Proceedings of the 2007 IEEE
Workshop on Information Assurance
United States Military Academy, West Point, NY, 20-22 June 2007

A Linux Implementation of Temporal Access Controls

Ken Chiang, Thuy D. Nguyen, Cynthia E. Irvine

Abstract— **Control of access to information based upon temporal attributes can add another dimension to access control. To demonstrate the feasibility of operating system-level support for temporal access controls, the Time Interval File Protection System (TIFPS), a prototype of the Time Interval Access Control (TIAC) model, has been implemented by modifying Linux extended attributes to include temporal metadata associated both with files and users. The Linux Security Module was used to provide hooks for temporal access control logic. In addition, a set of utilities was modified to be TIFPS-aware. These tools permit users to view and manage the temporal attributes associated with their files and directories. Functional, performance, and concurrency testing were conducted. The ability of TIFPS to grant or revoke access in the future, as well to limit access to specific time intervals enhances traditional information control and sharing.**

I. Introduction

In many situations, access to information should not be perpetual. For example, limited temporal availability could be applied to items ranging from student exams to medical prescriptions. Exams should be available to students during a pre-determined exam period, whereas prescriptions might only be valid for a few weeks or months after they have been written. Current access control systems do not provide a conceptually simple and complete mechanism for modulating access of subjects to files based upon temporal attributes: a start time when access is allowed and a stop time when access is revoked.

Afinidad formally modeled temporal access control in the Time Interval Access Control (TIAC) model [1], [2]. To understand the practical design implications of such a system, a prototype implementation of the TIAC model has been developed for the Linux operating system. The Time Interval File Protection System (TIFPS) consists of a modified Linux Security Module that implements the TIAC access control logic [3]. Extended attributes are used to associate temporal metadata with files and directories. To demonstrate the usability of TIFPS at the application level, a number of file management utilities were modified to take advantage of the temporal access control mechanism. Our initial implementation requires the administrator to set temporal attributes on executables, viz. the login shell, and objects prior to user access to the system. Implementation of temporal access control mechanisms in the operating system offers several advantages: a consistent and coherent abstraction presented to all applications; an encapsulated mechanism; and protection of the mechanism from arbitrary modification by applications. The TIFPS prototype adds another layer of protection against unauthorized access to files and directories, and serves as a starting point for experimentation in temporal access control systems.

Irvine, Nguyen, Chiang: Department of Computer Science, Naval Postgraduate School, Monterey, California

Chiang now at Sandia National Laboratory, Livermore, CA

The authors are grateful for support from the Office of Naval Research and the National Science Foundation under grant CNS-0430566. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ONR or the NSF.

II. Background

This section briefly reviews related work in temporal access controls contrasting it with the TIAC model. Possible implementations of TIAC are discussed and features of Linux relevant to a TIAC implementation are presented.

A. Time Interval Access Control (TIAC) Model

Authorization models using temporal constraints or temporal attributes have been proposed previously. Bertino et al. described a model [4], [5] that associated temporal constraints with access authorizations and models temporal dependencies among authorizations. The notion of associating temporal constraints with authorizations was extended in an access control model that supported discontinuous temporal constraints on authorizations [6]. Role-Based Access Control (RBAC) has also been extended to support temporal constraints to the activation and deactivation of roles [7]. Alturi and Gal [8], [9] proposed a model somewhat more closely related to TIAC: access control constraints are based on temporal attributes associated with the data as well the time of the data access request.

None of the authorization models mentioned above support policies based on temporal attributes associated with both subjects (e.g., a process representing the user) and objects (e.g., data). Seminal work [10], [11], [12] has modeled authorizations for subjects' to access to objects using an access matrix or tabular form. In particular, Graham and Denning [10] described how the access matrix was derived from the 3-tuple of subjects, objects, and a set of allowed modes of access. The TIAC model [1], [2] extends this approach by adding time as a decision variable.

Using interval algebra [13], the TIAC model associates

ISBN 9-9999-9999-9/99/ $20.00

temporal attributes with subject and object entities and describes access authorizations in terms of access graphs. The TIAC formal semantics is unambiguous and provides the ability to precisely decide when a subject with a given set of temporal attributes has permission to access an object, which is also endowed with temporal attributes. Since the model requires only three time intervals: those associated with subject and object, and the time interval during which access is requested, TIAC-based access policies can be checked for consistency using decidable algorithms [1].

To demonstrate the feasibility of constructing a fine-grained temporal access control system based on the TIAC model, a prototype was designed that utilized a combination of hardware and software, the Time Interval Memory Protection System (TIMPS) [14].

TIMPS employs the TIAC logic to control memory access at the page level.

The access control mechanism was logically divided into an *initial authorization phase*, an *ongoing access phase*, and a *termination phase*. For the ongoing phase, the use of hardware in combination with software offered some performance benefit; however, an implementation based entirely in software is both more flexible and practical.

To gauge the performance impact of TIAC, we first considered implementing TIMPS entirely in software. It was quickly realized that such an implementation is impractical due to the page-level granularity that would be imposed. In particular, if temporal access controls are based on pages and the higher-level controlled entities are not page aligned, then the access control mechanism would hinder system functionality.

The software implementation of the TIAC model described here controls access at the file-level, where *file* or *regular file* refers to a regular file in a mounted file system; it does not include pipes, sockets, devices, etc. Linux was chosen as the target system for our TIAC implementation.

B. Linux File Management

The Linux kernel implements a Virtual File System (VFS) [15][16] that allows different file systems to coexist and interoperate, and enables a homogeneous set of high-level file operations.

To allow additional management and control over file accesses, Linux Kernel Versions 2.6 and later implement Extended Attributes for most file systems. These extended attributes take the form of $< name, value >$ pairs, and are associated and stored permanently with files. These attributes provide a consistent means to extend file system capabilities while maintaining the independence of the underlying file system implementation. In this work, the extended security attributes supported in the Linux 2.6.15 kernel were used to add a single set of temporal attributes to files and directories.

In the TIFPS prototype, attributes are set by the administrator, i.e. the *root* user, and runtime logic propagates attributes in support of the access control policy. Files and directories lacking temporal attributes are treated as if temporal access is permitted at all times. A simple command-line utility facilitates administrative control over temporal attributes associated with files and directories. The utility also permits normal users to view temporal attributes, while additional options allow administrators to modify those attributes. To provide user-level TIFPS support, additional command-line utilities have been adapted to be TIFPS-aware.

III. Design and Implementation of TIFPS

Starting with requirements, this section describes the design and implementation of the Time Interval File Protection System (TIFPS) and noteworthy aspects of the implementation. We conclude with a short description of a number of Linux command line utilities that have been made TIFPS-aware.

A. Kernel Support for TIFPS

To ensure that the TIFPS implementation provided a coherent set of functions, a set of objectives was established.

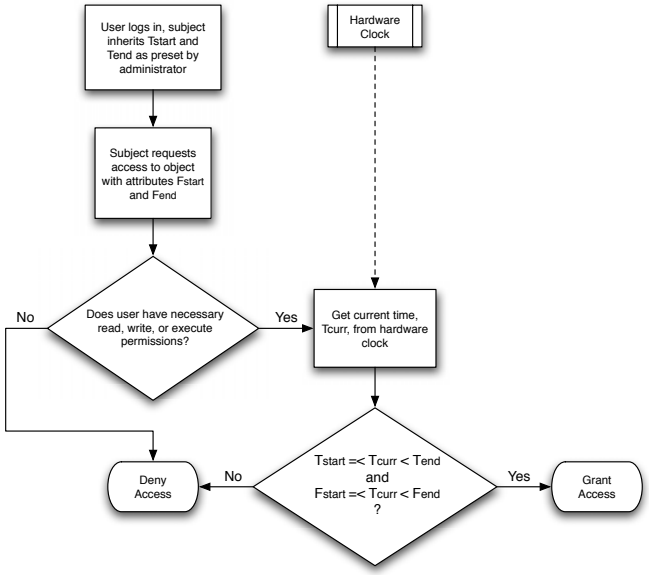

Fig. 1. High-level flow logic for TIFPS access control.

- Existing access control policies will be supplemented by temporal access controls such that access is granted only if all policy checks succeed.
- The prototype will not apply temporal access controls to objects that have not been assigned temporal attributes.
- For all objects with temporal access attributes, the kernel will mediate temporal access to those objects based on

those attributes. All forms of access will be mediated, viz. read, write, and execute.

- For the TIFPS prototype, only the administrator, i.e., the super user, may modify the temporal attributes associated with objects.
- When the time of access expires, access revocation should take place to at least one second precision.
- For operations that could result in the creation of copies of temporally controlled information, the destination files must take on the most restrictive temporal attributes of any files read by the copying process.
- The administrator will be able to set time-of-allowed access for subjects, i.e. user accounts, and objects, i.e. regular files and directories.

A high-level view of the logic for TIFPS access control decisions is illustrated in Figure 1. When a user logs into a TIFPS-enabled system, the login shell inherits the temporal attributes T_{start} and T_{end} specified in advance by the administrator. F_{start} and F_{end} define the time interval of allowed access for a particular object. For pre-existing files, the system administrator specifies in advance the temporal attributes F_{start} and F_{end} for those objects that require temporal access control.

Invocation of executables via the shell results in the process inheriting the temporal attributes of its parent. When a process (subject) attempts to access a file and after the standard Linux read, write, execute permissions are checked, the system checks the current time T_{curr} against the temporal attributes T_{start} and T_{end}. If current time is within the time interval specified by T_{start} and T_{end}, then the objects's time attributes are checked. If the current time falls within the time interval, F_{start} and F_{end}, specified for the object, then access is granted. Thus, access is granted in TIFPS only if the following is true:

$$T_{start} \leq t_{curr} < T_{end} \; and \; F_{start} \leq t_{curr} < F_{end}$$

To prevent unauthorized extension of access to information by copying, when read access to an object is requested in TIFPS, the process's temporal attributes are updated to take on the intersection of the temporal attributes of the object being read and the process's current temporal attributes. This is illustrated in Figure 2. After a program reads objects with temporal attributes $F_{1-start}, F_{1-end}$ and $F_{2-start}, F_{2-end}$, any write operation to new or existing objects will transfer the most restrictive time attributes associated with any of the objects read to the objects written.

To support temporal access control on subjects, as defined by the TIAC model [1][2], temporal attributes for the first process (subject) of a user session are initialized by copying the rights on the login directory to the subject. The administrator is authorized to grant and revoke time-based access to users by applying temporal attributes to home directories. When the user logs in, the process executing on his behalf inherits the attributes of the directory. By ensuring that each child process inherits the temporal attributes of its parent, temporal access control can be imposed on all subjects created during a user's session.

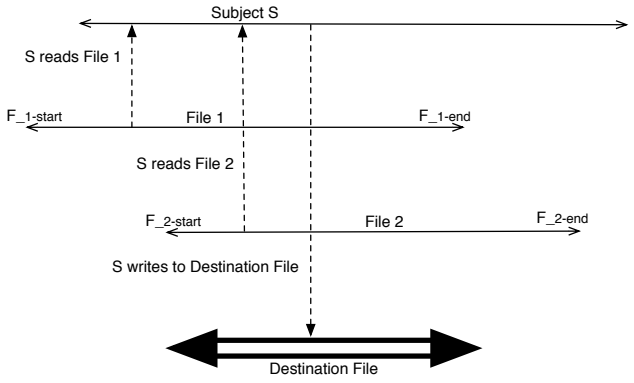

Fig. 2. TIFPS read and write policy.

A.1 Implementation considerations

In Linux, time is represented by a 4-byte signed integer, which specifies the number of seconds since the start of the Unix epoch. A negative integer represents the number of seconds before the Unix epoch. Since, for TIFPS, times prior to 1970 are irrelevant, the TIFPS extended attribute has a value range of 0x00000000 to 0x7FFFFFFF, i.e., from 1 January 1970 at 00:00:00 UTC to the year 2038, which we call T_{Unix0} and $T_{UnixINF}$, respectively.

Extended attributes are used for persistent storage of the TIFPS temporal attributes for objects and are stored as strings. The string representation of the *name* of the extended attribute for TIFPS is "security.tifps". The *value* of the extended attribute has the format ":0x00000000:0x7FFFFFFF\0", where the first hexadecimal number represents F_{start} and the second hexadecimal number represents F_{end}. Storing temporal attributes in this format simplifies string parsing during access control operations.

"Ext3", a popular journaling file system that is installed by default and supports extended attributes [17] was chosen for the TIFPS prototype. However, to the extent possible, the prototype was kept sufficiently generic to support other extended attribute file systems, such as "ext2" and "xfs".

To avoid implementing temporal access control either inside the kernel or with custom security hooks, an existing framework was needed. The Fedora Core 5 (FC5) distribution includes the Linux Security Module (LSM), a modular security framework that provides kernel-callable security hooks [18]. These generic security hooks can be used to implement different security policies.

Two other Linux security frameworks were considered. Rule Set Based Access Control (RSBAC) [19], unlike LSM,

does not require that the security hook functions be exported to user-space programs. Another is Grsecurity [20]. It is a multi-layered, detection, prevention, and containment framework for Linux security. Despite the enhanced security features of RSBAC and Grsecurity, LSM was chosen for the TIFPS implementation because of its stability and broad acceptance. Since TIFPS addresses only files and directories, a subset of the LSM security hooks was sufficient.

In addition to LSM, VMware Server 1.0.0 was used both to host a dedicated Subversion 1.3.0-4.2 [21] versioning server, and for target kernel development and testing. Fedora Core 5 - Kernel 2.6.15 [22], the target operating system, was reduced to the minimum number of kernel modules and drivers required to run the system. The kernel configuration file is available in the prototype source [3]. Source Insight 3.5 [23] and Emacs 21.4-14 supported kernel source code inspection and modification.

To use LSM, the *__init()* and *__exit()* functions had to be defined. The *security_operations* structure was used to implement custom security functions for each of the security hooks [3]. Since default security hooks have no effect, it was sufficient to implement only the security hook functions necessary to achieve the desired system behavior.

In the Linux kernel, *task_struct* contains metadata on processes and inodes contain metadata on files, directories, and other file system objects. The Linux Security Module predefines in each of these data structures a security object pointer to a security structure that is custom defined for the specific LSM implementation. In the TIFPS LSM implementation, the security structure defined for processes is named *tifps_task_security_struct* and has the following fields: a 4-byte back pointer to the *task_struct*, a semaphore data structure used for synchronization, and two signed integers representing T_{start} and T_{end} for allowed access by the process. The inode security structure is named *tifps_inode_security_struct* and has the following fields: 4-byte back pointer to the inode struct, a semaphore data structure, and two signed integers representing F_{start} and F_{end} for allowed access to the object represented by the inode structure [3].

A.2 Implementation details

On system initialization, with TIFPS LSM loaded, the kernel allocates a *tifps_task_security_struct* for the current running process, initializes the semaphore struct, and sets the TIFPS start and end times to T_{Unix0} and $T_{UnixINF}$, respectively. Subsequent tasks are also allocated a *tifps_task_security_struct*. Figure 3 depicts the low-level time policy enforcement logic.

Preliminary analysis shows that the restrictive policy with respect to time intervals introduces a problem. Consider an example: Assume that a user reads a file that expires 5 minutes after the user has logged into the system. After reading the file, then, due to the inheritance of temporal attributes, the process's time-of-allowed access also expires in 5 minutes. So, after 5 minutes, the task will not be allowed to access any other files.

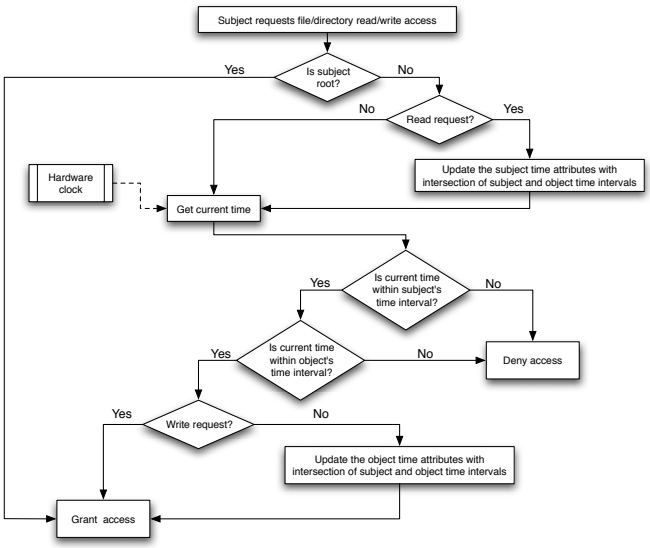

Fig. 3. Flow chart of low-level TIFPS enforcement logic.

One modification of the policy that was considered but not implemented is described here. Since the intent is to preserve the temporal attributes on information, the *tifps_task_security_struct* could be implemented to "keep track of" (as opposed to inherit) the most restrictive temporal attributes based on the intersection of previously accessed objects' attributes. Only during an attempt to write would the system enforce access control and transfer the temporal attribute with the most restrictive time interval to the objects being written. This work-around was not implemented because the file read operation implies a write operation to the kernel stack, and thus the same problem would persist.

For the purpose of the current prototype, the fork-and-exec paradigm of Unix-based operating systems obviates the problem. When a user logs into a Unix system, that user's login shell runs as a process. Execution of a program by the shell starts with a *fork()* invocation which clones the parent, thus creating the child process. An *exec()* within the child process replaces the executable of the child so that the intended new process runs. Hence, it is the child process that actually executes the command, reading from, and writing to files. Thus the temporal attributes of the parent login shell are not affected.

Also, an implementation-specific choice was also made with respect to attribute inheritance. When creating a new process via a *fork*, which clones the "parent" process, the *security_task_alloc()* security hook function is called from

the *copy_process()* kernel function. Since the attributes copied to the child are those that the parent had when the parent was created, this meant that the forked "child" inherited the temporal attributes of its "grandparent". To ensure that "child" processes inherited the temporal attributes of their "parents" at the time of child process creation, the parent attributes are used to determine the child process's temporal attributes.

To prevent increasingly restrictive access as directories dynamically inherit temporal attributes, the current TIFPS prototype sets the temporal attributes of directories upon their creation or through explicit administrative modification operations and does not dynamically update them. This prevents the most restrictive temporal attributes of any user accessing it to be applied to a shared directory such as */tmp*.

The TIFPS policy and permission check logic were implemented in the *tifps_enforcer()* function in the helper functions section of the *tifps_hooks.c* source code file. The file is divided into two sections, one implementing the security hook functions called by the kernel as part of LSM, and another implementing all the helper functions that the security hook functions call to provide temporal access control.

Although TIFPS was designed as a loadable module for the Linux Kernel, the kernel configuration utilities were modified to compile TIFPS as either a loadable module or as an *in situ* kernel module. Compatibility with other security modules such as NSA's SELinux [24] or BSD's Secure Level LSM has not been considered or tested.

B. TIFPS-Aware Command-Line Tools

To provide an interface to the temporal attributes associated with files and directories, a new tool *modtime*, was developed to meet the following objectives:

- Relative time will be with respect to current time, T_{curr}. (Note that internally the system enforces its temporal policy based upon absolute time, e.g., on November 5, 2007 at 1700 hours revoke access to parliament.txt)
- Temporal attributes will be set by specifying them in either absolute time or relative time.
- The administrative interface will be easy to use; it will not require complicated time calculations by the administrator.
- The tool will be able to take multiple arguments to change or display the temporal attributes of multiple files and directories at once.
- Usage instructions will be made readily available.
- The tool will display useful error messages to interactive users.
- The tool will allow the temporal attributes of files and directories to be easily viewed.

Fedora Core 5 as well as other Linux operating systems running Linux 2.6 and up include a set of user-space programs for setting and getting extended attributes: *setfattr()* and *getfattr()*, respectively. *Setfattr* can only be run by the administrator account, whereas *getfattr* can be run by any user. The *modtime* command was designed as a wrapper program that encompassed these. The *modtime* tool presents standard Linux command line tool syntax and semantics. A **man** page describing its usage was written [3].

The following existing command-line functions were modified to be TIFPS aware: *mkdir*, *rmdir*, *touch*, *chmod*, *ls*, *stat*, *file*, *find*, *rm*. Neither *mv* nor *ln* required modification to become TIFPS-aware, and testing of these two utilities showed that they were constrained by the underlying temporal controls. **Man** pages for the TIFPS-aware utilities were modified to reflect their new capabilities.

IV. Testing and Analysis

Test plans for validating TIFPS for correct functionality, measuring its performance overhead, and gauging its robustness in multi-user situations were developed.

A. Functional Tests

Functional testing was conducted to ensure that the access control mechanism of TIFPS LSM enforced the policies as expected. Both static and dynamic testing were performed. The static tests included experiments to observe:

- Enforcement of temporal policies for reading, writing, and executing files and directories, where "execution" has the standard Linux semantics,
- Inheritance of temporal attributes in file and directory creation operations and in file-copy operations,
- Possible corrupted file format information due to incomplete writes resulting from access revocation. (Note that directory writes are not a problem, as these are atomic with respect to the access checks.)

The static tests for explicit file and directory creation resulted in expected temporal attribute inheritance behavior.

A set of copy tests was devised to ensure that information copied from one object to another would have the most restrictive combination of attributes of the pair. Figure 4 illustrates three scenarios and the expected inherited time interval for the created file is shown. For each of the three scenarios, three ways to copy files in Linux were tested: the *cp* command, redirection, and pipes. The tests using pipes had unexpected results and will be discussed in IV-A.1.

Tests were created to examine the behavior of the system when access to a file is revoked during a write operation. The results showed that file corruption could occur if access is revoked while an application is writing state information to a file.

Dynamic attribute modification tests were designed to observe the behavior of the system when temporal at-

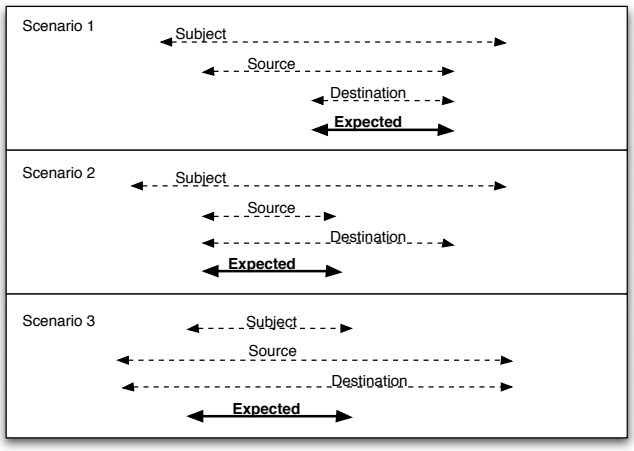

Fig. 4. File copy scenarios.

tributes are changed by an administrator during a user session. Two sets of scripts were developed: one to examine the effect of subject attribute modification and the other to determine the impact of object attribute modification. System behavior from the subject (i.e., user)'s perspective was recorded before and after the change by the administrator.

As expected, dynamic modification of the subject's temporal attributes did not affect the subject's continued access to files and directories; the subject inherits temporal attributes at the time of login, and modification of subject attributes does not take effect until the next login. In contrast, dynamic modification of object attributes is effective immediately and results in successful revocation of access upon expiration of its allowed temporal interval.

A.1 Analysis

Two unresolved problems were encountered during static testing. In addition, access revocation during file write presented a problem.

To ensure that the system consistently enforced the inheritance policy for copied information, multiple ways of copying files in a Linux system were tested. The system behaved as expected except when pipes were used to copy files. In this case *tee* read from standard input and sent the bytes read to two streams: standard output and a specified destination file. Since *tee* reads from the pipe and, in this implmentation, the pipe does not have temporal attributes (see II-A), the sequence of commands illustrated in Figure 5 successfully copies the contents of *source.txt* into *destination.txt* without preserving the temporal attributes of *source.txt*.

In Linux, inode data structures are associated with pipes. Thus, they can be assigned temporal attributes. This form of copying can be separated into the following individual

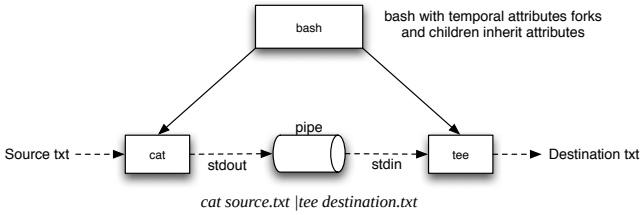

Fig. 5. Use of *tee* to copy files.

operations, where the parenthesized actions indicate anticipated TIFPS attribute inheritance:
1. *cat* reads from source.txt (*cat* inherits attributes from source.txt)
2. *cat* writes to the pipe (pipe inherits attributes from *cat*)
3. *tee* reads from the pipe (*tee* inherits attributes from *pipe*)
4. *tee* writes to destination.txt (destination.txt inherits attributes from *tee*)

Within our test environment, we determined that the sequence does not necessarily occur in the order given above. For example, on some occasions, the kernel scheduler was observed to schedule step 3 first, and the *tee* process will block until the *cat* process writes data to the pipe. Since the LSM security hook is called when the *tee* process requests read permission to the pipe and not after it wakes from blocking, the temporal attributes of the original file will not be correctly inherited. A solution to this problem will require further investigation.

The second problem was encountered when using the tab-completion feature of the *bash* shell. This feature allows a user to list all executables available in his/her path and when used by the login *bash* shell, all the executables in a user's path are read. Consequently the shell inherits the most restrictive temporal attributes of all the executables. Additional examination of attribute inheritance by executables is needed to solve this problem and is relegated to future work.

Finally, the TIFPS LSM does not provide transactional support for file writes. Hence, if file state information has not been written prior to access expiration, the file's state could be inconsistent. Again, further investigation is required to determine a solution. One solution is to support file system recovery in the kernel.

B. Performance Tests

Performance testing measured the additional overhead incurred by TIFPS LSM temporal access checks compared those for an unmodified kernel. The added overhead for TIFPS access control is approximately 5% for read operations, 20% for write operations, and 9% for copy operations.

To perform the tests, a set of scripts was created to time

Kernel	Read				Write				Copy			
	Single-file		Multi-file		Single-file		Multi-file		Single-file		Multi-file	
	Attr	No attr	Attr	No attr	Attr	No attr	Attr	No attr	Attr	No attr	Attr	No attr
Normal - ave	4.41	4.39	4.47	4.40	26.77	26.56	27.58	27.05	6.50	6.42	6.71	6.85
TIFPS - ave	4.65	4.59	4.72	4.65	32.28	31.91	32.59	32.20	7.09	7.09	7.25	7.40
Difference	5.44%	4.55%	5.51%	5.68%	20.6%	20.1%	18.16%	19.06%	9.13%	10.44%	8.05%	7.98%

TABLE I
TIFPS Performance Test Summary (units are in seconds)

the reading, writing, and copying of files on an unmodified 2.6.15 kernel and the same kernel loaded with the TIFPS LSM. The two kernels were guest operating systems on a machine running virtualized VMware server images of Fedora Core 5. The hardware running the VMware image has an Intel Pentium 4 processor running at 3.00 GHz. The RAM allocated for the image is 256M.

Two additional factors were considered: the intrinsic overhead associated with TIFPS-enabled entities; and performance variations between repeated actions on the same file and similar actions on different files. It was hypothesized that differences in data structure allocation and initialization might affect performance.

The results in Table I suggest that, contrary to hypothesis, the presence of TIFPS attributes did not significantly affect the performance. The reason for this result could be that most of the performance overhead of TIFPS occurs in the setup of the function calls to the TIFPS security hook implementations. In the TIFPS security hook implementations, access control logic is skipped in the absence of TIFPS attributes. It appears that skipping sections of code within a security hook function call did not significantly reduce performance overhead. Further experimentation is needed to localize the cause of performance overhead in TIFPS.

C. Concurrency Tests

Concurrency tests were performed to gauge the robustness of the TIFPS LSM in situations where multiple subjects with different temporal attributes request access to the same objects. Three user accounts were created, each with different temporal attributes. Four tests were performed.

1. When three subjects acting on behalf of their respective users attempted to continuously read the same file, it was found that the revocation time for read access for each subject corresponded to the time that the corresponding user's time attributes expired.
2. Three subjects attempted to continuously write to the same text file. When a subject's write access was revoked, the revocation time was recorded. In this case, the file correctly inherited the TIFPS permissions of the user whose temporal attributes are the most restrictive. At file expiration, the write access was properly revoked for all subjects.
3. When copying files to their home directories, each of the subjects' copies of the file inherited the temporal attributes associated with the individual user.
4. When subjects attempt to concurrently copy private files into a shared directory, it was found that each user's respective temporal attributes were preserved as expected. The shared directory retained its original temporal attributes as expected.

V. Discussion and Future Work

The benefits of TIFPS include kernel-level protection of the mechanism as well as consistent policy enforcement across all applications. TIFPS enforces proper inheritance of temporal attributes by subjects and objects for copy operations. This feature results in a tension between correct security behavior and the availability of system services.

To enforce proper inheritance in a temporal access control system, a policy similar to the High Watermark [12] should be implemented: a subject's level of access becomes increasingly restrictive as the subject accesses various objects. As a consequence, enforcement of this policy may result in the association of increasingly restrictive temporal attributes with a subject during the course of a session.

For a typical Linux shell, this problem is mitigated by the fork-and-exec paradigm. Since the child process incurs the increasing restrictions as various objects are accessed, the parent is unaffected. A new child process will inherit the original less restrictive attributes of the parent shell. However, the fork-and-exec paradigm introduces additional issues. For example, inheritance of temporal attributes was not consistently enforced in the copy operation performed using pipes to communicate information between sibling processes within the system. Therefore, future implementations should explicitly address the intended semantics of the file system to ensure that object and subject temporal attributes are preserved.

Our future work on temporal access controls includes enhancements to both the TIAC model and to the TIFPS implementation.

TIAC:

- Explore the implementation of TIAC in a distributed environment. This must include consideration of a reliable global clock.
- Extend the TIAC model to support a temporal attribute

inheritance policy that can be formally checked for consistency and reasonable semantics.

TIFPS:

- Examine the Linux fork-and-exec functionality to ensure proper policy enforcement when attributes are inherited by new processes. Related to this topic is the issue raised by the case of *bash* auto-completion for executables.
- Support times beyond the year 2038.
- Modify the implementation to include other file systems that support extended attributes, in addition to "ext3".
- Investigate write operations to ensure consistent file state upon temporal attribute expiration.
- A new form of temporal attribute could be created for TIFPS that could be used to modulate access periodically. For example, access to certain files might be permitted only between 8 AM and 5 PM.
- Support a higher degree of granularity for temporal attributes. For example read, write, and execute operations could each have separate temporal attributes.
- Consideration to the semantics of *write()* will be required since some file systems, e.g. AFS [25], use delayed writes and access could expire prior to a *close()*, which would force another write.

At the application level, the *modtime* tool could be extended to support recursive directory descent. In addition, further investigation of tools and APIs for use in TIAC-enabled systems will enhance utilization of temporal access control systems.

In summary, temporal access control systems can augment traditional access control mechanisms to support dynamic security services by changing access permissions based upon time. The capability of such a system to grant or revoke access in the future, as well to limit access to a specific time interval can provide another dimension for information control and sharing not available in traditional access control systems. The Linux-based TIFPS prototype implementation presented here can be used as a starting point for future temporal access control systems.

References

[1] F. Afinidad, T. E. Levin, C. E. Irvine, and T. D. Nguyen, "A model for temporal interval authorizations," in *Hawaii International Conference on System Sciences, Software Technology Track, Information Security Education and Foundational Research*, (Kauai, Hawaii), p. to appear, January 2006.

[2] F. Afinidad, *An Interval Algebra-Based Temporal Access Control Protection Architecture*. PhD thesis, Naval Postgraduate School, Monterey, California, June 2005.

[3] K. H. Chiang, "A prototype implementation of a time interval file protection system in linux," Master's thesis, Naval Postgraduate School, Monterey, California, September 2006.

[4] E. Bertino, C. Bettini, and P. Samarati, "A discretionary access control model with temporal authorizations," in *Proceedings of the 1994 Workshop on New Security Paradigms*, pp. 102–107, 1994.

[5] E. Bertino, C. Bettini, and P. Samarati, "A temporal authorization model," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pp. 126–135, 1994.

[6] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An access control model supporting periodicity constraints and temporal reasoning," in *ACM Transactions on Database Systems*, vol. 23, pp. 231–285, September 1998.

[7] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A temporal role-based access control model," in *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, pp. 21–30, July 2000.

[8] V. Atluri and A. Gal, "An authorization model for temporal and derived data: Securing information portals," *ACM Transactions on Information and System Security*, vol. 5, pp. 62–94, February 2002.

[9] A. Gal and V. Atluri, "An authorization model for temporal data," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pp. 144–153, November 2000.

[10] G. S. Graham and P. J. Denning, "Protection – principles and practice," in *Proceedings of the Spring Joint Computer Conference*, pp. 417–429, May 1972.

[11] B. W. Lampson, "Protection," in *Proc. 5th Princeton Conf. on Information Sciences and Systems*, (Princeton, NJ), pp. 437–443, 1971.

[12] C. Weissman, "Security controls in the ADEPT-50 time-sharing system," in *Proc. AFIPS 1969 Fall Joint Computer Conference, Vol 35*, pp. 119–133, AFIPS Press, Montvale, N.J., 1969.

[13] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[14] F. Afinidad, C. E. Irvine, T. D. Nguyen, and T. E. Levin, "A time interval memory protection system," Tech. Rep. NPS-CS-06-002, Naval Postgraduate School, Monterey, California, 2005.

[15] M. Bar, *Linux file systems*. New York, NY: McGraw-Hill, 2001.

[16] D. P. Bovet and M. Cesati, *Understanding the Linux kernel*. Sebastopol, CA: O'Reilly and Associates, Inc., January 2001.

[17] "The ext2/ext3 file system." http://e2fsprogs.sourceforge.net, 2006.

[18] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security modules: General security support for the linux kernel," in *Proceedings 11 USENIX Security Symposium*, (San Francisco, CA), August 2002.

[19] A. Ott, "RSBAC and LSM." http://www.rsbac.org/documentation/why_rsbac_does_not_use_lsm, April 2006.

[20] "Grsecurity." http://grsecurity.net/lsm.php, July 2006.

[21] "Subversion." http://subversion.tigris.org/, September 2006.

[22] Red Hat, Inc., "Fedora." http://fedora.redhat.com/, September 2006.

[23] "Sourceinsight." http://www.sourceinsight.com/, September 2006.

[24] P. Loscocco and S. Smalley, "Integrating flexible support for security policies into the linux operating system," tech. rep., National Security Agency, October 2001.

[25] J. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith, "Andrew: a distributed personal computing environment," *Comm. A. C. M.*, vol. 29, no. 3, pp. 184–201, 1986.

ISBN 9-9999-9999-9/99/ $20.00