A DOS Based *POset* Implementation

by Dr. Frederick B. Cohen ‡

In this paper, we describe and discuss a DOS based *POset* (i.e. *Partially Ordered Set*) implementation. We begin with a short review of previous results on *POset* based protection. We then describe implementation details of *POset* based protection under DOS. Next we discuss management tools used to implement and control the protection system. Then we describe the problems encountered in integration into existing DOS environments and how automation is used to resolve these problems and keep management complexity low. Finally, we summarize results, draw conclusions, and describe further work.

search terms: *POset*s, Access Control, Privacy, Integrity, Trusted Systems, Network Protection, Password Protection, Authentication, Trusted Path

‡ This research was funded by ASP, PO Box 81270, Pittsburgh, PA 15217, USA

1 Background

Protection in modern information networks depends heavily on the ability to control the flow of information [1,2,3,4,5,6,7]. A *POset* is the most general mathematical structure for accurately modeling the information flow behavior of a general purpose transitive information network with sharing [1]. The *POset* in this context may also be thought of as a classification scheme, just as the Bell-LaPadula security levels [6], the Biba integrity levels [7], and Denning's lattices [3] can be considered as classification schemes. A *POset* is based on an information flow relation (f) between *domain*s (a, b, c) from a set of *domain*s (S), and is specified in [1] as:

$$\begin{array}{lll}(S,f): & \forall a,b,c \in S, & \\ & (afa) & \text{;reflexive} \quad \text{eq. 1} \\ & (afb) \text{ and } (bfc) \Rightarrow (afc) & \text{;transitive} \quad \text{eq. 2} \\ & (afb) \text{ and } (bfa) \Rightarrow (a=b) & \text{;antisymetric} \quad \text{eq. 3}\end{array}$$

In a DOS based system, we normally have general purpose function and transitivity of information flow [5], so a *POset* is appropriate for modeling and controlling information flow. A *POset* is easily displayed in matrix form, and in this form, corruptive and leakage effects of *domain* collusion can be easily determined by respectively ORing rows and columns of any set of colluding *domain*s to find their effective joint flow [1]. An extension of standard risk analysis techniques [2] for *POset*s reduces the complexity of analysis by taking advantage of the restricted interdependence of *domain*s under a partial ordering [1,4,5].

Limited function *POset* based systems have been prototyped for both PC and a Unix based systems [8,9], and a hardware mechanism for enforcing a *POset* has been demonstrated [10], but no operating system level *POset* protection has previously been reported in a practical system, even though implementation under many existing secure systems appears to be quite straightforward. In this paper, we describe the first fully integrated *POset* based protection in an operating system.

2 Implementation of POsets under DOS

We decided to implement a *POset* based protection system under DOS because modifying the DOS operating system to add protection is relatively easy to do without having access to the operating system source code [11]. DOS has the disadvantage that there is no inherent memory protection to protect the operating system from external attack, so the implementation without additional hardware cannot be very secure. On the other hand, the lack of underlying protection makes DOS a good candidate for add-on protection.

From an implementation standpoint, there are several issues to be addressed. The placement of the mechanism is vital to its proper operation because placing it at a lower level tends to make it less portable, larger, and slower, while placing it at a higher level tends to make it easier to defeat.

The file structure is partitioned into *domain*s so as to provide *POset* structuring, but it must be transparent to the normal DOS user and programmer in order to be practical. There may have to be a *superuser* mechanism with global access in order to provide systems administration and mangement capabilities. Identification, authentication, auditing, installation, and configuration management must also be addressed if the implementation is to be effective. As we will see, some of these design issues are heavily dependent on the structure of DOS, while others have more widespread implications.

2.1 Placement of The Mechanism

DOS system calls and hardware routines are performed via a set of 'interrupt vectors' that point to the location of the operating system routine that performs services. In order to modify the operating system calls, you only have to load the modified code into memory, inform DOS that that region of memory is reserved, store the previous interrupt vectors for use by the modified mechanism, and change the interrupt vectors to point to the new routines. Such a mechanism can be emplanted at any point in DOS operation, but to provide *login* protection, it is best placed during the DOS bootstrap procedure. This is easily done as a device driver or as a replacement for the default DOS command interpreter, and is facilitated in the DOS '\config.sys' file [11].

Unfortunately, such a mechanism can easily be violated by bootstrapping a DOS system from the floppy disk and then modifying the protection on the disk. To prevent this from being done, a 'BootLock' facility can be used [15]. A BootLock facility locks out external access to a disk by modifying the low-level disk descriptor information so as to make it difficult to access without using a special bootstrap program not normally provided with DOS. In practice such a mechanism is easily implemented and there are several commercial packages on the market which provide such a facility.

With this combination of techniques in place, and assuming the *POset* mechanism protects itself from modification by restricting flow into its storage areas, a reasonable degree of protection can be provided. More assurance is usually provided by obscuring disk file structures, using system-wide encryption, installing hardware based protection, or using combinations of these methods.

2.2 Partitioning the File Structure

As a design decision, the root directory and each top-level directory directly under the root directory corresponds to one *domain* in the *POset*. Thus, the *POset* controls the flow of information between these directories only. All deeper subdirectories and files are treated identically to their parent directories, so protection extends recursively down the directory structure. Disk identifiers are ignored, so that the same subdirectories accessible on the default disk drive are accessible on all other disk drives. This design decision has several effects.

The major negative impact is that the directory structure of most current systems is not structured with protection in mind, so there may be restructuring required in order to provide desired information flows. This decision also makes protection very large grain, which may be incompatible with some organizational goals.

The major positive impact is that the control program is very fast (about 100 instructions per decision) because it only has to test the first part of a pathname in order to make a protection decision, and requires only a very small amount of storage (<2Kbytes) for operation [12]. Since DOS only has 640 Kbytes of directly accessible memory and often operates on relatively low performance hardware, these are very important features. Structuring directories for protection also leads to better understanding of protection and ties system structure directly to protection decisions.

To understand how this works, we begin with a picture of a the top-level directories of a simple DOS system. In this case, we have a the root directory with subdirectories 'Dos', 'Asp', 'Src', 'Usr', and 'WP'.

$$\text{Root}\left|\begin{array}{l}\text{Dos}\\\text{Asp}\\\text{Src}\\\text{Usr}\\\text{WP}\end{array}\right.$$

Figure 1 - A DOS Directory Structure

In figure 2, we see a matrix representation of the information flows in a *POset*.

	Asp	Root	Dos	WP	Src	Usr
Asp	f	f	F	F	F	F
Root		f	f	f	f	F
Dos			f			f
WP				f		f
Src					f	f
Usr						f

Figure 2 - A *POset* Flow Control Matrix

In this matrix, 'f' represents a direct flow from the row directory to the column directory, and 'F' represents an indirect flow. For example (by eq. 2):

$$(\text{Asp } f \text{ Root}) \text{ and } (\text{Root } f \text{ Dos}) \Rightarrow (\text{Asp } F \text{ Dos})$$

Next, we look at a picture of the *POset* described in the matrix of figure 2.

$$\begin{array}{cc|c|c} & & \text{WP} & \\ \text{Asp} & \text{Root} & \text{Src} & \text{Usr} \\ & & \text{Dos} & \end{array}$$

Figure 3 - A Picture of the Flow Structure

Note that the *POset* structure is completely different from the directory structure, and can be manipulated independently. No change is required in the directory structure in order to get any desired flow structure. The only problem comes when the contents of two directories are co-dependent. In this case, it is impossible for a *POset* to describe the flow accurately. On the other hand, if the two *domain*s are co-dependent, they are really not separate *domain*s and should be grouped together. [1] In our DOS based implementation, this is accomplished by creating a single top-level directory containing the co-dependent subdirectories.

2.3 Login Procedures

Access to the system is controlled via a *login* program that is placed in the DOS \config.sys file as a replacement for the default command interpreter. The *login* program performs system integrity checks, provides a notice about proper system access, displays notices about auditing trails that should be attended to, and requests a *domain* identity and *password*. After a proper *domain* identity and *password* are provided, the *login* program turns on the protection mechanism and passes control to the default system command interpreter.

[1]Per eq. 3, $[afb]$ and $[bfa] \Rightarrow [a = b]$

If the user fails to provide a proper *domain* identity and *password* after a set number of attempts, the system refuses further entries until the system is rebooted. All failed *login* attempts are logged in an audit trail.

To provide *superuser* access, all that is required is to not turn on the protection mechanism. Because the default mechanism in DOS provides no protection, this makes the entire system accessible without any special provisions.

Once invoked, the mechanism is not designed to be removed, so protection can only be reliably changed by rebooting the system. On a timesharing system, this would be inappropriate, but under DOS, this does not present a substantial problem in most environments. This also limits the exposures due to programmed password guessing attacks. At 'hard-restart', there is a secure channel between the keyboard and the protection software (unless hardware is modified) and there is no other mechanism for trying passwords once protection is turned on.

2.4 Suicides

In order to provide network protection, it is necessary to have a means by which we assure that local protection is operating on the PCs connected to the network. Otherwise, well documented vulnerabilities of commonly used networks can be easily exploited both by automatic and other means. One simple way to do this is with 'suicides'. In a suicide, we create a situation where protection prevents the execution of a suicide program that would prevent LAN access. If there is no integrity checking in place, the suicide succeeds, and LAN access is denied. If there is protection in place, the suicide is prevented, and LAN access is not denied. Although this is not a secure way to assure compliance, it is effective in most LANs.

A simple way to implement a suicide is to replace the LAN *login* program with a program that executes a 'HALT' program followed by the default LAN *login* program as follows:

```
New login :=
            execute(HALT)
            execute(LAN login)
```

We then place HALT in a directory that is inaccessible when protection is turned on. If protection is turned on, the system will refuse to run HALT, but succeed at running the default LAN *login* program, thus granting the user access to the LAN. If the user is using default DOS protection, HALT will be run, the *login* program will not be run, and LAN access will be denied.

2.5 Superuser Implementation

Superuser capability (operation with protection turned off) is implemented with two separate mechanisms; the so-called “trusted program facility”, and the ‘superuser’ user identity.

In the “trusted program facility”, the modified operating system permits one specific program which is stored in a protected area (readable but not modifiable by users under the *POset*) to operate with protection turned off. This “trusted” program must be carefully implemented to assure that it does not introduce protection problems. This is best done with limited function. In our implementation, we provide a customizable facility in source form so that the implementation can be adapted for local needs. We only provided facilities for changing passwords in this facility.

In the case of the ‘superuser’ user identity, a specific user identity is provided for which protection is never turned on. In this case, the *POset* based protection is never loaded into the system, and the system operates as a normal DOS based system.

2.6 Multi-user Domain Models

Although the *POset* protection model provides a nice mandatory access control facility, there is a legitimate use for providing multiple user identities as access points to any given *domain*. As an example, we can implement a Bell-LaPadula (or Biba) based protection system with security (or integrity) levels [6,7] by setting flows as shown in figure 4.

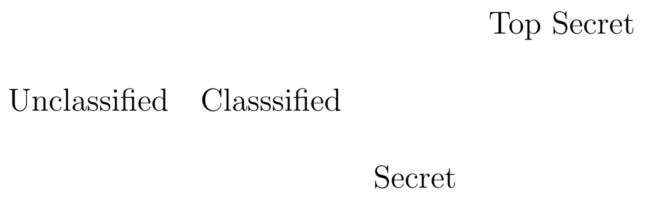

Figure 4 - A Security (Integrity) Model *POset*

The problem with such an implementation is that we cannot distinguish individuals from a standpoint of auditing, we cannot take access away from a user without changing the *password* for the *domain* and notifying all of the legitimate users of the new *password*, and a user with access to multiple *domain*s must learn the common *passwords* for each domain. A simple way to implement multiple users is to provide multiple identities with independent *passwords* access to a given *domain*. We can then audit based on the identity and *domain*, provide separate *passwords* to each user in each *domain*, and take access by eliminating access for a given user in a given *domain*. A user can have their own *passwords* for each accessible *domain*, and as long as these *passwords* meet system-wide *password* requirements, they do not have to common with other users.

In implementation, we decided to name users within *domain*s with a 'domain.name' convention wherein a user 'joe' in a *domain* 'secret' uses the user identity 'secret.joe' to *login* to the system. Each 'domain.name' has their own *password*, and audit trails reflect this full identity. Each *domain* also has a *domain* identity (e.g. 'secret') which can be used for generic *login* if enabled.

3 Management Tools

Password maintenance, audit trail generation and maintenance, exposure analysis, protection configuration, and protection of the protection mechanism have all been widely cited as vital components of a proper protection environment. Although they are not more critical to a *POset* based system than other systems, the description of how they operate in our implementation may be helpful to others who may follow. These issues are dealt with by a set of management tools designed to automate as much of the control and assurance as possible.

The import of management tools to protection management cannot be underestimated. To clarify this point, let's look at a simple example. The default DOS protection requires 5 bits per file, and even a moderate DOS installation on a single PC has about 1,000 files, so the total protection state is about 5,000 protection bits. The standard Unix protection mechanism takes over 10 bits per file, and a typical single user Unix system has several thousand files, so the total protection state has tens of thousands of bits. A typical DOS based local area network has about 100 PCs, so the protection state of such a network typically consists of over 500,000 protection bits! Furthermore, most protection management systems provide only bit by bit control over the protection state. The large number of protection bits combined with the fact that they are controlled one bit at a time, makes accurate maintenance of protection very difficult.

This problem is further exacerbated by the fact that in most systems a single error in a protection bit translates directly into denial of service or granting of excessive access. In most current systems, these sorts of errors are widespread [13,17].

3.1 Managing Structure

The large granularity of protection provided by this *POset* implementation reduces complexity by reducing the total number of protection bits to less than $n^2/2$ bits [2] for n *domain*s [1]. A typical DOS system requires only about 20 domains, thus requiring control of only 200 protection bits. For 100 domains, this comes to only 5,000 protection bits, which is more than enough for most LANs, and two orders of magnitude less complex than typical current networks.

It usually turns out far better than this because most *POset*s can be determined from a smaller generating set. A simple technique is to create a sequence of known dependencies and automate their inclusion when they apply. For example, with less than 20 characters per software package, we typically provide generating information for fully automatic *POset* configuration. In networks we have seen, fewer than 50 lines are required.

In order for *POset* protection to be properly implemented, it is necessary to limit the protection bits so as to meet the mathematical requirements of the structure. Since people are generally unable to do this manually without a great deal of effort, automation is required to facilitate the process. In our implementation, we provide automation to make this process simple and easy for the typical systems administrator. The underlying routines provide a means for adding or removing flows between *domain*s only if they don't violate the *POset* structure, ordering, displaying, saving, and restoring the structure, and converting the stored structure into a form suitable for the operating system driver. To add flows, these routines compute transitive closure for each *domain* in the *POset* on a stepwise basis, refusing flows that would violate the *POset* structure [1].

3.2 Managing Passwords

People have historically tended to select easily guessed *passwords*, and a great deal of time and effort has been spent to reduce this problem [14,15]. Common techniques include limiting minimal *password* length, providing system generated *passwords* or pass-phrases, algorithmic authentication, biometric devices, challenge response systems, and electronic time variant keys. We took the decision to use a standard *password* system except that we require *passwords* to meet an organizationally controlled minimum standards for length and content, provide automatic *password* and pass-phrase generation, and show the expected time to guess the *passwords* given a fairly sophisticated and knowledgeable attacker [14].

[2] A *POset* can always be represented by an upper triangular matrix, but not all upper triangular matrices are unique *POset*s. It also turns out that there is an easy to generate 2-level example with $n^2/4$ unique *POset*s. Thus complexity is $O(n^2)$, bounded from above by $n^2/2$, and bounded below by $n^2/4$.

By default, *passwords* are set to prohibit all access. The *superuser password* is entered at installation and forced to meet minimum standards. By default, after installation is complete, the systems administrator must *login* and provide *passwords* to authorized domains. The administrator can remove all access or modify *domain passwords* to change authorized access. Automated installation can also provide default values for various *domain passwords* so as to eliminate any user entry of *passwords* or force automatically generated *passwords* to be used for user accounts. New top-level directories are only accessible after the *superuser* defines appropriate flows. User *login* to new top-level areas is only possible after the *superuser* defines a *password* for that *domain*.

3.3 Exposure Analysis

Exposure analysis with *POset*s is easily automated [1,4]. We start with the assumption that all *domain*s have equal unit values and calculate direct and indirect exposures by summing all *domain* values in the transitive closure of information flow. This is done both for leakage and corruption values. The *superuser* can modify values and structure and have exposures calculated automatically. Values can be saved and restored, and the presentation operates in a fashion similar to a spreadsheet. This system can be extended so as to limit *POset* structure in order to enforce acceptable exposure limits [1,4], but this has not yet been implemented.

3.4 Managing Audits

Audit trails are provided for failed *login* attempts and other non-*POset* related security relevant events. In addition, the *login* process assures that audit trails will not be ignored by forcing audit information to be displayed. Audit trails are only used to track failures, so as they grow larger, they indicate increasing numbers of violation attempts. At various thresholds of failure numbers, the *login* process first notes, then notifies, then warns about, then automatically displays audit trails indicating violations. This has the effect of forcing the user to examine and clear audit trails. Audit trails can be examined, cleared, or eliminated altogether by the management automation facilities, and contain the time, date, action, parameters, and reason for the failure.

4 Operational Experience

The DOS based *POset* implementation has been operating since late in 1990, and as of this writing has been in everyday use by a substantial number of users for several months. Before being released to test sites, the system was operated for several weeks in our research facility to determine how it would interact with a variety of other packages. It has now been used in conjunction with over 50 PC based software products including spreadsheets, database systems, document processing systems, system utilities, customized software packages, accounting systems, memory managers, and network controllers. We now present the results of our limited operational experience.

4.1 Changes to the User

The average user using an application is only aware of the fact that a *domain* identity and *password* are required in order to use the system. After *login*, many users are placed in a menu system that provides selected services. As long as the system administrator performs adequate configuration testing prior to installation, there are no operational differences; protection is completely transparent and automatic.

In systems with more sophisticated users, configuration control is not quite as simple, but it is still easily automated and transparent to the user. In most development environments, operation proceeds exactly as if protection were not present. Several program developers have used the system without any side effects whatsoever.

For some system utility design and other similar tasks, it is necessary to perform operations that violate the *POset* policy. For example, installing new packages for common use, modifying system configuration files, and installing new device drivers, typically call for full access to the system. In most cases, limited understanding is required for these tasks, but considerable understanding of the *POset* structure must be used in order to perform systems management in the presence of systems programmers. This is roughly comparable to the situation in a typical timesharing system, except that file by file protection issues can be ignored in favor of the larger grain protection decisions of the *POset*.

In several months of intensive systems program development and testing, the protection issues added by the presence of the *POset* were found to be far less than the difficulty required for similar tasks in a Unix system with standard Unix based protection.

4.2 Software Compatibility Issues

One of our major concerns with DOS based software is that it tends to assume that the entire system is available for its use. At first we thought that this would be a very serious problem, but it appears that locality of reference holds for most DOS programs we have encountered.

The difficulty that would arise from non-local reference in a *POset* based system would be from software packages that require modification to data placed in a specific path not accessible to users using the package. In such a circumstance, it would be necessary for the user to be granted flow both from and to that area, and this would create a co-dependency.

For example, if a program stored in the 'WP' *domain* in figure 3, had such a restriction, a user logged into the 'Usr' area could not make the required modification. This is related to a serious security problem in most Unix based systems where the '\tmp' directory is accessed by numerous programs. Any user trying to prevent information from external access cannot use the standard systems programs (e.g. the compiler) without placing potentially confidential information in a commonly accessible area. Software with this property makes effective protection very difficult to implement. [3]

One well known DOS editor reconfigures its stored binary image on disk in response to changes in outward appearance. Another well known database package stores all of it's intermediate files in a specific disk path. In these cases, a user without access to affected areas cannot successfully use the package. On the other hand, any user with access to those areas can modify information used by any user that accesses the package, and is thus granted more privilege than may be desired.

Another problem we thought we would encounter very often was opening of files with both read and write access when only one or the other access is actually required. Programs that don't follow the 'least-privilege' convention will tend to be unable to access information because they request too much access.

In our experiments, it turned out that none of these were a significant problem because the programs tended to only change their configuration files under relatively limited circumstances. Thus locally loadable temporary configurations can be used on a session at a time basis without difficulty. In some cases, we had to *login* to one *domain* to do program configuration and another for normal usage, but this is entirely appropriate for protecting the program from unauthorized modification. In some limited cases, we felt the integrity of

[3] On IBM's VM system, this problem is resolved through the use of 'virtual machines' for each user, while under AT&T secure Unix, this is handled by mapping the \tmp area into separate physical disk areas for each domain.

the application program was not worth extra effort and placed the application in the same domain as its users.

With only minor program changes, the locality of reference problem can be solved both for Unix and for DOS based applications, and we fell that this is an important precursor to effective protection in these systems.

4.3 Configuration Issues

Another major concern was that the complexity of configuration management would be unduly high. It turned out that this was not the case, since most DOS programs depend only on the basic operating system functions and the programs provided with the package. Thus we had a very simple structure similar to the one shown in figure 3. In fact, we can almost always configure all applications in parallel and all users at the level where 'Usr' is in figure 3.

This provides very rapid operability for a set of independent users, but it does not provide for more complex configuration management capabilities such as those required in a typical protected database application. As an example, we have a database designed for data entry by users in one set of *domain*s and data analysis by several other users in other *domain*s.

$$\begin{tikzcd} & \text{Entry}_1 & \text{Analysis}_1 \\ \text{Dos} & & \text{Analysis}_2 \\ & \text{Entry}_2 & \text{Analysis}_3 \end{tikzcd}$$

Figure 5 - A More Complex *POset*

In this example, the normal DOS areas are readable by all user *domain*s, but data entered by Entry_1 and Entry_2 can be analyzed in different combinations by Analysis_1, Analysis_2, and Analysis_3. As a practical example, imagine that salary data is entered from Entry_1, while name and address information is entered in Entry_2. Analysis_1 can get salary figures and compute aggregate information, but it cannot easily link specific employees to specific salaries. Analysis_2 can generate paychecks and access all employee information. Analysis_3 can access all employee information except for salaries. [4]

[4] There are statistical techniques for analyzing aggregate information to derive specific information [3].

4.4 Problems with Low Level DOS Utilities

The only major problems we encountered were from systems programs that bypass DOS and directly access low level file structures on the disks. For example, the 'Norton Utilities' disk compression utility and the DOS 'Chkdsk' utility compare the low-level file structure to the DOS file structure. Since large portions of the DOS file structures are not accessible through DOS with protection in place, these utilities find numerous inconsistencies. The resolution to this problem is to only use low-level utilities when global access is provided. This is consistent with the use of low-level utilities in Unix and other protected systems. This is facilitated by providing a special *domain* for low-level system programs.

4.5 Networking Experiences

Our implementation of the *POset* system is based on DOS file structures rather than lower level structures, and as such it provides a very transparent interoperability with current networking technologies. In essence, decisions are made at the file level, and since most modern networks map network file server areas into user file structures, the protection scheme operates over a network in the same manner as it operates on any other disk on a system. To our surprise, the first networked implementation operated without any modification whatsoever. In the tens of networks now installed, no modification has ever been required in order for protection to operate properly, and we suspect that this will continue to be the case for virtually all current networks.

5 Remaining Vulnerabilities

Although the *POset* implementation under DOS was successful, it should be obvious that this protection is severely limited due to the inability of the **PC** to restrict user processes from examining and modifying operating system memory. In addition, without disk encryption, it is impossible to protect the disk areas from being examined or modified by brute force. Since this weakness exists for any purely software based protection mechanism for the **PC** which allows user programs to gain processor control, we detail these attacks here to clarify the basic **PC** weakness.

5.1 The Memory Modification Attack

Since any DOS process can modify the entire memory, an expert user with access to the ‘debug’ program provided with DOS (or any other general purpose DOS program) can trace the operation of system calls and track down the instructions in memory used for making protection decisions. Once this is done, the memory can be modified so as to bypass protection, auditing, and all other protection mechanisms discussed herein and most other purely software based protection mechanisms in use today.

5.2 The Brute Force Disk Attack

Without some sort of disk encryption, an attacker can examine the contents of disk so as to reveal its contents. Even though the ‘BootLock’ facility prevents trivial examination of a disk, an attacker with some persistence can create an operable partition table for the disk using brute force to find appropriate parameters. This soon reveals the contents of the disk to the attacker.

5.3 The Need for Physical Security

This then serves to point out one of the principle difficulties of securing a PC. If we assume attackers with physical access to a system, sophistocated maintenance tools, intimate knowledge of the system’s internals, and time to launch an attack, there are very few protection systems that are likely to succeed. Without physical security, there is no security.

As one final example to clarify this point, an attacker with physical access could easily replace the external connections to the keyboard and display so as to transmit all input and output to the attacker, and enable the attacker to type on the legitimate user’s behalf.

6 Summary, Conclusions, and Further Work

We have described a DOS based *POset* implementation and described how implementation decisions were made and why. The decision to partition *domain*s based on top-level directories proved successful and made automated configuration management very easy and effective. Implementation did not disturb normal DOS or LAN operations, and the protection provided has proven to be both comprehensive and easily managed, although due to the lack of memory protection in DOS it remains insecure against sophistocated attackers. Automated installation has successfully provided protection without disturbing user operations in almost every case. Many of the problems we anticipated were not realized because of locality of reference and the large granularity at which protection is maintained.

Although the implementation of *POset*s has only been in widespread use for a short time, we must conclude that *POset* based protection is both feasible and practical and provides numerous benefits to both the user and manager. We believe that *POset* implementations are practical for all modern operating systems, provided a small amount of care is taken to assure that locality of reference is maintained wherever possible. This requires little or no modification to existing software, while providing effective and easily managed protection. It appears that replacing current mandatory access control mechanisms with *POset*s will have numerous benefits and few if any drawbacks.

Further work will be required to implement *POset* based protection on a wide variety of systems and evaluate their applicability as a replacement for the current generation of protection mechanisms. We look forward to numerous other implementations.

7 References

1. F. Cohen, "Protection and Administration of Information Networks with Partial Orderings", IFIP-TC11, "Computers and Security", V6#2 (April 1987) pp 118-128.
2. T. Saltmarsh, P. Browne, "Data Processing - Risk Assessment", Advances in Computer Security Management, V2, 1983.
3. D. Denning, "Cryptography and Data Security", P279-280, Addison Wesley, 1982
4. F. Cohen, "Design and Administration of Distributed and Hierarchical Information Networks Under Partial Orderings", IFIP-TC11, "Computers and Security", V6#3 (June 1987), pp 219-228.
5. F. Cohen, "Computer Viruses", Dissertation at the University of Southern California, 1986.
6. D. Bell and L. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", The Mitre Corp. 1973
7. K. Biba, "Integrity Considerations for Secure Computer Systems", USAF Electronic Systems Division, 1977
8. F. Cohen, "Design and Protection of an Information Network Under a Partial Ordering: A Case Study", IFIP-TC11, Computers and Security, V6#4 (Aug. 1987) pp 332-338.

9. F. Cohen, “Two Secure Network File Servers”, IFIP-TC11, “Computers and Security”, V7#4, August, 1988.
10. F. Cohen, “Designing Provably Correct Information Networks with Digital Diodes”, IFIP-TC11, “Computers and Security”, V7#3, June, 1988.
11. J. Mueller and W. Wang, “The Ultimate DOS Programmer’s Manual”, 1991, Windcrest Books, Blue Ridge Summit, PA 17294-0850, USA
12. ASP Press, “The ASP Integrity ToolKit - Version 3.3”, 1990, ISBN#1-878109-12-X, ASP Press, PO Box 81270, Pittsburgh, PA 15217, USA
13. F. Cohen, “A Short Course on Computer Viruses”, 1990 ISBN#1-878109-01-4, ASP Press, PO Box 81270, Pittsburgh, PA 15217, USA
14. F. Cohen, “Algorithmic Authentication of Identification”, Information Age, V7#1 (Jan. 1985), pp 35-41
15. See many fine references in the IFIP TC-11 journal “Computers and Security”
16. F. Cohen, “A Note On High Integrity PC Bootstrapping”, IFIP-SEC “Computers and Security” (submitted, 1991)
17. R. Lefkon, Proceedings of the 2nd annual DPMA computer virus symposium, New York, NY, 1989.