NPS-CS-06-002

Trustworthy Commodity Computation and Communication

SecureCore Technical Report

A Time Interval Memory Protection System

Francis B. Afinidad, Cynthia E. Irvine, Thuy D. Nguyen,
and Timothy E. Levin

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 with support from DARPA ATO. 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 National Science Foundation or of DARPA ATO.

Author Affiliations

Naval Postgraduate School:

Francis B. Afinidad, Cynthia E. Irvine, Timothy E. Levin, Thuy D. Nguyen

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

TABLE OF CONTENTS

Abstract

Time is often a critical factor for making decisions regarding access to information. To manage and protect critical data in this regard implies that information systems need to enforce temporal security policies. However, we lack operating system and hardware support for implementing temporal protection mechanisms.

A time interval memory protection architecture to support enforcement of temporal policies is presented. It implements a Time Interval Access Control model in which authorizations are determined by time attributes associated with subjects and objects, and constraints on the relations of those attributes relative to the time of access. The policy is enforced at the memory page level by kernel mechanisms and related hardware extensions. The notion of memory access *phases* and the principle of *efficiently mediated access* are articulated to support the analysis of the design. A simulator-based prototype shows that the architecture is both feasible and practical.

I. Introduction

Time is often a critical factor for making decisions regarding authorization or access to information. Electronic commerce, multimedia content providers, financial institutions, digital libraries, and federal/military intelligence agencies all must manage and protect critical data that may be perishable, valid for a specific period, or where access is limited by time factors associated with both the data being accessed and the user or process accessing the data.

Credit cards, identification cards, X.509 certificates, phone cards, library cards, and other access tokens typically have expiration dates or a period of validity, which can be considered as time attributes that constrain the use of the access token. The tokens may be used to access data from sources such as: a) digital libraries, where temporal control over digital books is required, b) multimedia content providers, where there is a requirement to control viewing of multimedia content for a limited period, and c) financial institutions, where control over the time at which financial transactions are allowed to occur.

Temporal security policies may be based on other time attributes such as duration of allowed access, time during which data is valid, or time at which access to data is no longer authorized. Additionally, authorized access may be constrained by temporal factors such as the time during which a user is allowed to access data, the time during which the user has a valid authorization to access data, or a time at which a user in no longer authorized to access data. Thus, the time attribute associated with the user and that associated with the data are independent access decision variables. The Time Interval Access Control (TIAC) model [1] permits the specification of temporal policies where authorization is determined from four conditions—the mode of access, time attributes associated with subjects and objects, and constraints on the relation of those attributes with respect to the time of access. However, the low-level software and hardware protection mechanisms in current systems lack support for temporal attributes and there are no hardware mechanisms in modern CPU architectures to support management of accesses, e.g. revocation, based on temporal constraints. Thus, system-wide temporal

security policies would have to be enforced wholly by operating systems or applications. This adds complexity to policy enforcement, making it difficult to verify whether a protection system controls access as specified.

A. *Contribution of this work*

This paper presents a temporal protection architecture, the Time Interval Memory Protection System (TIMPS), to support enforcement of temporal policies in conjunction with the existing kernel and hardware access control mechanisms. It includes hardware support for automatic revocation of access to data after access permissions have expired, and hardware logic that is able to evaluate time interval access graphs. To provide a clear and simple conceptual framework for the temporal protection architecture, a new perspective to access control that views it with respect to time as a sequence of phases was developed. A functional simulation of our hardware protection architecture demonstrates its feasibility and practicality.

II. Background

We provide an overview of the Time Interval Access Control Model, followed by a review of existing hardware support for memory protection and some principles that guide the design of our architecture. The Principle of Efficient Mediated Access is introduced.

A. *Time Interval Access Control Model*

The Time Interval Access Control (TIAC) model [2, 3] provides formal semantics to express temporal security policies, in which time attributes of subjects and objects are used to determine authorized accesses. In this model, interval algebra [4] provides the necessary expressive power to precisely describe a desired temporal policy, and an efficient way to computationally reason about whether a given access request may be acceptable within the constraints of that policy. Temporal authorizations are represented by time attributes associated with both subjects and objects, and a "time interval access graph" that defines restrictions on the relations of those attributes with respect to the time of access.

In this model, a subject initiates a request to access an object in a given mode to occur at a particular time and the request is evaluated against the defined temporal policy.

A key advantage of the TIAC model is that the representation of the temporal policy is independent from time attributes of subjects and objects upon which specific access decisions are based.

B. *Protection mechanisms*

From the mid-1950's through the early 1960's, computer hardware protection mechanisms were developed to help improve the reliability, availability, and correct operation of time-sharing systems [16, 23, 37, 6, 17, 36, 22, 48]. By the mid-1960's, the risk of accidental or deliberate disclosure of private user data or corporate sensitive data was a concern [9, 18, 53, 27, 54, 55, 29, 56, 51]. Hardware memory protection mechanisms and memory protection schemes were developed [28, 57, 56, 8, 47, 52].

Recent memory protection systems have explored: (1) memory protection schemes to enforce access permissions at either the granularity of individual words [58, 59] or by creating multiple domains of protection within a task [10], (2) memory protection support for single address space operating systems that use 64-bit address spaces [14, 38, 15], (3) memory protection support for fast capability systems [24, 49], (4) multi-protection page tables [50], and (5) tamper-resistant execute-only memory [41, 40]. However, none of these mechanisms include support for temporal policies. Basic hardware memory protection schemes are reviewed next.

Page and Segment-Level Memory Protection

Paged-based virtual memory protection was introduced in the Atlas computer in the late 1950's through the early 1960's [23, 37, 36, 39]. Virtual memory was partitioned into small memory blocks called "pages". Each page in memory was associated with a page address register [23, 37].

Paged-based virtual memory protection is commonly enhanced to provide hardware-based access control. Instead of a single lockout bit associated with each page, additional permission bits have been added for more granular access control (e.g., read-only, write, read/write, read/execute, execute only). The metadata for a page is global to the system and must be managed by the software to provide per-process permissions.

Segmentation was first introduced in the Burroughs 5000 [12]. A per process segment descriptor is used to check memory accesses against memory bounds and access permissions. A flag bit is used to distinguish code and data so that hardware would not inadvertently execute data or alter code. The Intel IA-32 [31] provides protection mechanisms that can operate at both the segment level and page level, while the PowerPC [46] and HP PA-RISC 2.0 [33] both use segmentation as a virtual memory scheme combined with page-level protection.

Other hardware improvements include: 1) the ability to group pages in memory into separate address protection domains [33, 30, 32], 2) the use of privilege levels as an additional attribute for determining access permissions [47, 33, 31, 30, 32], and 3) the use of a protection look-aside buffer (PLB) [38] to accelerate permission checks by caching permissions in hardware and by separating access checks from address translation.

Capabilities

A capability can be thought of as a token containing access permissions to a unique object in a computer system. [20] To ensure the integrity of each capability, capabilities must be protected, e.g, by using a tagged approach or a partition approach.[21]

Capability systems allow processes to propagate or exchange capabilities with other processes. Once a capability is granted to a process, it can be difficult to revoke that capability while ensuring that the revocation process does not affect or interfere with other processes [34, 35, 13].

Principles for Security Mechanism Implementation

There are three essential design requirements for the reference validation mechanism based upon the ideal of the Reference Monitor Concept [5]: 1) It must be tamperproof,

meaning that the reference validation mechanism cannot be altered, 2) It must always be invoked, meaning all access to system resources must be mediated, and 3) It must be small enough to be subject to verification, meaning that it is understandable and to operate strictly as specified. The goal of the TIMPS architecture is to design a memory protection system such that all design requirements for the validation mechanism can be adhered to as closely as possible.

In their survey, Saltzer and Schroeder [46] described design principles applicable to the implementation of protection mechanisms. Three are applicable to the design of the TIMPS mechanism: the principle of complete mediation, the principle of least privilege, and the principle of fail-safe defaults. Although various hardware security mechanisms have been recommended [6, 17, 48, 9, 18, 19, 25, 43, 7, 26, 44, 45, 52, 42, 60], there has been no articulation of a principle to guide the choice between a hardware or software security implementation for a mechanism. To guide our reasoning, we introduce the *Principle of Efficient Mediated Access*.

III. Principle of Efficient Mediated Access

Inherent in the development of secure systems is the complexity of understanding the interaction between software and hardware security mechanisms and how these security mechanisms are combined to enforce the desired security policies. To address controlled access to protected resources, three major components have to be considered: 1) the security policy that specifies access constraints, 2) the software enforcement mechanisms, and 3) the hardware enforcement mechanisms. Clearly, the different mechanisms must work in concert to enforce the policy.

Which parts of the enforcement mechanism should be allocated to hardware and which should be performed by software?

If hardware provided no support to the overall access mediation mechanism, then every access to a physical resource would require software intervention. This means that the system would provide a form of emulation for even the simplest primitive instructions, such as *move-byte*. This is highly inefficient, but affords considerable design flexibility, due to the greater malleability of software. Alternatively, for a special-purpose system, the entire mechanism might be hardwired. An example is an in-line encryptor. This is highly efficient, but inflexible, since any modification will require new hardware.

To achieve a balance between flexibility and efficiency, the *principle of efficient mediated access* [1] is introduced. This principle states that: *the access control mechanism should be allocated to the lowest possible level of the system while still meeting system flexibility requirements*.

If a system is constructed of hierarchical layers, with hardware constituting the lowest layer, then, when possible, the most efficient choice is to allocate the access mediation mechanism to the hardware. Hence, the hardware must have the *sufficient* access mediation mechanisms to enforce access to hardware resources for a broad range of policies and architectures. This principle guided the construction of the TIMPS mechanism.

IV. Phases of Memory Access

To support enforcement of a temporal policy, a memory protection architecture must mediate all memory accesses and incorporate time attributes in its decision logic. We define three different *phases* of memory access, based on the different functions of the protection architecture. This insight is key to understanding the coordinated interaction between the kernel and hardware when controlling memory accesses and helps justify the chosen delineation between software and hardware protection mechanisms.

The *Initial Access Phase* is when a subject requests and is granted access to an object for the first time, or for the first time since the last "termination phase" (see below). This can be seen as a request to the kernel to instantiate an object into a subject's virtual address domain, such as a "file open" operation. Object access is considered 'established' when a subject has the mechanical means to access an object, such as a valid memory descriptor.

Once initial access is established, a subject will be allowed the specified mode of interaction with an object beginning at the end of the initial access phase and ending at the time that access to the object is no longer allowed (expiration time). This time interval is the duration of the *Allowed Access Phase* and is characterized by the subject's invocation of operations such as "move-byte."

The *Access Termination Phase* starts when access to an object expires or a request to terminate access to an object has been initiated. This phase ends when the system has removed or invalidated the memory instance of the object in the subject's virtual address domain, or has otherwise performed all system-dependent actions related to termination of a subject's ability to interact with the object.

In each of these phases, there is a need for enhancements to the existing memory protection mechanisms to facilitate enforcement of temporal aspects of the security policy.

V. Time Interval Memory Protection System

TIMPS supports mediation of all memory accesses during all phases. In the following discussion, the term *kernel* refers to the low-level software responsible for resource allocation and the software portion of protection mechanisms.

A. *Temporal Authorization Matrix*

TIMPS defines a kernel-based *temporal authorization matrix* that is indexed by subject and object, in which each entry consists of a mode (or set of modes) and a *phi field*, which is a representation of the TIAC model's time interval access graph. The function to compute whether an initial-phase access request is authorized can be implemented in software or hardware. If implemented in hardware as part of the memory-page translation mechanism, the objects would be memory pages; if performed by the kernel, the objects would be kernel abstractions such as memory blocks.

The phi field consists of three subfields *to*, *ts*, and *so*, which specify the allowed relations between the subject time attribute, the object time attribute, and the requested time of access. Table 1 describes each subfield and Figure 1 shows its layout.

Phi Field	Name	Description
to	*time_to_object interval relation*	*This field is a 16-bit field that represents the allowed time-interval relation(s) between the object and current time of access. The default value is 0.*
ts	*time_to_subject interval relation*	*This field is a 16-bit field that represents the allowed time-interval relation(s) between the subject and current time of access. The default value is 0.*
so	*subject_to_object interval relation*	*This field is a 16-bit field that represents the allowed time-interval relation(s) between the subject and object. The default value is 0.*

Table 1. Contents of the phi field

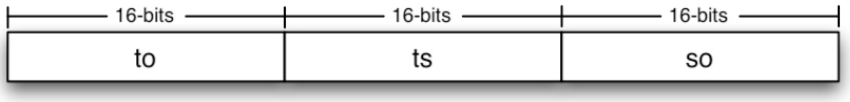


Figure 1. Layout of the phi field

Encoding Scheme for the Phi Subfields. The encoding scheme for *to*, *ts*, and *so* are identical. Each 16-bit subfield is a bitwise representation of the disjunction of possible interval algebra relations between two intervals (x and y in Table 2).

For example, consider the following temporal constraints, where τ represents a time interval, and $now.\tau$, $o.\tau$, and $s.\tau$ are symbolic representations of the time, object and subject intervals (respectively) that the constraints will be applied to:

to: DURING($now.\tau$, $o.\tau$) ∨ STARTS($now.\tau$, $o.\tau$) ∨ FINISHES($now.\tau$, $o.\tau$)

ts: DURING($now.\tau$, $s.\tau$) ∨ FINISHES($now.\tau$, $s.\tau$)

so: OVERLAPS($s.\tau$, $o.\tau$) ∨ FINISHED_BY($s.\tau$, $o.\tau$) ∨ INCLUDES($s.\tau$, $o.\tau$)

Using Figure 2 and Table 2 the bitwise encoding for *to, ts and so* are as follows: (Note: bit positions 13-15 are reserved in this example.)

to = 0000010000010100

ts = 0000010000000100

so = 0000001100000010

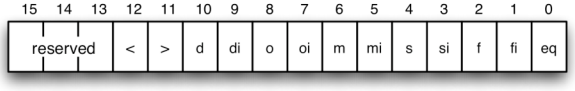
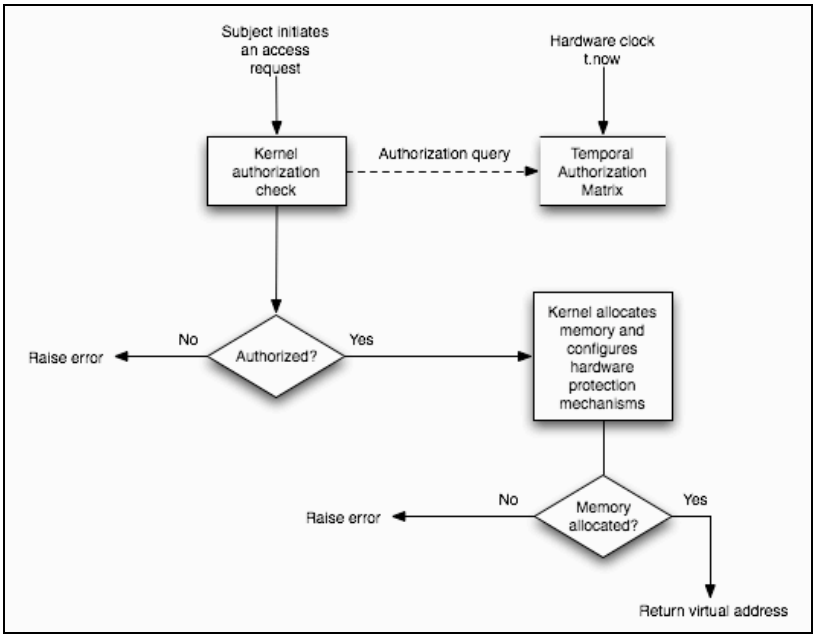


Figure 2. Bitwise encoding for to, ts, and so subfields

Bit Pos	Symbol	Definition	Bit Pos	Symbol	Definition
0	=	EQUALS(x,y)	8	o	OVERLAPS(x,y)
1	fi	FINISHED_BY(x,y)	9	di	INCLUDES(x,y)
2	f	FINISHES(x,y)	10	d	DURING(x,y)
3	si	STARTED_BY(x,y)	11	>	AFTER(x,y)
4	s	STARTS(x,y)	12	<	BEFORE(x,y)
5	mi	MET_BY(x,y)	13	--	reserved
6	m	MEETS(x,y)	14	--	reserved
7	oi	OVERLAPPED_BY(x,y)	15	--	reserved

Table 2. Bit position mapping for time interval relations

Using Figure 1, the *phi* field becomes 00000100000101000000010000000100000000110000001 0, which during the authentication check can be easily compared to a similar encoding of the relations between the actual subject, object, and time intervals.



B. *Access Control During the Initial Access Phase*

During the Initial Access Phase, (1) a subject submits an *access request* to the kernel, (2) the kernel processes the request by consulting the authorization matrix regarding the requested mode of access and current time, (3) if authorized, the kernel configures the hardware to add the object to the subject's address space for the requested mode, and (4)

returns a handle or other information for the subject to begin to access the memory location requested. Figure 3 illustrates the steps in the initial access phase.

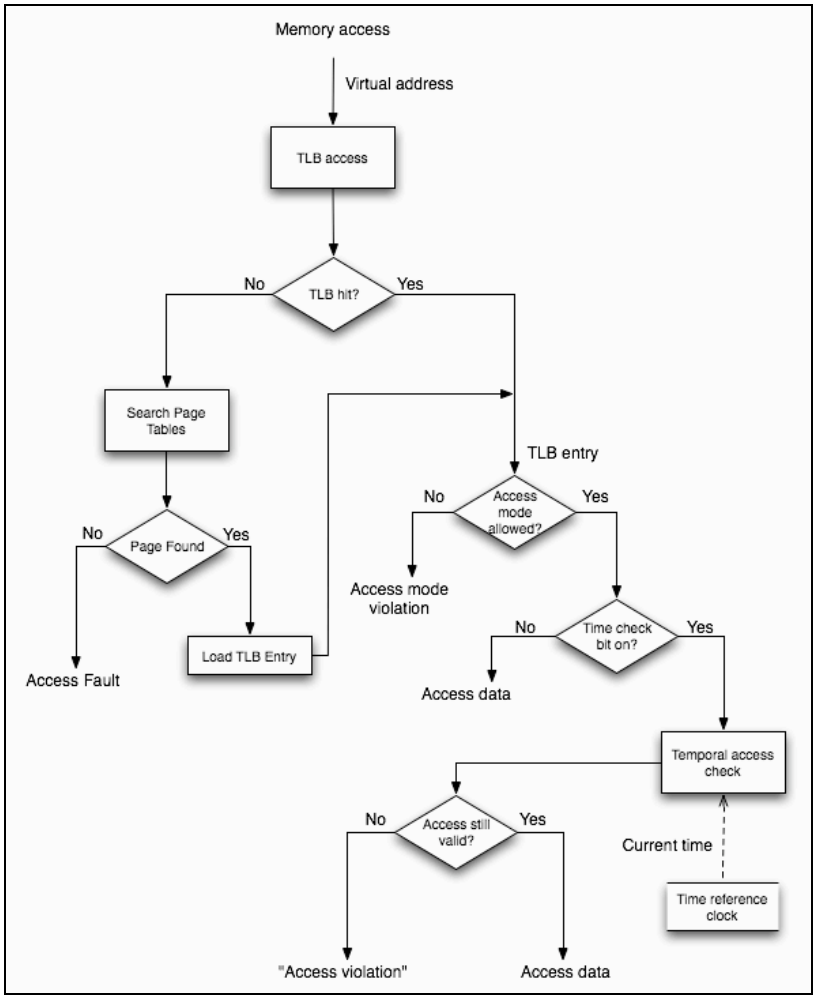


Figure 3. Logical flow diagram for mediating memory access during the Allowed Access Phase

C. *Access Control During the Allowed Access Phase*

During this phase, the hardware enforces the temporal access constraints on every interaction with the object. These are called "access checks" to distinguish them from the authorization check of the initial access phase. The TIMPS architecture regulates access to memory via the virtual memory address translation mechanism.

A subject may try to use an invalid or unauthorized virtual address, or an unauthorized mode of access. In these cases, the hardware returns a memory access violation. If the operation is allowed, then a hardware time_check flag associated with the virtual memory address is checked. If the time_check flag is set (true) then a *temporal access check* is performed, otherwise, access to memory is permitted.

In the temporal access check, the hardware ensures that the expiration time for the authorization has not passed. The operation faults if the subject is no longer authorized. Figure 4 shows a logical flow diagram of the Allowed Access Phase.

D. Access Control During the Access Termination Phase

The Access Termination Phase begins when access to a previously permitted virtual memory address location is no longer allowed by the policy, and continues until the virtual memory address is invalidated, or the hardware representation of the temporal policy is changed to once again allow access. The key point is that the addressing mechanism may allow access during this time, so the temporal mechanism must assure denial.

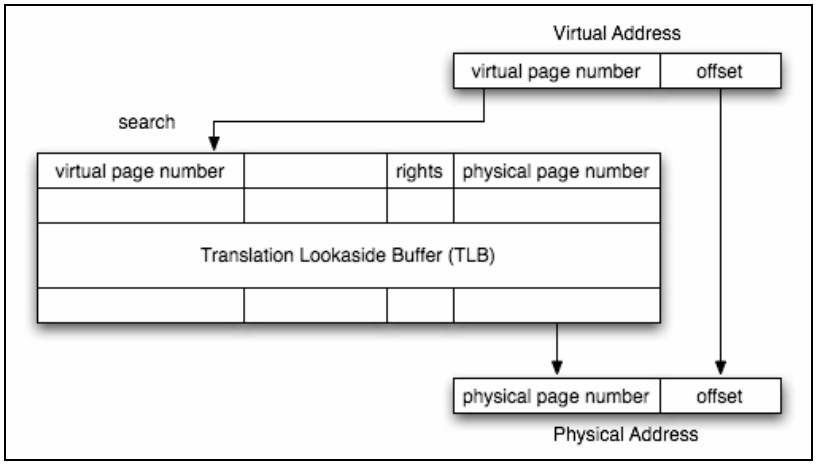


Figure 4. Virtual memory translation/protection scheme without temporal protection mechanisms1

VI. TIMPS Architecture

The TIMPS architecture provides the necessary and sufficient mechanisms for supporting the temporal access control concepts discussed above. Modern CPUs perform memory protection checks during the virtual memory translation process as illustrated in Figure 5. The rights field stores the access mode currently allowed for a particular memory address location, such as read, write, and execute access. Other fields typically used (not shown) in memory protection are those that specify privilege levels and "region" level protection rights. This approach will be used to illustrate the TIMPS architectural extensions.

[1] Adapted from Figure 4-2 in Intel Corporation, "IA-64 System Architecture," in *Intel IA-64 Architecture Software Developer's Manual*, vol. 2, p. 4-3: Intel Corporation, 2000

		Virtual Memory Scheme 1	Virtual Memory Scheme 2	Virtual Memory Scheme 3
Initial access phase	**Allocation of authorization check**	Kernel routine with respect to kernel objects	Specialized TIAC hardware module with respect to pages	Specialized TIAC hardware module with respect to kernel objects
	Allocation of Expiration time algorithm	Kernel routine	n/a	Either 1) kernel routine or 2) TIAC hardware logic with expiration module
Allowed access phase	**Type of access-check mechanism**	Simple TLB/page table extensions	More extensive TLB/page table extensions and TIAC hardware module	Simple TLB/page table extensions

Table 3. Summary of differences in the virtual memory protection schemes

During the memory address translation process, the TIMPS temporal access check is invoked just after the access mode permission check (see Figure 4). In this section, three virtual memory protection schemes for TIMPS are presented. Details of the specific hardware extensions and software components required to support temporal access control are discussed elsewhere [1]. Table 3 summarizes the major differences of the three schemes.

Hardware Reference Clock

TIMPS requires a hardware reference clock that can be used to determine the time of access during an access check. The clock must be able to provide absolute or relative time in the same units used for the temporal authorizations, and must be able to represent time in two ways: 1) a unit interval and 2) a point.

For example, if the clock has a 1 second resolution and the current time (t_{now}) is 4:10:00, then in absolute time:

- unit time interval = [t_{now}, t_{now} + 1) = [4:10:00, 4:10:01)
- point in time = t_{now} = 4:10:00

If the temporal authorizations are based on relative time, the output of the reference clock, for the above example where 4:10 is the time, would be:

- unit time interval = [t_{now}, t_{now} + 1) = [0, 1)
- point in time = t_{now} = 0

In the schemes discussed below, the kernel uses the unit time interval format [t_{now}, t_{now} + 1), shorthand notation for which is [*tnow*-, *tnow*+).

Virtual Memory Protection Scheme 1

Virtual memory protection scheme 1 is intended to have a minimal amount of hardware support. In this scheme, the TIMPS architecture extends traditional TLB/Page-Table entry structures by adding two protection fields: 1) *ex* (expiration time), and 2) *tc* (time_check flag). See Table 4. The *ex* and *tc* fields are stored in the Page-Table entry structure so that temporal protection information can be preserved when the TLB cache is flushed or when a TLB entry is invalidated. The bit-length of the *ex* field is the same format and length used by the time reference clock. Additional CPU instructions are needed to allow the kernel to configure the *ex* and *tc* fields when an access request is granted during the Initial Access phase. These instructions must be privileged and executable only by the kernel to ensure that the fields are not modified or bypassed by less privileged subjects.

During the authorization check, the kernel computes the expiration time based on the constraints of the temporal authorization matrix. The expiration time represents the time at which access is no longer allowed and is stored by the kernel in the *ex* field of the appropriate TLB/Page-Table entry. Details regarding expiration time computation are described in the TIAC model [3].

TLB/Page-Table Entry Field	Name	Description
ex	*expires*	The time value (unsigned) used to determine whether access has expired. This field is only used if the *tc* bit is set to 1. Its default value is 0.
tc	*time_check flag*	A 1-bit field that indicates whether temporal access checks are enabled for a specific TLB entry. The default value for this field is 1.

Table 4. TLB/page-table protection fields for virtual memory scheme 1

The kernel sets the *tc* field of the appropriate TLB/Page-Table entry when allocating memory during the Initial Access phase. The *tc* field can serve either of two purposes: 1) to indicate whether a temporal access check is required for a particular page in memory or 2) to indicate that the time interval during which access is allowed is infinite. The *tc* field is set to '1' to enable temporal access checks for a particular TLB/Page-Table entry. Where access to a memory location has no temporal constraints or access is allowed for an infinite amount of time, the *tc* field is set to '0'.

Once the kernel has allocated memory and configured the *ex* and *tc* fields, the hardware controls access during the Allowed Access Phase (see Figure 6). If a temporal access check is required, the current value of the time reference clock (current time of access) and the expiration field *ex* associated with the memory page being accessed are compared to determine whether access is allowed:

if $(t_{now} < t_{expires}) = true$ *then*
    access is *allowed*
else
    access is *revoked*
where :

t_{now} is the current time of access (derived from the reference clock)
$t_{expires}$ is the expiration time stored in the *ex* field

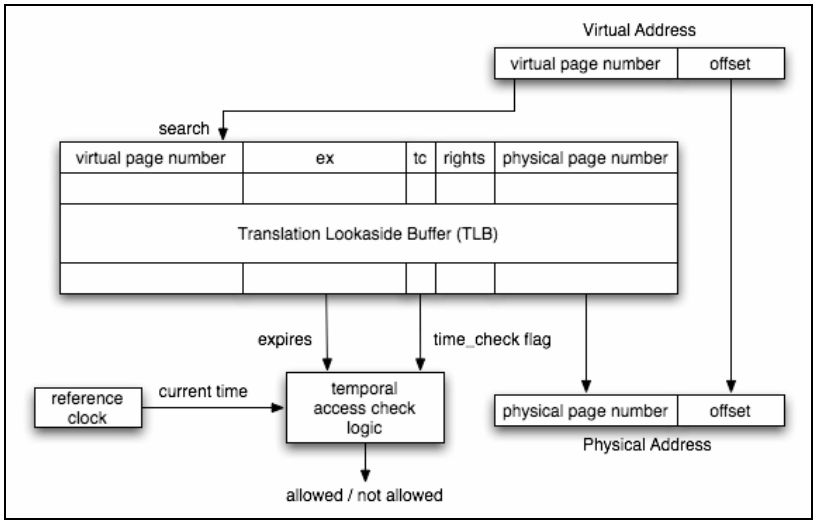


Figure 6. Conceptual virtual memory protection scheme 1

When the current value of the reference clock is greater than or equal to the value stored in the *ex* field, the hardware raises a fault to the kernel to indicate an access violation. As long as the *ex* field is not modified, any future attempts to access that particular memory location will be denied. The kernel is responsible for invalidating the TLB entry where the violation had occurred and performing the appropriate steps to modify or delete the corresponding page table entries as necessary.

TLB/Page-Table Entry Field	Name	Description
t-	*t-minus*	This field is the closed end-point of the half-open interval [*t-*, *t+*) used to represent the time attribute associated with the TLB/Page-Table entry. The bit-length of this field is implementation-dependent but must be the same bit-length as *t+* and the bit-length used by the time reference clock. Its default value is 0
t+	*t-plus*	This field is the open end-point of the half-open interval [*t-*, *t+*) used to represent the time attribute associated with the TLB/Page-Table entry. The bit-length of this field is implementation-dependent but must be the same bit-length as *t-* and the bit-length used by the time reference clock. Its default value is 0.
Phi	*phi*	This field is used to store the encoded access graph that represents the temporal access constraints between the current process's time attribute [*pt-*, *pt+*), the TLB/Page-Table entry's time attribute [*t-*, *t+*), and time of access. This field is 48-bits in length and contains three subfields *to*, *ts*, and *so* that are 16-bits each. These fields are described in detail later. Default value is 0

Tc	*time_check flag*	This is a 1-bit field indicates whether temporal access checks are enabled for a TLB entry. When this bit field is set to 1, temporal access checks for the TLB entry are enabled. When this bit field is set to 0, no temporal access checks are required for this TLB entry. This could mean that no temporal constraints apply to this memory location or that access to this memory location is allowed for an infinite duration. Its default value is 1.

Table 5. TLB/page-table protection fields for virtual memory scheme 2

Virtual Memory Protection Scheme 2

In virtual memory protection scheme 2 a TIAC hardware module relieves kernel software of the burden of computing the temporal portion of the authorization check and the expiration time during the Initial Access phase. In this scheme, the traditional TLB/Page-Table entry structures are extended with fields to represent the page's temporal attributes as well as the related authorizations (See Table 5 and the phi field in Section 5.1). The hardware context of the process is extended to include its temporal attributes (p- and p+). The temporal access check is performed by hardware during address translation, e.g., implemented as part of the arithmetic logic unit or as a separate hardware chip.

This approach was found to be inefficient during analysis due to the larger hardware table requirements, and will not be discussed in detail.

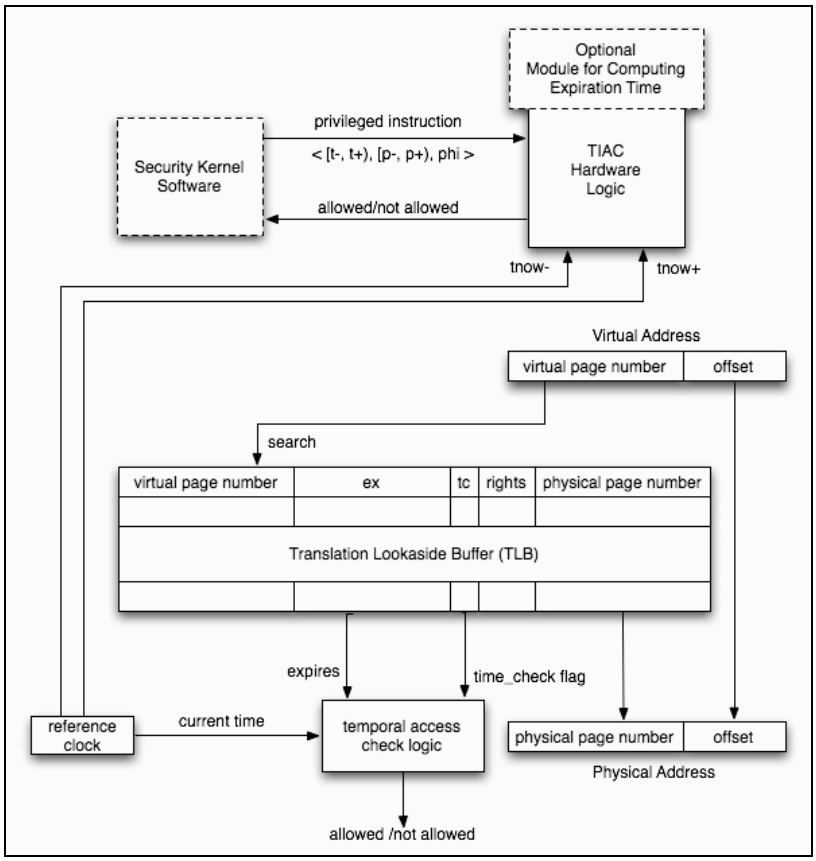


Figure 7. Virtual memory protection scheme 3

Virtual Memory Protection Scheme 3

Virtual memory protection scheme 3 is a hybrid of schemes 1 and 2. As depicted in Figure 7, the TLB/Page-Table entry structure and access check performed during the allowed access phase are the same as in scheme 1.

The Initial Access phase is similar to scheme 2. If the kernel determines that the temporal authorization matrix allows the requested mode, it uses the TIAC hardware module to check the request against the temporal constraints. If the request is authorized, the kernel computes the expiration time, using the TIAC hardware module if the expiration function is included there, and configures the *ex* and *tc* protection fields for the virtual memory object.

VII. Comparison of TIMPS Virtual Memory Protection Schemes

Each virtual memory protection scheme described earlier uses a different combination of temporal protection mechanisms to control access during each of its access control phases. Each scheme supports three main functions: 1) evaluation of the time interval access graph φ, 2) computation of the expiration time, and 3) hardware mediation of every access to virtual memory. A comparative analysis of performance and memory usage was conducted for the three TIMPS virtual memory protection schemes [1].

As shown in Table 6, scheme 2 is the most efficient mechanism during the Initial Access Phase when compared to scheme 1 and scheme 3 with the TIAC expiration module. However, the bulk of memory accesses in most usage scenarios are during the Allowed Access Phase, so the performance advantage this provides is negligible. In Table 7, it is shown that schemes 1 and 3 perform better than scheme 2 during the Allowed Access Phase. It is important to note that performance (i.e., latency) during the Allowed Access Phase weighs heavily in determining the efficiency of mediated access. Another drawback for using Scheme 2 is that it requires more space in the TLB (see Table 8).

Temporal Protection Functions	**Virtual Memory Protection Scheme**			
	Scheme 1	**Scheme 2**	**Scheme 3**	
Evaluation of the time interval access graph φ	*access_allowed()* (*1750 ns*)	*TIAC hardware logic* (*20.820 ns*)	*TIAC hardware logic* (*20.82 ns*)	
Computation of expiration time	*compute_expires()* (*1000 ns*)	Not required	*compute_expires()* (*1000 ns*)	*TIAC hardware expiration module* (*36.926 ns*)
Total Delay	***2750 ns***	***20.820 ns***	***1020.82 ns***	***57.746 ns***

Table 6. Comparison of the total latency (delay) of the temporal protection mechanisms used during the Initial Access Phase

Temporal Protection Function	Virtual Memory Protection Scheme		
	Scheme 1	**Scheme 2**	**Scheme 3**
Evaluation of every memory access during Allowed Access Phase and Access Termination Phase	*Temporal Access Check Hardware Logic*	*TIAC Hardware Logic*	*Temporal Access Check Hardware Logic*
Total Delay	***10.215 ns***	***20.820 ns***	***10.215 ns***

Table 7. Comparison of the total latency (delay) of the temporal protection mechanisms used during the Allowed Access Phase and Access Termination Phase

TLB/Page-Table Temporal Protection Fields	Virtual Memory Protection Scheme		
	Scheme 1	**Scheme 2**	**Scheme 3**
ex	32 bits	n/a	32 bits
tc	1 bit	1 bit	1 bit
t-	n/a	32 bits	n/a
t+	n/a	32 bits	n/a
phi	n/a	48 bits	n/a
Total Number of Bits	***33 bits***	***113 bits***	***33 bits***

Table 8. Comparison of the number of additional bits required in a TLB/Page Table Entry for each virtual memory protection scheme

Virtual memory protection scheme 1 is recommended for systems where there will be relatively few authorization checks versus access checks (e.g., personal desktop computer, laptop, or PDA). Virtual memory protection scheme 3 (with TIAC expiration module) is recommended for systems where a high number of initial access requests (e.g., web-server or database-server) is expected.

VIII. Sim-tiac Simulation Results

The SimpleScalar framework [11] is a software tool that simulates a hardware processing environment through the use of various emulators. A new simulator called *sim-tiac* was developed for SimpleScalar to demonstrate the concept of time interval access control. Sim-tiac implements the TIMPS virtual memory protection scheme 1, and provides an environment for testing various supporting mechanisms.

The page-table entry structure defined in SimpleScalar's original baseline memory module (memory.h) was extended to include the following temporal protection fields: 1) *ex* (expires) field and 2) *rights* field that includes *tc* (time check), *r* (read), *w* (write), and *x* (execute) bits. To ensure that all virtual memory accesses are mediated in sim-tiac's simulation environment, a new direct memory access interface for use by the host operating system was developed as the *tiac_mem_access()* function, and other direct memory access functions were disabled.

A series of tests was conducted to verify the TIMPS architecture concept and the operation of sim-tiac. For each test, a user program accessed memory locations within its simulated virtual memory address space. The tests exercised the protection mechanisms of each memory access phase (i.e. Initial Access Phase, Allowed Access Phase, and Access Termination Phase).

The test results showed that the TIMPS architecture concept can efficiently and effectively enforce temporal security policies. We conclude that hardware protection mechanisms can support a feasible solution to temporal access control.

IX. Future work and summary

Several areas related to TIMPS are still being investigated. In the context of the TIAC mode, work is needed to determine how to maintain temporal authorizations, e.g., creation and deletion of subjects, objects, and temporal authorization entries, as well as the implication of tranquility of temporal attributes associated with subjects and objects. The semantics of the TIAC model can be generalized so that it could specify an access request that uses a different reference time other than current time, which would allow the model to check for previous, current, and future authorizations. The model can also be extended to support the specification of event-based security policies and tools can be developed for creating and checking temporal security policies expressed using the TIAC model semantics.

Although a simulation of TIMPS was developed, it would be useful to prototype the CPU extensions in hardware. This could include development of efficient designs for accessing the time reference clock for every temporal access check and for minimizing the overhead of temporal access checks. One approach to the latter would be to perform them concurrently with instruction execution, provided the result in an exception before the completion of the memory cycle. This concept was demonstrated in the Mondrian Memory Protection system [59]. Use of a Protection Lookaside Buffer [PLB] [38] might also be considered.

TIMPS opens up many possibilities in the development of hardware and kernel software for support temporal protection mechanisms. At the operating system level, it remains to be determined how current file systems could be extended to support association of time attributes (protection labels) to files.

A. *Summary*

The Time Interval Memory Protection System (TIMPS) architectural concept was developed to support enforcement of temporal security policies specified using the TIAC model. TIMPS protection mechanisms support temporal access control at the kernel

software and hardware level to provide efficient mediated access. The TIMPS approach to analyzing temporal protection mechanisms for each memory access phase supports a clear articulation of the interaction between the kernel and hardware temporal protection mechanisms to ensure complete mediation.

References

[1] F. B. Afinidad, An Interval Algebra-Based Temporal Access Control Protection Architecture, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, June 2000.

[2] F. B. Afinidad, T. E. Levin, T., C. E. Irvine, and T. D. Nguyen, "Foundataion for a Time Interval Access Control Model", *Mathematical Methods, Models, and Architectures for Computer Networks Security*, MMM-ACNS 2005, St. Petersburg, Russia, Septebmer 24-28, 2005, Proceedings, Lecture Notes in Computer Science, ed. V. Gorodetsky, I. Kotenko, and V. Skormin, Springer-Verlag GmbH, Vol. 2685, pp 406-411, St. Petersburg, Russia, Septebmer 24-28, 2005.

[3] F.B. Afinidad, T. E. Levin, C. E. Irvine, and T. D. Nguyen, A Model for Temporal Interval Authorizations, *Hawaii International Conference on System Sciences, Software Technology Track, Information Security Education and Foundational Research*, Kauai, Hawaii, January 2006, to appear.

[4] J. F. Allen, "Maintaining Knowledge About Temporal Intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832-843, November 1983.

[5] J. P. Anderson, "Computer Security Technology Planning Study," Electronic Systems Division, Air Force Systems Command, Bedford, Massachusetts, Technical Report ESD-TR-73-51, Vol. II, October 1972.

[6] J. P. Anderson, S. Hoffman, A., J. Shifman, and R. J. Williams, "D825 - A Multiple-Computer System for Command & Control," in *Proceedings of the Fall Joint Computer Conference*, 1962, pp. 86-96.

[7] C. G. Bell, "Fundamentals of Time Shared Computers," *Computer Design*, vol. 7, nos. 2 and 3, pp. 44-59 and pp. 38-46, February and March 1968.

[8] A. Bensoussan and C. T. Clingen, "The Multics Virtual Memory: Concepts and Design," *Communications of the ACM*, vol. 15, no. 5, pp. 308-318, May 1972.

[9] H. W. Bingham, "Security Techniques for EDP of Multilevel Classified Information," Rome Air Development Center, Griffis Air Force Base, New York, Technical Report RADC-TR-65-415, December 1965.

[10] C. Bryce and M. Gilles, "Matching Micro-Kernels to Modern Applications using Fine-Grained Memory Protection," in *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing*, October 25-28, 1995, pp. 272-279.

[11] D. Burger and T. M. Austin, "The SimpleScalar Tool Set Version 2.0," University of Wisconsin-Madison, Computer Sciences Department, Technical Report, June 1997.

[12] Burroughs Corporation, "The Descriptor - A Definition of the B5000 Information Processing System," http://www.cs.virginia.edu/brochure/images/manuals/b5000/descrip/descrip.html, 1961, Last accessed: June 2005.

[13] N. P. Carter, S. W. Keckler, and W. J. Dally, "Hardware Support for Fast Capability-based Addressing," in *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 5-7, 1994, pp. 319-327.

[14] J. S. Chase, H. M. Levy, M. Baker-Harvey, and E. D. Lazowska, "How to Use a 64-Bit Virtual Address Space," Department of Computer Science and Engineering, University of Washington, Technical Report 92-03-02, March 1992.

[15] J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska, "Sharing and Protection in a Single-Address-Space Operating System," *ACM Transactions on Computer Systems*, vol. 12, no. 4, pp. 271-307, November 1994.

[16] J. Cocke and H. G. Kolsky, "The Virtual Memory in the STRETCH Computer," in *Proceedings of the Eastern Joint Computer Conference*, December 1-3, 1959, pp. 82-93.

[17] F. J. Corbató, M. Merwin-Dagget, and R. C. Daley, "An Experimental Time-Sharing System," in *Proceedings of the Spring Joint Computer Conference*, May 1-3, 1962, pp. 335-344.

[18] F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System," in *Proceedings of the Fall Joint Computer Conference*, November 30, 1965, pp. 185-196.

[19] J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems," *Journal of the Association for Computing Machinery*, vol. 12, no. 4, pp. 589-602, October 1965.

[20] J. B. Dennis and E. C. Van Horn, "Programming Semantics For Multiprogrammed Computations," *Communications of the ACM*, vol. 9, no. 3, March 1966.

[21] R. S. Fabry, "Capability-Based Addressing," *Communications of the ACM*, vol. 17, no. 7, pp. 403-412, July 1974.

[22] P. Fagg, J. L. Brown, J. A. Hipp, and D. T. Doody, "IBM System/360 Engineering," in *Proceedings of the Fall Joint Computer Conference*, 1964, pp. 205-231.

[23] J. Fotheringham, "Dynamic Storage Allocation in the Atlas Computer, Including an Automatic Use of a Backing Store," *Communications of the ACM*, vol. 4(10), October 1961.

[24] K. Ghose and P. Vasek, "A Fast Capability Extension to a RISC Architecture," in *Proceedings of the 22nd EUROMICRO Conference*, September 2-5, 1996, pp. 606-613.

[25] E. L. Glaser, J. F. Couleur, and G. A. Oliver, "System Design of a Computer for Time Sharing Applications," in *Proceedings of the Fall Joint Computer Conference*, 1965, pp. 197-202.

[26] G. S. Graham and P. J. Denning, "Protection---Principles and Practice," in *Proceedings of the Spring Joint Computer Conference*, May 16-18, 1972, pp. 417-429.

[27] R. M. Graham, "Protection in an Information Processing Utility," presented at *ACM Symposium on Operating System Principles*, Gatlinburg, Tennessee, 1967.

[28] T. F. J. Hatch and J. B. Geyer, "Hardware/Software Interaction on the Honeywell Model 8200," in *Proceedings of the Fall Joint Computer Conference*, December 9-11, 1968, pp. 891-901.

[29] L. J. Hoffman, "Computer and Privacy: A Survey," *ACM Computing Surveys*, vol. 1, no. 2, pp. 85-103, June 1969.

[30] IBM Corporation, PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors, 2000.

[31] Intel Corporation, "System Programming Guide," in Intel Architecture Software Developer's Manual, vol. 3, 1997.

[32] Intel Corporation, "IA-64 System Architecture," in Intel IA-64 Architecture Software Developer's Manual, vol. 2: Intel Corporation, 2000.

[33] G. Kane, PA-RISC 2.0 Architecture. Upper Saddle River, NJ: Prentice Hall PTR, 1996.

[34] P. A. Karger, "Improving Security Performance for Capability Systems," PhD Dissertation, University of Cambridge, Cambridge, England, 1988.

[35] P. A. Karger, "New Methods for Immediate Revocation," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 1-3, 1989, pp. 48-55.

[36] T. Kilburn, D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, "One-level Storage System," *IRE Transactions on Electronics and Computers*, vol. 2, pp. 223-235, April 1962.

[37] T. Kilburn and R. B. Payne, "The Atlas Supervisor," in *Proceedings of the Eastern Joint Computer Conference*, December 12-14, 1961, pp. 279-294.

[38] E. J. Koldinger, J. S. Chase, and S. J. Eggers, "Architectural Support for Single Address Space Operating Systems," Department of Computer Science and Engineering, University of Washington, Seattle, Washington, Technical Report 92-03-10, July 1992.

[39] S. H. Lavington, "The Manchester Mark I and Atlas: A Historical Perspective," *Communications of the ACM*, vol. 21(no. 1), pp. 4-12, January 1978.

[40] D. Lie, C. A. Thekkath, and M. Horowitz, "Implementing an Untrusted Operating System on Trusted Hardware," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, October 19-22, 2003, pp. 178-192.

[41] D. Lie, C. A. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 168-177.

[42] E. J. McCauley and P. J. Drongowski, "KSOS---The Design of a Secure Operating System," in *Proceedings of the National Computer Conference*, June 4-7, 1979, pp. 345-353.

[43] B. Peters, "Security Considerations in a Multi-programmed Computer System," in *Proceedings of the Spring Joint Computer Conference*, April 18-20, 1967, pp. 283-286.

[44] G. J. Popek and C. S. Kline, "Verifiable Secure Operating System Software," in *Proceedings of the National Computer Conference and Exposition*, May 6-10, 1974, pp. 145-151.

[45] J. H. Saltzer, "Protection and the Control of Information Sharing in Multics," *Communications of the ACM*, vol. 17, no. 7, pp. 388-402, July 1974.

[46] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," in *Proceedings of the IEEE*, vol. 63, no. 9, September, 1975, pp. 1278-1308.

[47] M. D. Schroeder and J. H. Saltzer, "A Hardware for Implementing Protection Rings," *Communications of the ACM*, vol. 15, no. 3, pp. 157-179, March 1972.

[48] J. I. Schwartz, E. G. Coffman, and C. Weissman, "A General-Purpose Time-Sharing System," in *Proceedings of the Spring Joint Computer Conference*, 1964, pp. 397-411.

[49] J. S. Shapiro, J. M. Smith, and D. J. Farber, "EROS: A Fast Capability System," *Operating Systems Review*, vol. 34(5), pp. 170-185, December 1999.

[50] T. Shinagawa, K. Kono, and T. Masuda, "Fine-grained Protection Domain Based on Segmentation Mechanism," in Proceedings of the Japan Society for Software Science and Technology, *2003 Workshop on Systems for Programming and Applications*, March 20-22, 2000.

[51] G. E. Short, "Threats and Vulnerabilities in a Computer System," in Data Security and Data Processing, vol. 5, Study Results: TRW Systems, Inc.: IBM Corporation, Technical Publications - Systems, Department, 1974.

[52] F. M. Stepczyk, "Requirements for Secure Operating Systems," in Data Security and Data Processing, vol. 5, Study Results: TRW Systems, Inc.: IBM Corporation, Technical Publications - Systems, Department, 1974.

[53] V. A. Vyssotsky, F. J. Corbató, and R. M. Graham, "Structure of the Multics Supervisor," in *Proceedings of the Fall Joint Computer Conference*, 1965, pp. 203-212.

[54] W. H. Ware, "Security and Privacy in Computer Systems," in *Proceedings of the Spring Joint Computer Conference*, April 18-12, 1967, pp. 279-282.

[55] W. H. Ware, "Security and Privacy: Similarity and Differences," in *Proceedings of the Spring Joint Computer Conference*, April 18-20, 1967, pp. 287-290.

[56] W. H. Ware, "Security Controls for Computer Systems (U): Report of Defense Science Board Task Force on Computer Security," The RAND Corporation, Santa Monica, CA, February 1970.

[57] C. Weissman, "Security Controls in the ADEPT-50 Time-Sharing System," in *Proceedings of the Fall Joint Computer Conference*, November 18-20, 1969, pp. 119-133.

[58] E. Witchel, J. Cates, and K. Asanovic, "Mondrian Memory Protection," in Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems, October 5-9, 2002, pp. 304-316.

[59] E. J. Witchel, "Mondriaan Memory Protection," PhD Dissertation, Massachusetts Institute of Technology, 2004.

[60] W. D. Young, "Verifiable Computer Security and Hardware: Issues," Computational Logic Inc., Austin, Texas, Technical Report 70, September 1991.