Hardware Requirements for Secure Computer Systems: A Framework[†]

Carl E. Landwehr

Computer Science and Systems Branch
Naval Research Laboratory
Washington, D.C.

John M. Carroll

Computer Science Dept.,
University of Western Ontario,
London, Ontario,
Canada

ABSTRACT

This report develops a new set of criteria for evaluating computer architectures that are to support systems with security requirements. Central to these criteria is the concept of a *domain*, here interpreted as a set of information and authorizations for the manipulation of that information in a computer system. Architectural requirements are grouped in three categories: logical structure, the processing of logical structures, and physical structure. These criteria were developed in order to assess the utility of Navy standard computers as bases for secure embedded systems, but they are not specific to those computers.

1. Introduction

Many Navy embedded computer systems must satisfy stringent security requirements. To meet these requirements while imposing minimal constraints on the operational environment, system software and hardware must enforce appropriate controls on the system. Experience has shown that building systems to meet these requirements is difficult and that the architecture of the chosen computer can be a significant factor in determining whether security requirements can be met[1]. In many cases, systems unable to meet requirements have had to operate under waivers or substitute expensive and inconvenient physical controls and clearance procedures for controls lacking in the system.

2. The Problem

Security requirements can be expressed as a set of assertions that a system must enforce. An example assertion is, ‘‘No user can display information that has a classification higher than his clearance.’’ There are several ways to enforce such an assertion about a computer system: all classified information can be removed from the system, physical controls can deny access to computer terminals to any user not cleared for all of the information in the system, all users can be cleared to the highest level, or the controls can be implemented in the computer so that it can compare the classification of specific information with the clearance of a particular user and determine whether or not to display the requested information. The first three solutions can be imposed on any computer, regardless of its hardware architecture, but they impose substantial constraints on the outside environment. The last solution constrains the computer

† Published as Landwehr, C.E., and J.M. Carroll, "Hardware Requirements for Secure Computer Systems: A Framework," *Proc. 1984 IEEE Symposium on Security and Privacy*, Oakland, CA, April 23-26, 1984.

system, but leaves the environment relatively unrestricted. Solutions of this kind are of primary interest here.

Other security assertions are possible, and lists have appeared elsewhere[2,3]. The question addressed here is ‘‘What hardware architectures simplify both the enforcement of assertions like these and the demonstration to observers that these assertions are enforced?’’

3. Role of Verification Technology

One approach to answering this question is based on techniques for top-down design and program verification. For systems developed using this method, the security assertions would be stated formally as part of the top level system specification, and as successively more detailed refinements of the design were created, each would either be proven directly to enforce the assertions or to correspond to the previous refinement, which enforces the assertions. The final, most detailed refinement could be the programs. Alternatively, this approach could be extended to the underlying hardware; parts of the formal specifications for the Honeywell SCOMP correspond to functions implemented by hardware[4].

If all of the required security assertions could be proven about the system software, hardware structure might be of less concern. The remaining issues would be whether the compiler generates code correctly and whether the hardware correctly implements the semantics assumed by the compiler, since the programs would have been proven not to violate the security assertions. In theory, even if the underlying hardware provided no more structure than a Turing machine, security violations would be impossible. Some Burroughs systems have relied on a related kind of protection: they are vulnerable to programs written in assembly language, but software controls allow users to program only in higher order languages. That is, users can only load and execute code that has been compiled by a certified compiler. The security of such a system depends on preventing users from creating their own translators (or coercing certified compilers) to generate insecure assembly language programs. (Wilkinson[5] provides a description, with an example of how the controls failed). It is also difficult to prevent users from generating, via a certified compiler, programs that violate security, because compilers can often be subtly coerced into generating and initiating execution of arbitrary bit strings.

In practice, the approach to verification described above (proving correspondence between specifications and implementation, down to the structures provided by hardware) is not yet feasible for systems of substantial size. Even if one could verify a system’s properties down to the the chips, the system would still be open to attacks, such as those that exploit hardware malfunctions or software errors, based on gaining access to the system through paths other than the ‘‘proven’’ top level set of functions. Consequently, even with the benefits that program verification can bring, characteristics of a machine’s architecture are still important in determining whether it is a suitable base for building systems that must enforce security assertions.

4. Requirements for Hardware Architectures to Support Secure Applications

Past studies have developed general criteria for secure systems [6,7] and a few have focussed specifically on computer hardware considerations [8,9]. The latter work, however, has been based on requirements developed from a specific security model and the notion of a security kernel. The DoD Computer Security Evaluation Center (CSEC) has developed a set of criteria for trusted computing bases[10] that mentions hardware features but does not address hardware requirements directly. Thus, there is no generally agreed upon set of requirements for hardware to support secure systems.

We seek to develop such requirements based on the structure of the problem rather than on a specific security model and implementation approach. Below we identify requirements for hardware structures that are to support secure processing. These requirements are grouped according to whether they concern (1) the logical structure of the hardware and its ability to support software structures of secure systems, (2) the ability of the hardware to process these structures rapidly enough to be of practical use, or (3) the ability of the hardware to resist external physical attacks (*e.g.*, eavesdropping, physical removal of media). The requirements and their categorization are based on the sources already cited and on the history of efforts to develop secure computer systems[1]. While we expect this set of requirements may be augmented or revised in the future, particularly if additional constraints are imposed according to the system to be implemented, we believe that the framework will remain a coherent way to assess the ability of a particular hardware architecture to support security requirements.

4.1. Requirements on Logical Structure

The abstract machine visible to system programmers defines the logical structure of the computer. There may be many ways to realize a given logical structure physically. In this subsection, we present four kinds of requirements on logical machine structure and then give examples of how each can be met in actual systems.

Requirements

Secret is derived from a verb that means ‘‘to distinguish’’ and whose components are *se* -- ‘‘apart’’ and *cernere* -- ‘‘to sift’’[11]. Thus, secrets represent information that is sifted apart from other information. For a computer to be able to keep secrets, it must be able to sift apart sensitive information from non-sensitive and cleared users from uncleared. This requirement leads to the concept of *domains* within the computer system. A domain is a set of information and authorizations for the manipulation of that information within a computer system. An instance of an abstract data type can define a domain, since it includes a set of information and a set of operations that are authorized to manipulate it. In some systems, the concept of process is identified with that of domain: every resource in the system belongs to some process that is authorized to use it without restriction[12]. Although hardware for systems based on the former view of domain may differ from hardware designed to support the latter, it is not necessary to choose among competing interpretations of the domain concept at the level of abstraction addressed here.

Lampson uses the term ‘‘domain’’ to refer to an entity that has access rights to objects, which could include files, processes, and other domains. These abstractions are rarely implemented directly by computer hardware; the concern here is how to implement domains using the constructs hardware commonly provides. For this purpose, we view a computer as providing a *name space*, a *value space*, a *mapping function* from the name space to the value space, and a set of *instructions* that can affect the mapping function and the name and value spaces. A domain comprises a subset of the instructions, a subset of the name space, a corresponding subset of the value space defined by a particular mapping function, and a set of authorizations (sometimes called *access modes*) that determine the instructions that the domain may apply to each element of the name space. The mapping function is sometimes referred to as a *context*, since it defines the environment that gives specific meaning to a name.

The name space is the set of addresses that a program can generate, the value space is the contents of the locations corresponding to these addresses, and the mapping function is defined by the tables or registers (if any) used to translate program-generated addresses (names) to physical addresses (locations). The value space refers only to primary storage; access to secondary storage is determined by the domain’s instruction subset, mapping function, and authorizations.

Define and Separate Domains

Any computer system that allows concurrent operations must include provisions for defining separate domains and protecting actions in one domain from improperly affecting those in another domain. Rushby and Randell[13] identify four ways to separate domains: physically, temporally, cryptographically, and logically. Particular computer hardware may simplify one or another of these approaches, but the requirement that the logical structure of the machine provide a way for programmers to define and separate domains is fundamental.

Establish Initial Domain

The ability to define and separate domains in a machine will be of little use unless there is a reliable way of getting started. This implies that the logical structure of the system must allow the programmer to distinguish the occurrence of a system initialization event and to establish a consistent state for the system -- a state from which additional domains can be initiated and separated. If the initial domain provides access to all of the information in the system, great care is necessary in constructing programs that operate in it. Arbitrary programs should not be able to invoke system initialization.

Link Users with Domains

Security requires that actions be attributable to individuals. This principle leads to a requirement for a link between a user and a domain. So that operations invoked in a domain can be traced to a specific user, the logical structure of the hardware must support the creation and destruction of links between users and domains. ‘‘Link’’ need not be interpreted as a physical entity; it merely denotes an element of a mapping from users to domains.

Control Communication between Domains

In most applications, absolute isolation of domains is impractical -- some information must be shared. The logical structure of the machine must support sharing with mechanisms that can limit communication in accordance with application requirements. For example, one domain may need to provide parameters to a function performed in a different domain. The hardware should support passing of these parameters without making additional information available to the recipient.

Detect and Handle Faults

Finally, the logical structure of the machine must reflect the possibility of machine faults. These can never be eliminated entirely, and they can be a source of security breaches. If the occurrence of hardware errors is concealed from programs, it will be impossible to protect them from the effects of those errors. Consequently, it is required that the logical machine structure include fault detection and fault handling mechanisms.

Examples

Definition of Domains

Techniques for segregating name/value spaces on the same computer include the following:

Segregating spaces in time: programs in early batch processing systems employed the same name space (*i.e.*, the entire address space of the machine) but were isolated in time. Programs did not interfere with each other so long as they were not executed concurrently. The operating systems that enforced this condition required mechanisms to keep themselves from being modified by user programs. Swapping systems are also based on this kind of isolation. Execution of different programs can be interleaved, but each program’s value space must be saved when its execution is suspended and restored when it is re-initiated.

Segregating spaces through mapping functions: different programs generate the same names, but all names are interpreted by mapping them to values. If the ranges of the mapping functions are disjoint, programs cannot interfere even if they are executed simultaneously. If the ranges of the mapping functions for two programs overlap, interference (or communication) is possible. Mechanisms for implementing mapping functions include relocation registers, base/bounds registers, and virtual memory mapping registers and tables. The mechanisms that effect the mapping should not themselves be in the name space of user programs (otherwise users can directly alter the mapping). The real address space of the machine need not be the same size as the name space available to a particular domain.

Segregating spaces through access permissions: different programs can generate names that map to the same value, but may have different access permissions for that value. For example, a program may be allowed to read a certain part of physical storage but not to write it. Whether a given machine associates access permissions with the name space, the mapping, or the value space will affect the flexibility of its protection system: if permissions are associated only with value space and some part of that space is shared, all parties sharing it will necessarily have the same access rights to it. Example of these kinds of controls include hardware read-only memory, storage keys, and tags on descriptors or capabilities.

Limiting a domain to a subset of the machine instruction set is accomplished through hardware-defined *modes* of execution. The current mode of execution (*e.g.*, ‘‘kernel mode’’ ‘‘executive mode’’, ‘‘user mode’’) is defined by the contents of a hardware register and determines whether a particular machine instruction is allowed. On some machines, the mode also imposes constraints on the name and value spaces for the domain. At least two modes are required in order to control access to the map of names to values. Several other functions are normally controlled in this way, including the ability to initiate I/O, handle interrupts and traps, and halt the machine. The instruction subsets defined by different modes are usually hierarchical, so that each more privileged mode includes all the instructions of its predecessors.

The accompanying table exhibits some of the differences in the way domains are defined by the DEC VAX 11/780 and two Navy standard embedded computers.

Establishing an Initial Domain

On some computers, the initial domain is established by loading a state word from a predefined location in response to a specific interrupt. This stateword places the machine in its most privileged mode and defines the location from which the next instruction (normally the start of a bootstrap loader) is to be fetched. Current medium scale and larger systems frequently include a separate processor to handle the system console and other utility functions, including system initialization. Initialization can include loading the microcode for the main processor itself as well as establishing its initial program load.

Linking Users with Domains

A user initially establishes a link with a domain through an exchange of passwords or some other form of authentication data. The user must authenticate the machine as well as vice versa. The user may be provided with the serial number of the processor, or he may arrange for an exchange of questions and answers with a program operating in an initial domain to confirm that he is accessing the intended machine.

Once the link is established, if a domain initiates a new domain in response to a user request, the new domain is accountable to the same user. An exception occurs when the initial domain (accountable to the system administrator) is executing the login procedure, and once a new user is authenticated, a new domain is established for which the new user is accountable. Links can be destroyed when the domain ceases to exist, or if accountability for the domain passes to another user. The establishment and maintenance of these links is primarily under software control, but their correctness depends on the hardware path between user and machine. Although this path can be made resistant to physical tampering, the strongest non-physical protection is provided by encryption.

Some hardware devices periodically re-authenticate the user, to assure that the user has not changed (*e.g.*, if one user leaves his terminal running and is replaced by another user), and software may be designed similarly. A hardware mechanism for implementing watchdog timers can help enforce periodic re-authentications by software.

Controlling Communication between Domains

Communication among domains can occur in several ways: (a) one domain may request service from a less-restricted one, (b) two domains may share part of the same value space directly, or (c) domains may pass messages to each other. Unless links are set up when the domains are created, the first kind of communication is a prerequisite for the other two: in order for two isolated domains to initiate address-space sharing or exchange messages, one of them must request that service from a domain that can alter mapping functions or pass messages.

Service requests can be decomposed into three parts: invoking the more privileged domain and passing arguments, service performed by the privileged domain, and returning arguments and control to the requesting domain. Protected entry points (‘‘gates’’) and ‘‘supervisor call’’ interrupts allow a privileged domain to control its invocation. Where hardware provides a hierarchy of three or more domains, it can limit the number of levels an individual request can traverse (*e.g.*, rings and ring brackets). An addressing mode that limits the privileged domain to the access rights of the requesting domain can help prevent the privileged domain from performing unauthorized operations on behalf of the requester. Such a mode is unnecessary in capability-based systems, since possession of a capability implies the right to use it. Mechanisms to protect a requesting domain from being entered at an improper location on completion of a request are rarely implemented, since the more privileged domain is normally considered benevolent and reliable.

When two domains share part of an address space, changes made by one domain are directly visible to the other. Control of address-space sharing is aided by access-mode tags on parts of the mapping function. For example, one-way communication between domains is possible if the shared space is limited to read-only in the receiver's map but is read/write or write-only in the sender's. Ring brackets can also be used to limit the ability of different domains to write in a shared space.

Communication via messages is less direct, so the need for authentication is greater. Mechanisms that can label a message with an unforgeable tag identifying the sending domain allow the receiving domain to authenticate its source. Encryption techniques can be applied to this problem.

Fault Detection and Handling

Hardware is subject to faults, and unless these are detected and compensated for, security can be compromised. Some hardware mechanisms for fault tolerance, such as error detection and correction codes on units of storage, can function without software assistance and may not be visible in the logical structure of the machine. Nevertheless, conditions will remain in which machine-detectable errors cannot be recovered without software assistance.

The logical machine structure should accommodate techniques for detecting, signaling, and recovering from machine errors. Detection methods are based on redundancy: a stored checksum is compared with a newly computed version, a critical value is checked against stored boundary conditions, or the same result is computed in two different ways. Programs that perform these functions may require access to particular domains (*e.g.*, the ability to read certain programs or storage locations), but do not impose specific requirements on logical machine structure. Once an error is detected, a safe method to enter a domain where the error can be handled (perhaps by stopping the machine) is needed. Standard interrupt and trap mechanisms can be used to handle this problem. The error-handling domain may have access to special machine instructions for diagnosing problems or reconfiguring the system.

4.2. Requirements on Processing of Logical Structures

In addition to supporting appropriate logical structures, the underlying hardware must be able to process those structures expeditiously. Several efforts to develop secure systems have attributed their performance problems to the inability of underlying hardware to switch rapidly between domains. In operational systems, if security checks cannot be implemented efficiently, they are likely to be omitted. This subsection describes requirements based on the logical structures discussed above and gives examples of mechanisms intended to satisfy them.

Requirements and Examples

Creating and destroying domains. Systems with stringent security requirements are likely to require a large number of small domains. Consequently, the time and storage required to create or destroy a domain are important. Relevant aspects of machine structure include the addressing structure, which determines the smallest amount of storage that can be allocated to a domain, the size of the of the system state, which determines how many different registers and storage locations must be initialized, and the availability of specialized instructions to clear residues and manipulate system state information. If creating and destroying domains is time-consuming, sometimes software can be organized so that the set of domains is static.

Switching domains. The speed with which the processor can suspend execution of a program in one domain and initiate execution of a program in a different domain is important for the same reasons listed in the previous paragraph, but the effects of high overhead in this operation are harder to remedy. Domains can be combined to reduce the need for switching, but this defeats the principles of least privilege and isolation of domains. Mechanisms that facilitate domain-switching include multiple register sets, which reduce requirements for saving and restoring registers, instructions that can save or restore the machine state in a single operation, interrupt and trap mechanisms that automatically initiate a domain-change when invoked, and a machine structure that minimizes or eliminates residual information that must be cleared when a domain switch occurs.

Moving information between domains. Hardware must be able to pass information efficiently between domains while enforcing security constraints. Hardware that could automatically check the security levels associated with domains and data to be passed between domains has been proposed but not implemented. Nevertheless, a variety of machines have included hardware that can assist in this task. For example, argument validation can be assisted by automatically passing the mode of the sending domain to the receiving domain, rings of protection can be used to control the domains from which information may be passed, and gates can be used to control the locations in a domain to which control can be passed.

Security checking on operations within a domain. Security checks that are to be made on each reference to an object within a domain must either be performed on each access with virtually no overhead or else a single check (*e.g.*, when a file is opened or a segment first referenced) must suffice for a large number of accesses. Hardware that supports virtual memory (*i.e.*, automates the mapping from name to value space) can be used to check the legality of each reference according to the current mapping function; if the access authorizations (read, write, execute) are associated with virtual memory addresses, these too can be checked on each reference. It is conceivable that hardware could support security level checking in the same way as access authorization checking. Systems have been built that utilize software checks at the first reference to a segment or file as well. If the check succeeds, the information can then be mapped into the user's domain, and the addressing mechanisms prevent references outside the domain. Base/bounds checks, virtual memory mapping mechanisms, associative stores, and data caches can help satisfy this requirement.

Moving information between levels of the storage hierarchy. Input/output may involve moving information within a single domain or between two different domains. It has frequently caused problems in secure system developments. Often the difficulties can be traced to the logical organization of the I/O subsystem, which may not mesh with the organization of addressing within primary storage. Although these are difficulties in the logical structure, they can cause performance problems -- because of the logical structure, the I/O must be handled by a highly privileged domain, which increases the demand for domain switches and inter-domain communication. An I/O structure that uses the same kind of interface to primary storage as the CPU (*i.e.*, the same mapping mechanism) can limit these problems.

4.3. Requirements on Physical Structure

Even if a computer provides the appropriate logical structures for building a secure system and it processes those structures effectively, it may be subject to physical threats. In this section we enumerate some needs for physical protection and methods for satisfying them.

Requirements and Examples

Prevent unauthorized physical access to the computer. Placing equipment in locked rooms, providing guards, and installing authentication systems do not directly affect the system hardware. The hardware itself can contribute to meeting this requirement by providing console, terminal, and cabinet locks, and having covers for control panels and keyboards. Protected usage meters can detect misuse of resources that occurs when physical controls are defeated.

Prevent unauthorized modification of removable media. Some removable media need to be protected against alteration (*e.g.*, master copies of programs and data bases). Write inhibit switches on disk drives and the physical rings on magnetic tapes address this requirement. A tape drive used for collecting audit data can be inhibited against rewinding to assure that the information is not be modified after it is written.

Assure secure communication with remote system components. Physical means can be used to secure connections between system components from wiretapping, and encryption can help assure that information that a wiretapper obtains is still protected.

Prevent unauthorized viewing of system output. Video display devices can be designed so that they are only visible head-on, to reduce visual eavesdropping. TEMPEST controls also address this requirement, since electromagnetic emanations are a form (albeit unintended) of system output.

Assure continuous service. Reliability is associated with security for several reasons: (1) hardware failures may defeat security checks, (2) a system is often more vulnerable to penetration while recovering from a failure, (3) failures may be used to obscure penetrations, (4) when a system fails frequently, operators and managers become insensitive to small changes in behavior that otherwise would give warning of attempted penetration, and (5) managers of unreliable systems, trying to improve system performance, may take shortcuts that diminish system security.

Many hardware features that contribute to security are designed primarily to increase system reliability. Some of them are:

(1) Redundant devices and modules, especially power supplies and and memory modules. Some systems can be reconfigured in case of partial failure. Security measures must not be inadvertently bypassed when the system is automatically reconfigured.

(2) Protection of storage against power failure.

(3) Coolant leak detectors.

(4) Manual input devices designed to reduce human error (*e.g.*, raised separators between control-panel keys).

(5) Annunciator lamps to warn that protective features have been intentionally overridden (‘‘battle short’’).

5. Summary and Conclusions

Security requirements for a system depend on its functions, the kinds of data it processes, the other systems with which it communicates, and the environment in which it operates. There are few specific properties that hardware *must* have if it is to support systems that have security requirements, but we have tried to develop a framework for that illuminates those properties of hardware that determine how well it will be able to support such requirements. In constructing the framework, we have tried to avoid assumptions about the specific security properties desired or the specific methodology used to design the system that implements them.

The requirements identified concern the logical structure of the computer, its ability to process those logical structures, and its physical structure. The logical structure should provide mechanisms for (1) defining and separating domains, (2) establishing an initial domain, (3) linking domains with users, (4) controlling communication between domains, and (5) detecting and handling faults. Expeditious processing of logical structures requires mechanisms for (1) creating and destroying domains, (2) switching domains, (3) moving information between domains, (4) checking the security of operations within a domain, and (5) moving information between levels of the storage hierarchy. Requirements on physical structure concern preventing unauthorized physical access to the computer, unauthorized modification of removable media, and unauthorized viewing of system output while assuring both secure communication with remote components and continuous service.

Although quantitative comparisons of different computer architectures are not within the scope of this scheme, we believe that comparisons of alternative systems within this framework will reveal qualitative differences in their abilities to support secure systems. The framework has been applied to two Navy computers[14], and we welcome efforts to apply it to other systems.

Acknowledgements

We thank Virgil Gligor and Stan Wilson for helpful comments on earlier drafts of this paper and H.O. Lubbes of the Naval Electronic Systems Command for his continuing encouragement and support.

References

[1] C. E. Landwehr, ‘‘Best available technologies for computer security,’’ IEEE *COMPUTER, Vol. 16*, No. 7, pp. 86-100, July, 1983.

[2] C. E. Landwehr, C. L. Heitmeyer, and J. McLean, ‘‘A security model for military message systems,’’ NRL Report, to appear, winter, 1984.

[3] J. K. Millen, ‘‘Kernel isolation in the PDP-11/70,’’ roc. 1982 IEEE Symposium on Security and Privacy, pp. 57-65, April, 1982.

[4] L. J. Fraim, ‘‘SCOMP: a solution to the MLS problem,’’ IEEE *COMPUTER*, *Vol. 16*, No. 7, pp. 26-46, July, 1983. July 1983.

[5] A. L. Wilkinson, *et. al.*, ‘‘A penetration analysis of a Burroughs large system,’’ ACM SIGOPS *Operating Systems Review 15, 1* pp. 14-25, Jan. 1981.

[6] R. B. Blue, Sr., and G. E. Short, ‘‘Computer system security technology and operational experience,’’ TRW Systems, Redondo Beach, CA, March 1974.

[7] *Data security and data processing, Vol. 5*, Joint study by IBM Corp., MIT, TRW Systems, and State of Illinois (Management Information Division). IBM Report Nos. G320-1370 -- G320-1376, 1974.

[8] L. Smith, ‘‘Architectures for secure computing systems,’’ ESD-TR-75-51, April 1975. Available as NTIS AD A009221.

[9] J. Tangney, ‘‘Minicomputer architectures for effective security kernel implementations,’’ ESD-TR-78-170, October 1978.

[10] *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83, DoD Computer Security Center, Ft. Meade, MD, 15 August 1983.

[11] *Webster’s Third New International Dictionary, Unabridged*, G.& C. Merriam and Co., Springfield, Mass., 1971.

[12] B. W. Lampson, ‘‘Protection,’’ *Proc. Fifth Princeton Symp. on Inf. Sci. and Systems*, Princeton U., pp.437-443, March, 1971, reprinted in SIGOPS *Operating Systems Review, 8*, 1 pp. 18-24, Jan., 1974.

[13] J. M. Rushby, and B. Randell, ‘‘A distributed secure system,’’ IEEE *COMPUTER, Vol. 16,* No. 7, pp. 55-68, July, 1983.

[14] J. M. Carroll, and C. E. Landwehr, ‘‘Hardware security considerations for the AN/UYK-43 and AN/UYK-44 computers,’’ NRL Technical Memorandum 7590-008, 16 January, 1984.

Examples of Domains in Specific Machines

Attribute	VAX 11/780[a]	AN/UYK-43[b]	AN/UYK-44[c]
Instruction Subset Sizes	Kernel: 325 Executive: 320 Supervisor: 320 User: 310	Executive: 215 Task: 159	Executive: 294 Task: 257
Name Space Size[d]	All modes: 2^{32}	Executive: 2^{32} Task: 2^{19}	Executive: 2^{22} Task: 2^{16}
Mapping Mechanism	Segment Base Register, Page Table Entry	Segment Register	Page Address Register
Mapping Granularity	512-byte page	64K-word segment[e]	1024-word page
Hardware-enforced Access Modes on Granules	15 modes: no access, read read/write for each of 4 privilege levels, with nesting[f]	16 modes: read, write, execute, allow indirect-addressing, independently settable	8 modes: read, write, execute, independently settable
Value Space[g] Size	All modes: 2^{23}	Executive: $5{\times}2^{19}$ Task: 2^{19}	Executive: 2^{20} Task: 2^{16}

a. Configured with maximum primary storage.

b. B-enclosure, configured with maximum primary storage (but excluding Computer Interconnection System). Source: *AN/UYK-43 Navy Embedded Computer System (NECS) Technical Description*, Sperry Univac, 1982.

c. Configured with Memory Address Expansion and expansion cabinet for maximum primary storage. Source: *AN/UYK-44 Navy Embedded Computer System (NECS) Technical Description*, Sperry Univac, 1982.

d. Number of distinct addresses a program operating in the specified mode can generate without intervention by programs operating in a more privileged mode.

e. Use of the Storage Protection Register can reduce the effective segment size to a single word, but a maximum of 8 segments are directly addressable at any time.

f. Granting an access mode to one privilege level implies that mode is granted to all higher levels as well (e.g., if (read/write, supervisor) then also (read/write, executive) and (read/write, kernel)).

g. Maximum number of distinct primary storage addresses to which names can be mapped without intervention by programs operating in a more privileged mode. Bounded by size of name space and quantity of primary storage that can be attached to a processor. Units for the VAX 11/780 are 8-bit bytes, for the AN/UYK-43, 32-bit words, and, for the AN/UYK-44, 16-bit words.