individuals, corporations, and government agencies/departments have shied away from UNIX because of their assumption that UNIX is not secure and can not be made secure. This paper addresses this problem from two perspectives. First, the security features found in many UNIX systems are discussed with emphasis on those aspects which can make the reader's UNIX machine that much more secure. Second, a brief discussion of those features a C2 system require are presented. These features would create a system with basic, evaluatable security enhancements.

Today, many UNIX machines have been evaluated or are being evaluated by the National Computer Security Center (NCSC) in the United States, this includes System V/MLS by AT&T, Gould UTX/32S, Sun's SunOS, and UNIX International's System Vr4.1. The list of "trusted" UNIX machines grows daily and work is in progress in creating secure versions of Minix, OSF/2, and Ultrix. The discussion on the NCSC Class C2 requirements clarifies exactly what many of these systems must meet, at a minimum, to acquire a C2 rating.

Further, a reading list of papers on UNIX and Security is presented at the end of this paper. Although the topic of UNIX and Security is quite interesting, unfortunately a full discussion of this topic would require a more in-depth piece than space allows. The curious reader or one wishing more in-depth coverage of specific features or enhancements will be able to track down appropriate documentation by examining the bibliography and recommended reading lists at the end of the paper.

2 UNIX's Inherent Security

Like most multi-user operating systems, UNIX offers a wide variety of security measures. Unfortunately, because of its roots as a research platform and its development both inside AT&T Bell Laboratories and academia, UNIX has become synonymous with insecure computing.

This is simply not true. UNIX, with a few minor changes, is evaluatable at the C2 Class under US National Computer Security Center (NCSC) guidelines found in the Trusted Computer System Evaluation Criteria (TCSEC) [3]; this is the same Class that VMS 4.3 achieved[2].

In order to evaluate the security of UNIX, first we discuss various aspects of UNIX's present security fea-

---

The viewpoints expressed within this paper are those of the author and do not necessarily represent those of the Canadian System Security Centre or the Government of Canada.

[1] UNIX is a trademark of AT&T Bell Laboratories.

[2] Only VMS version 4.3 has a rating. Digital Equipment Corp. did not bring future versions of the operating system in the NCSC for evaluation. Rumours have it that a future version of VMS will be evaluated at the NCSC at the B1 or above Class.

tures. The C2 requirements are then examined and compared to the features provided by UNIX, detailing modifications and enhancements as required.

2.1 Passwords

Under the UNIX operating system, all passwords are (usually) stored in the */etc/passwd* file. These passwords are encrypted with a modified version of the Data Encryption Standard (DES) [5, 8, 9, 10, 15] so that it becomes a *one way* crypt. When a user logs in, the password entered to the operating system at the`password:` prompt is encrypted using the modified DES routine and compared against the encrypted password inside of the */etc/passwd* file. If the two match, the user is allowed to log into the system.

However, to reduce the effectiveness of key search a defence called "password salting" is used. This method employs the use of a random 12-bit number, the salt, which is appended onto the password when the password is first entered into the system. This string, the 12-bit number and password, is encrypted and stored in the password file. When a user attempts to log in, the salt, stored in the password file, is appended to the supplied password, encrypted, and compared to the encrypted string. As usual, if the two strings match the user is allowed to log in.

The use of salts increases the complexity of breaking any given password by 4096, 2^{12}. Pre-encrypting a dictionary would require that the password breaker encrypt each and every word 4096 times [15]. Unfortunately, if you can grab the password file, you also have the salt!

To solve this problem many UNIX systems have begun to hide the encrypted passwords from casual users. UNIX System Vr4.0, as an example, uses what is known as a "shadow" [7, 8] file. The main */etc/passwd* file exists but it does not contain either password aging information nor does it contain the encrypted passwords. Instead, these are stored in the file */etc/shadow*. The "shadow" file is **not** readable by normal users.

To help ensure the quality of passwords chosen, many UNIX systems now [2, 5, 7, 8, 9, 10, 15]:

- refuse passwords less than a minimum length;
- allow both upper and lower case characters; and
- allow almost all special characters.

In choosing a password, users should ensure that the password can not be easily guessed nor should the password reflect a specific attribute of the user. Choosing an alphanumeric password, say a phrase with 2 digits as separators, will greatly diminish the possibility of password cracking[3].

David Curry [2] offers these excellent guidelines for choosing a password (regardless of which operating system you use):

a) Don't use your login name in any form.

b) Don't use your first or last name in any form.

c) Don't use your spouse's or child's name.

d) Don't use other information easily obtained about you. This includes license plate numbers, telephone numbers, social security numbers, the brand of your automobile, the name of the street you life on, etc.

e) Don't use a password of all digits, or all the same letter. This significantly decreases the search time for a cracker.

f) Don't use a word contained in (English or foreign language) dictionaries, spelling lists, or other lists of words.

g) Don't use a password shorter than six (6) characters.

h) Do use a password with mixed-case alphabetics.

i) Do use a password with nonalphabetic characters, e.g., digits or punctuation.

j) Do use a password that is easy to remember, so you don't have to write it down.

k) Do use a password that you can type quickly, without having to look at the keyboard. This makes it harder for someone to steal your password by watching over your shoulder.

Also, many UNIX systems offer password aging [15]. Thus, when a specific length of time elapses, the user is forced to change his or her password.

To further restrict access to privileged accounts, namely *root*, UNIX allows the system administrator to specify that *root* logins can *only* occur via a *su,*

[3] A phrase like "Little63GTO" is harder to crack than "GTO".

a routine that allows one user to transfer to another user id without logging out and back in. Not only does this ensure that a hacker can break *root* from the standard login, it also ensures that a log of all individuals becoming *root* is kept. This technique is commonly known as restricting *root* to active login sessions.

2.2 Discretionary Access Controls

All UNIX systems contain a form of discretionary access controls [14, 15]. These controls are of the form:

```
-rwxrwxrwx   user   group    filename

where - is one of d   directory
                  c   character
                  s   special
                  b   block
                  -   a regular file
      r is read
      w is write
      x is execute
```

Each file has a filename, an owner (the *user*) and an associated group. The first character in the `-rwxrwxrwx` specifies what type of file it is, the next three indicate the access rights of the user, the middle three indicate the access rights of the group, and the last three indicate the access rights of all other users on the system. A dash (`-`) in any of the last nine (9) positions indicates that that access right is **not** allowed for that particular type of user, i.e.,

```
-rw-r--r-- eugen staff glorbo.tex
```

indicates that the file `glorbo.tex` is readable and writable by *eugen*, but not executable, it is readable by the group *staff* and by everyone else, but is neither writeable nor executable by anyone else.

The file

```
drwxr-xr-x root system bin
```

indicates that the directory */bin* is readable, writeable, or executable (can be searched) by *root*, but is only readable and executable (searchable) by everyone else.

In conjunction with the standard access controls available on UNIX there is a feature called `umask` [14, 15]. `umask` allows the user to define a default file and directory creation mode. `umask` , like many features on UNIX, does not work in the expected way. `umask` expects a 3 digit octal number representing those aspects of the access controls which you wish **not** to give. For example, a `umask` of 777 denies everyone (including yourself) access to any files or directories created from when the `umask` is set. A secure `umask` is 027 which denies write privilege to group members and denies all access to all other users.

2.3 Audit

UNIX audits many things. Common audit items include the last time a user logged in [10, 15]; commands that a user executed during a login session; which users attempted or gained access to *root* privilege; and which users are logged in at any given time.

Most of this data is available to the normal user. If one wishes to view all the commands that one did during the past number of days, issuing a `lastcomm` command on many UNIX machines will list all the commands. Similarly, a `who` or `w` will list all active users[4] currently on the system. Last login time is normally printed at login so the user should become accustomed to examining the login banner to ensure that the last login time mentioned actually reflects the last time the user logged in. A discrepancy should force the user to change his or her password as well as notify the system administrator of a possible breach. The audit records concerning the *root* are never accessible to anyone other than the system administrator.

2.4 Networking

Although a system which allows a network can not be considered "trusted" in the Orange Book sense, most modern day UNIX machines are indeed networked.

The most common network configuration on a UNIX machine today is the NFS [11]system pioneered by Sun. This system allows for transparent connection of a wide assortment of hardware, all fully interconnected, all operating in a distributed manner. At the same time, NFS allows a user to login in from one machine to another via a utility called "rlogin" without having to specify a password [7, 8, 9, 11]. This feature of "rlogin" is known as trusted users/trusted host. The system administrator is allowed to create a file known as */etc/hosts.equiv* which lists all machines to which this machine can gain access to without the user having to reenter a password. Similarly, a file known as *.rhosts*, which resides in the users home directory, offers similar features. These two files were one of the main reasons why the Internet Worm of November 1988 spread so quickly. Users commonly had login ids on various machines. However, usually

[4] On secure UNIX systems this feature may not be available to normal users.

one person per machine had an account on another machine. The worm broke passwords, used various users *.rhosts* and the */etc/hosts.equiv* files and infiltrated more machines.

The solution to halting cascaded logins (getting access to one machine and thus getting access to many) is simply the removal of the */etc/hosts.equiv* and *.rhosts* files. After the Internet Worm many system administrators forced this policy among their users. Having to reenter one's password has not caused the users at these sites undue stress.

2.5 UUCP and E-Mail

`uucp`, UNIX to UNIX Copy, is a means of transferring files between two UNIX machines [15]. `uucp` allows for electronic mail (e-mail) and for the transferring of other information. Setting up uucp on a UNIX machine is rather straightforward but there are common mistakes many system administrators make during setup. Some of these mistakes are [6]:

a) Allowing for all remote machines to access your machine.

b) Trusting the remote machine when it calls in and gives its machine name. To counteract this, use uucp accounts which have a password only the true remote machine would know.

c) Allowing read and/or write access to your machine's entire file system. Read access would allow an intruder to steal your */etc/passwd* file and then attack it at his leisure.

d) Allowing remote execution (via `uux`) on your machine.

Many UNIX systems now offer a simple set up script to ensure that the most common errors are not made when installing uucp. However, because of the vulnerability of uucp, the system administrator must keep a constant watch on the uucp system.

2.6 FTP

Users with Internet access will love the Anonymous FTP (File Transfer Program) feature, however it can be a nightmare for system administrators and UNIX users who did not prepare their systems correctly [6, 8, 11].

FTP is a file transfer protocol which allows for a special type of file transfer. This special type of transfer is known as an "anonymous ftp". A user simply has to enter:

```
ftp glock.speil.com
```

to be connected to the machine "`glock.speil.com`"[5]. When the system requests a login and password the user may enter the user id "anonymous" and any password and will be on `glock.speil.com`. If the system administrators have set the anonymous ftp login correctly, only those files that the system administrator wishes to be seen will be seen. However, if it has not been set up properly, the user may be able to see the entire file system. Regardless of how the system is configured, anything the user can see he can download, and download anonymously!

To maintain the tightest security it is best to remove FTP capability. However, if FTP is desired but one wishes to make it more trustworthy then the system administrator should remove the TFTP (trivial FTP) while retaining the standard FTP software. Or, better still, the system administrator can disable the FTP software and only enable it when it is actually required.

Further discussion on this and UNIX network security features can be found in David Curry's paper [2].

2.7 A Checklist

Patrick Wood [10] supplied this checklist at the NRL/NCSC Invitational Workshop on UNIX and Security.

- Think of some key vulnerabilities of your system:
  - Do you have modems? Are the telephone numbers published?
  - Are you attached to a network? Who else is attached to it?
  - Do you use programs from unknown or unreliable sources?
  - Do you have sensitive information on your machine?
  - Are your users knowledgeable or are they novices?

[5] This is a fictitious name, I hope.

- Do your users have a healthy respect for security?
- Is the users' management committed to security?

- Keep the integrity of your system files secure.
  - Check the permissions of all system files
  - Any program SUID to root is a candidate for replacement by the bad guys
- Be especially careful of the permissions of your device files.
- Be suspicious of files in user directories that are SUID/SGID to system ids/groups.
- Don't mount a user's file system without first checking it for SUID/ SGID programs and device files.
- Keep backups of your disks in a secure area.
- Turn password aging on.
- Keep track of who your users are and which systems they are authorized to use.
- Put in a front end processor or PBX that screens external users.
- Find stale logins and disable them.
- Make sure you don't have logins without passwords.
- Turn on accounting.
- Look for unusual usage patterns, e.g., large disk usage, large amounts of CPU time, large number of processes, lots of su attempts, lots of invalid login attempts, lots of network traffic to a particular system, strange UUCP requests.
- Modify the shell so that it terminates after a period of inactivity.
- Provide security tools to your users.
- Educate your users.
- Modify or enable login so that it:
  - prints the last time a user logged in
  - hangs up the line and exits after three invalid attempts
  - logs unsuccessful login attempts
  - doesn't let root log in anywhere but at the console
  - only allows authorized users access from external lines
  - only allows authorized users access during off hours
  - enforces password aging
  - uses one-time password generators
- Examine software being installed for SUID/SGID programs and ensure that these permissions are necessary.

3 The TCSEC

The Trusted Computer System Evaluation Criteria (TCSEC) [3], or Orange Book, is a set of criteria describing minimal sets of features and functionality for a trusted system to attain in order for that system to be evaluated by the NCSC to a specific class. The TCSEC covers trusted systems from Class D, no trust, to Class A1, as trustworthy as the state-of-the-art allows. Table 1 lists all the possible classes of evaluation within the TCSEC.

Class	Requirements
A1	Verified Design
B3	Security Domains
B2	Structured Protection
B1	Labeled Security Protection
C2	Controlled Access Protection
C1	Discretionary Security Protection
D	Minimal Protection

Table 1: TCSEC Ratings Chart

Although attaining the lower classes is rather straightforward, obtaining a rating at the B1 class and above is much more difficult, and beyond B2 extremely difficult, at the moment. A selection of computer systems are supplied in Table 2. This list should not be considered complete.

3.1 C2 Features

Class C2, as defined in the TCSEC, provides simple, yet fundamental computer security features. Most major operating systems (UNIX, VMS, MVS, CP-6, etc.) either contain or are easily modifiable to contain the features listed below. Most of this section was taken directly out of the US TCSEC, Division C, Class C2.

The Class C2 requirements are:

a) Discretionary Access Control;

b) Object Reuse;

c) Identification and Authentication;

d) Audit;

e) Operational Assurance;

f) Life-Cycle Assurance; and

g) Documentation

Computer System and Manufacturer	Rating
Secure Communications Processor (SCOMP) STOP Release 2.1 Honeywell Information Systems (HIS)	A1
Multics MR11.0 Honeywell Information Systems (HIS)	B2
UNIX System V/MLS Release 1.1.2 American Telephone and Telegraph (AT&T)	B1
VAX/VMS Version 4.3 Digital Equipment Corporation (DEC)	C2
UTX/32S Release 1.0 Gould, Inc. Computer Systems Division	C2

Table 2: Excerpted Evaluated Products List

3.1.1 Discretionary Access Control

Discretionary access controls (DAC) are controls which allow a given user to indicate who may or may not view a file that the given user owns. In its simplest form it allows the user to restrict read, write, and execute permissions to specific individuals listed within the access control list (ACL).

3.1.2 Object Reuse

Object reuse is the requirement that whenever an object within the TCB is no longer required by a given subject, that its contents be erased by the TCB prior to it being returned to the free object area. Thus, if a user program allocates an array of 100 elements and after finishing using it returns the array to the system, the system must ensure that the contents are no longer understandable. This can be accomplished by simply writing over[6] the contents a given number of times by a predefined pattern followed by a random pattern. For a system to achieve C2 it must perform this for objects returned in memory as well as objects returned to disk (i.e., deleted files, removed records within a database, etc.).

3.1.3 Identification and Authentication

Identification and authentication (I&A) is the requirement of the system to require the user to identify him or herself to the Trusted Computing Base (TCB). Once the user identifies himself, then the system will request information in order to authenticate the user's identity. The simplest form of I&A is the login ID and password pair.

3.1.4 Audit

Audit is the requirement that the system maintain and protect from modification an audit trail of all accesses to the object the TCB protects. Access to the audit records will be restricted by the TCB to specific system administrative users. The TCB shall audit:

a) use of identification and authentication mechanisms (i.e., logins);

b) introduction of objects into a user's area (i.e., file creation, program initiation);

[6] Under the Canadian Trusted Computer Product Evaluation Criteria encrypting the contents of the object with a CSE approved device would meet these requirements.

c) deletion of objects;

d) all actions performed by system administrative staff; and

e) other relevant security events (i.e., failed login attempts, etc.).

Each event recorded shall identify:

a) date and time of the event;

b) user;

c) type of event; and

d) success or failure of this event.

And, the system administrators shall be able to selectively audit specific users and their actions.

3.1.5 Operational Assurance

Operational assurance is divided into two aspects, system architecture and system integrity. With regards to system architecture, the system must be able to isolate its resources that are required to be protected so that they can be subject to access control and audit as well as being able to protect itself from external interference or tampering (i.e., by modification of its code or data structures).

To attain system integrity, the TCB must contain hardware and/or software elements which can be used periodically to validate the correct operation of the hardware and firmware.

3.1.6 Life-Cycle Assurance

Life-cycle assurance is composed of various tests to ensure that the TCB functions as indicated in the documentation and that there are no *obvious* methods of bypassing or defeating the security mechanisms of the TCB. Further tests must be done to search for obvious flaws that would allow someone to violate the TCB's resource isolation or that would allow unauthorized access to the audit or authentication data.

3.1.7 Documentation

The documentation required by a system evaluated at C2 are:

- **Security Features User's Guide** which documents the protection mechanisms provided by the TCB, explanations and examples of their use, and explanations of their interaction;
- **Trusted Facility Manual** which cautions the system administrator about various functions and privileges that should be controlled when running a secure facility as well as procedures for examining and maintaining the audit records. A description of the audit record structure for all types of audit event shall be described within this manual;
- **Test Documentation** which describes the system developer's test plan, test procedures indicating how the security features were tested, and results of the testing; and
- **Design Documentation** which describes the manufacturer's philosophy of protection and an explanation of how this philosophy was translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

4 Making a UNIX System C2

It is interesting to postulate what would be required to build a C2 system. As has already been shown, many of the features of class C2 already exist within UNIX. Some may require strengthening while others would have to be included. However, what effort would be required to take an existing UNIX system and make it C2 compliant?

For this exercise we'll assume that we wish to secure Minix [4, 12], a UNIX-like operating system, chosen because of its ready availability in both book and electronic form. Because of its accessibility, readers can obtain a copy of Tanenbaum's book (which includes the source code) and understand the effort required in creating a C2 system[7]

[7] It must be noted that a C2 system must go through an evaluation. This evaluation would look at the additions to the OS at a more in-depth level. Although this section covers all features required to achieve a C2 system, it is not guaranteed that if such a system was presented to either the National Computer Security Center (NCSC) or the Canadian System Security Centre (CSSC) that it would achieve the desired rating.

4.1 Object Reuse

Whenever an object that the system manipulates is returned to a common pool of similar objects, the returned object must have all of its internal information removed. The simplest method of doing this is by writing successive binary patterns over the object followed by a random pattern.

In Minix, there are only 2 types of data that have to be overwritten: data on disk and data in RAM. To implement object reuse it becomes necessary to overwrite the data elements, of any type, as they are returned to the system. Thus, if a user process creates various files, deleting them when it is through with them and allocates various amounts of memory throughout its life, all of these objects (files, memory allocated, etc) must be purged of their information. In the case of system memory, this is easily accomplished by overwriting the returned portion of memory with alternating 0s and 1s, followed by a random pattern. For disk files, or portions of disk allocated for specific user utilization the system must overwrite the entire allocated. In the case of a file this may be multiple i-nodes and the associated disk space. The space, even if the file occupied only a portion of it, must be cleansed. The process to follow is similar to that of cleansing system memory. Associated tables which reside in a publicly accessible location (i.e., i-node tables, etc) must have the object completely removed.

A third, less obvious object reuse requirement is the requirement for Minix to clear the screen once a user has logged out. This "feature" is only viable on CRTs.

4.2 Discretionary Access Controls

The NCSC has stated that the access controls that currently exist within UNIX are sufficient for the C2 [4, 14] level. Guidelines for higher levels of classification are available [20] and reflect requirements of the B3 class.

To ensure an appropriate level of trust in the access controls, audit considerations must be addressed. The controls may require an increased level of protection from attack. This requirement is relative to the existing controls of the UNIX system in question.

Further, it may be required to run the operating system (Minix) in a protected state on the host hardware – in protected mode on a 80386, say. This would further protect the operating system from attempts at breaking or cracking the system[8].

[8] It should be noted that standard practice for trusted systems has been to use hardware that offer some level of hardware protection for the operating system. This would then restrict which hardware platforms a "Trusted" Minix could be written for. Examples of hardware platforms which have a protected state are Motorola MC68020 and MC68040 and Intel 80286, 80386, and 80486

In terms of Minix increased auditing must occur with regards to the access controls.

4.3 Audit

As has been indicated, most UNIX systems currently perform some level of audit. However, to achieve a C2 rating a few additions are required. Minix, our testbed, requires more changes than most UNIX systems [4]. Due to its modest beginnings most of the audit capabilities of larger, more mature UNIX systems are not present in Minix. To achieve a C2 level of trust, the following additions/changes must occur to the existing (modest) Minix audit capability.

All logins and logouts must be audited in an audit file. Failed login attempts must be recorded as well as the port at which entry was attempted. All modifications (creation/deletion/modification) to the user work area (his directory/directories) must be recorded and the modifier listed in the records.

All actions performed by the system administrator while *root* must be audited, in an audit file separate from the standard audit record. All other relevant security events (failed logins, repeated attempts to gain access to a file, etc.) must be audited and logged in a separate audit record of attempted security breaches.

All of these audit files must be protected by the operating system to the fullest extent possible.

Most UNIX systems audit most of the above information, however, C2 requires the separation of the audit records. Further, all audit records (however stored) must be timestamped (date and time of the event), indicate which user was operating at that time, the event/command/breach/etc the user was audited for, and success or failure. Consistent attempts at breaking into the system should result in the attacking user id being disabled for a period of time (system setable) as well as a message being passed to both the console and the system administrator.

4.4 Other C2 Requirements

Minix, like most UNIX-like systems, stores the user passwords (encrypted) in the */etc/passwd* file. To attain C2 would require the use of a "shadow" file to store the encrypted passwords as well as any password

aging information.

4.5 Operational Assurance

The most straightforward method of achieving operational assurance is to use a hardware platform which has protected states. As mentioned under Discretionary Access Controls, certain hardware come equipped to protect the operating system from external interference or tampering. The use of this type of hardware, with the addition of a memory manager, allows for the simple creation of a domain for the TCB's execution. Work to ensure proper usage of the hardware protection features and memory manager would be required.

4.6 Documentation

There are three manuals that **must** be supplied by the manufacturer. These three manuals are the detailed user, the detailed administrator, and the detailed design documents.

During the initial discussions with the NCSC or the CSSC the manufacturer would be told of the requirements and how to meet them with respect to the documentation. Insufficient or incorrect documentation must be corrected before a product can be properly evaluated. During the course of an evaluation a document may go through many revisions as features are either added or dropped.

4.7 The Difficulty of Achieving a C2 Rating

It has probably become obvious to the reader that achieving a C2 rating is not as easy as it sounded at the beginning of the paper. Modifications, many small for most UNIX systems, would require months and months of work. Examination of the operating system's source code and determination if it meets the documentation as well as functioning correctly is an arduous task. Although creating a C2 Minix system is feasible within six (6) months, it would not necessarily receive a rating.

As operating systems become more complicated, the difficulty of achieving a rating beyond C2 becomes enormous. Many vendors are presently working on operating systems which are B1, B2, and even B3 compliant. Many of these systems may not even complete the evaluation process. C2 is, in the opinion of the author, the lowest useful level of security for a modern operating system. Unfortunately the problem does not lie with the criteria, US or Canadian, but rather with the manufacturers and end users. With the requirements on manufacturers to retain "backward compatibility" the manufacturers must keep the bugs and security holes that users want to get rid of. Unfortunately, many of these same "holes" are being used by various end-users. To keep their client base, manufacturers must keep their old operating systems. As users progress into the 90s with networking and desktop computing, new operating systems with security built from the ground up will appear.

5 UNIX Security and the CTCPEC

In 1989 the Canadian System Security Centre released its initial draft, version 1, of a "Made In Canada" criteria, the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC). In December of 1990, a draft of version 2 was released with numerous updates and modifications. Foremost on the list of modifications was the introduction of granularity to the criteria [1]. Instead of approaching the criteria from the point of view of the TCSEC, six hierarchical levels, the Canadian Criteria split the criteria along the lines of functionality (or feature).

This splitting along functionality resulted in a criteria which contains five major portions: Confidentiality, Integrity, Availability, Accountability, and Assurance. Of these five, only Confidentiality, Accountability, and Assurance have their roots in the US TCSEC. Integrity and Availability, still in their initial stages, form an effort by the CSSC to fold these vital aspects of data security into the criteria. At present the Availability portion of the criteria are not complete.

Each major heading has been further subdivided into major aspects of each portion:

Unlike the US Criteria, within a given functionality not all aspects of that functionality has to be implemented. Therefore, if one wishes to make a system which only has Mandatory Protection within the Confidentiality Criteria, this is acceptable. Of course, specific combinations of functionality require other functionality. So, if you created a "Mandatory Machine" it would require some level of Object Reuse. A fuller discussion can be found in [16].

A UNIX system to be evaluated within Canada would be evaluated under the CTCPEC. A manufacturer who wishes to create a C2 system (in US parlance) would be required to create a machine which had:

a) Confidentiality/Discretionary CD-2: Controlled Access Protection

b) Confidentiality/Discretionary CO-1: Object

Confidentiality	Discretionary Protection
	Mandatory Protection
	Partitions: Hierarchies and Comnpartments
	Object Reuse
Integrity	Discretionary Protection
	Mandatory Protection
	Separation of Duties
Availability	(Not Yet Complete)
Accountability	Identification and Authentication
	Audit
	Trusted Path
Assurance	Operational Trust
	Life Cycle Trust
	Documentation

Reuse

c) Accountability/I&A WI-1: Discreationary Security Protection

d) Accountability/Audit WA-2: Controlled Access Protection

e) Assurance/Operational Trust TO-2: Independently Tested

f) Assurance/Life Cycle Trust TL-2: Independently Tested

g) Assurance/Documentation TD-2: Independently Tested

Obviously, due to the mapping required, most functionality is at the "2" level. However, if the manufacturer wished to enhance the audit under the US TCSEC, his rating would remain C2, the Canadian Criteria, on the other hand, would evaluate the Audit controls and give the manufacturer a higher level than the "WA-2" above. A more complete discussion of this can be found in Eugen Bačić's paper on the CTCPEC [16] as well as in the CTCPEC [1] itself.

6 Conclusion

UNIX, with all of its problems, is no worse than any other operating system. In some ways it is actually better. The small kernel and modular design make it much simpler to modify to include security features above the B2 level [14]. Many modern day operating systems can never achieve a rating beyond B1, and for some B1 is impractical. To say UNIX is insecure is an oxymoron, it is better to say UNIX system administrators are insecure.

This paper has showed the UNIX operating system in a light it is not usually shown in. Hopefully the reader can now view the UNIX system not as a security incident waiting to happen but a useful system which is only as strong as the features which are activated by the system administrator and the end user.

Further reading on computer security in general and on Secure UNIX in particular can be found in the bibliography. Over the past few years much effort has gone into research to develop B2 and B3 rated operating systems. As the research progresses operating systems, UNIX and non-UNIX alike, will benefit resulting in systems that are secure and yet easy to use. The end result will be computer environments that are not only productive, but easy and safe to use as well.

7 Bibliography

[1] *The Canadian Trusted Computer Product Evaluation Criteria. Version 2.0* Canadian System Security Centre, Ottawa, Canada. December 1990.

[2] David A. Curry. *Improving the Security of Your UNIX System.* SRI International Publication, ITSTD-721-FR-90-21. April 1990.

[3] *Department of Defence Trusted Computer System Evaluation Criteria.* DOD 5200.28-STD, National Computer Security Center, Fort Meade, Maryland, December 1985.

[4] Albert Donaldson, John Taylor, Jr. & David Chizmadia. *Trusted MINIX: A Worked Example.* Thirteenth National Computer Security Conference, Washington, DC. October 1 - 4, 1990. pp. 307 - 317.

[5] F.T. Grampp & R.H. Morris. *UNIX Operating System Security.* AT&T Bell Laboratories Technical Journal. Volume 63, Number 8. October 1984. pp. 1649 - 1671.

[6] M.S. Hecht, et al. *UNIX Without the SuperUser.* Summer USENIX '87. USENIX Technical Conference and Exhibition, June 1987. pp. 243 - 254.

[7] Sharon A. Kapilow & Geoffrey A. Wilson. *UNIX Security.* EDPACS, Volume XVII, Number 6, December 1989. pp. 1 - 16.

[8] Michael J. Knox & Edward D. Bowden. *UNIX System Security Issues.* Information Age (Security), Volume 11, Number 2, April 1988. pp. 67 - 72.

[9] Emily H. Lonsford. *UNIX Security: Fact Not Fiction.* Datamation, February 15, 1990. pp. 44 - 48.

[10] *NRL/NCSC Invitational Workshop on UNIX^{TM} and Security* (Slide Packages) . Naval Research Laboratory, Washington, D.C. April 1- 2, 1987

[11] Bradley Taylor & David Goldberg. *Secure Networking In The Sun Environment.* Sun Microsystems document, 1987.

[12] Andrew S. Tanenbaum. *Operating Systems: Design and Implementation.* Prentice-Hall, Prentice-Hall Software Series, Englewood Cliffs, New Jersey 07632. 1987.

[13] Dena Tensa. *Typical Weaknesses in Operating System Software.* Information Age (Security), Volume 9, Number 2, April 1987. pp. 74 - 78.

[14] Trusted UNIX Working Group (TRUSIX). *Rationale for Selecting Access Control List Features for the UNIX System.* NCSC-TG-020- A, National Computer Security Center, Fort Meade, Maryland. August 1989.

[15] UNIX Programmer's Manual, Volumes 1 & 2. Bell Laboratories. 1983.

[16] Eugen Mate Bačić. *The Canadian Trusted Computer Product Evaluation Criteria.* Sixth Annual Computer Security Applications Conference, Tucson, Arizona. December 3 - 7, 1990.

7.1 Recommended Reading

David P. Duggan. *Administrative Tools for UNIX System Security.* Sandia National Laboratories. Sandia Report SAND88- 3386-UC. January 1989.

Charles W. Flink II and Jonathan D. Weiss. *System V/MLS Labeling and Mandatory Policy Alternatives.* AT&T Technical Journal, May/June 1988. Summer USENIX '87. USENIX Technical Conference and Exhibition, June 1987. pp. 243 - 254.

Mark Funkenhauser. *B1 TUNIS: A Kernel for a Secure UNIX System.* Proceedings of the 1^{st} Annual Canadian Computer Security Conference. Ottawa, Canada. January 30 - February 1, 1989. pp 153 - 161.

Morrie Gasser. *Building A Secure Computer System.* Van Nostrand Reinhold, New York, New York/Macmillan of Canada, Agincourt, Ontario. 1988.

Steven M. Kramer. *Retaining SUID Programs in a Secure UNIX.* Summer USENIX '88, San Francisco, California. June 20 - 24, 1988. pp. 107 - 118.

Craig Rubin. *UNIX System V with B2 Security.* Thirteenth National Computer Security Conference, Washington, DC. October 1 - 4, 1990. pp. 1 - 9.

B2 Tunis Feasibility Study. Documents #1 - #4. SSC Contract no. W8477- 6- CC04/01SV; Thompson-Foss Incorporated, Contractor. January 1987.

A Bibliography of TUNIS, Secure TUNIS, and Turing (Compiled Papers). Computer Systems Research Institute, University of Toronto. Toronto, Canada. 1989.

USENIX. *UNIX Security Workshop I Proceedings.* Portland, Oregon. August 29 - 30, 1988.

USENIX. *UNIX Security Workshop II Proceedings.* Portland, Oregon. August 27 - 28, 1990.