ISSN: 2167-1435
v30 n2511

Computer Security Primer: Systems Architecture, Special Ontology and Cloud Virtual Machines

Leslie J. Waguespack, Ph.D.
lwaguespack@bentley.edu
Computer Information Systems Department
Bentley University
Waltham, Massachusetts 02452, USA

Abstract

With the increasing proliferation of multitasking and Internet-connected devices, security has reemerged as a fundamental design concern in information systems. The shift of IS curricula toward a largely organizational perspective of security leaves little room for focus on its foundation in systems architecture, the computational underpinnings of processes and protection. Yet these architectural features are the foundation of systems security for all the layers above that they enable. They are also the prototypical mechanisms of protection that must be modeled throughout systems design to realize system security: confidentiality, integrity and availability. This paper presents a learning unit that proposes a special ontology of computer system architecture to explain computer security on the host-level and by extension the emerging standard security architecture of the cloud, the virtual machine. The ontology appears as a prose tutorial, a set theoretic model, and a two-page study reference that facilitates a security discussion ranging from host architecture to web-services. This treatment is a concise, self-contained module for standalone use or embedded in a systems course (analysis, modeling, design, database or systems architecture) where complete operating system or computer organization coverage may not be feasible.

Keywords: computer security, computer protection, special ontology of systems architecture, virtual machine, IS pedagogy

1. INTRODUCTION

The proliferation of multitasking personal devices and Internet-connected users thrusts security into the forefront of information system design considerations. Even casual computer users are beset with security concerns and must rely on the device and network designer for protection from violations of confidentiality, integrity or availability. Coincidentally, the shift of IS curricula toward a broader, organizational perspective on security leaves little room in curricula for a focus on the computational underpinnings of processes and protection that are the foundation of computer system security. (Topi, Valacich, Wright, Kaiser, Nunamaker, Sipior & de Vreede, 2010) These architectural features not only form the basis of systems security for all the system layers above them that they enable but, they also represent prototypical mechanisms of protection in organizational systems security. This computer security primer that follows offers an option to fill a curricular gap.

The primer begins with an abbreviated literature survey of the theory, policy and application of computer security as a context and a reading list for students who may wish to delve much deeper. Then follows the special ontology of systems architecture, a framework for explaining the role of protection in host computer security. That framework

includes the protection provided by virtual machine architecture, the primary design platform for security in cloud computing. Finally there are some brief thoughts on applying the security primer as a learning unit in undergraduate curricula. The primer is a concise self-contained module suitable for use standalone or embedded in a systems course (analysis, modeling, design, database or systems architecture). Appendices provide a set-theoretic representation of the ontology and a two-page reference handout as a study guide.

2. WHAT IS COMPUTER SECURITY?

The model posed in (McCumber, 1991) has stood the test of time as the de facto definition for computer security (See Figure 1).

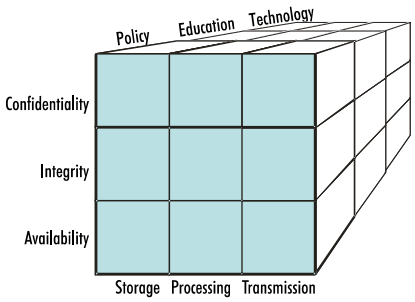


Figure 1 – IS Security Model: McCumber

Confidentiality, *integrity*, and *availability* express the trinity of properties that underpin virtually all the literature on computer security (Landwehr, 2001). The term computer security ranges over a large expanse of stakeholders, disciplines and theory often mediated by a particular stakeholder perspective. Those perspectives have shaped the security literature and settled into a layered decomposition of topics as indicated in (Bishop, 2003, p. 22). (See Figure 2)

Figure 2 – Hierarchic Model of Security

Much of the early attention to computer security focused on supporting governmental and military requirements for control of information. The seminal work casting security control in the formal, mathematical paradigm is (Bell & LaPadula, 1973). This multi-layered, military security policy addressing a four-tiered classification scheme (i.e. unclassified, confidential, secret, and top secret) received extensive research attention over decades focusing primarily on protecting confidentiality, the non-disclosure of sensitive information (Denning, 1976, McLean, 1985). Non-military or commercial asset concerns lean more toward integrity (the protection of information from loss or corruption). (Lipner, 1982) Denial-of-service (DoS) attacks exemplify attempts to compromise the availability property of security (Wikipedia, 2013). Information security is affected by social-economic factors that extend beyond the business stakeholders who directly interact with information systems. These indirect factors shape the motivations and often the response affecting security threats and influencing policy. In many instances socio-engineering efforts are preferable to expanded protection mechanisms. (Anderson, 2001)

Security modeling efforts reflect the desire to integrate security into design. (Basin, Doser & Lodderstaedt, 2006, Best, Jürjens & Nuseibeh, 2007) International standards for best practice in information security management emanate from ISO/IEC (the International Organisation for Standardization/the International Electrotechnical Commission. (ISO/IEC 27000, 2012)

3. A SPECIAL ONTOLOGY OF COMPUTER SYSTEMS ARCHITECTURE

The computer and information sciences adopt special ontologies to identify a domain of interest within which the elements of relevance may be defined and their relationships explored to demonstrate concepts or theories. The special ontology proposed herein is consistent with the practice in computer science and information science categorizing a domain of concepts (i.e. individuals, attributes, relationships and classes). (See Figure 3 below.) The ontology abstracts the elements of systems architecture pertinent to computer security at the design, implementation, and operations / maintenance layers of the security life cycle. This abstraction focuses on the security properties and relationships that are often obscured by idiosyncratic processor or process architecture implementations. (A

somewhat more formal and concise exposition of the special ontology in its set theoretic form may be found in Appendix A.) (Waguespack, 1975, 1985)

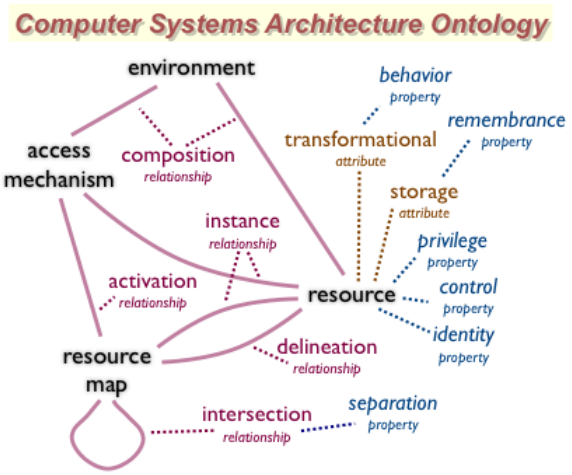


Figure 3 – A Special Ontology of Computer Systems Architecture

The reader is encouraged to envision resources at the hardware instruction set level in the descriptions that follow. Subsequently the exposition will expand that view to encompass all the layers indicated in the hierarchic model of security.

Individuals in this ontology are defined as *environment*, *resource*, *resource map*, and *access mechanism*. The *environment* is the union set of all other elements in a system and in fact defines the universe of interest and discourse as conceived by the stakeholders of the system. *Resources* encompass all the "namable" elements in the *environment* and thus are uniquely distinguishable, the property of *identity*. Two subsets of *resources* define the range and function of *resource* manipulation: 1) a *resource map* that *delineates* a resource subset called an *access scope,* and 2) an *access mechanism* that *activates* access to *resources* in an *access scope*.

Attributes characterize the individuals in the ontology. Storage resources exhibit the attribute of *remembrance*, the capacity to retain state information in the environment. Transformational resources exhibit the attribute of *behavior* operating on instances of storage resource to set, access, and/or modify state information. (The most common form of transformational resource is the machine instruction that interacts with state information sometimes accessing – sometimes modifying the state. In this special ontology state changes result exclusively through the behavior of transformational resources.)

Relationships define the interaction / interdependence of individuals in the special ontology. The environment is the *composition* of all resources. Both access mechanism(s) and resource map(s) are *instances of* resource. The relationship between a resource map and the set of all resources is the *delineation* of a subset of those resource instances, an *access scope*. The relationship between two resource maps defines an *intersection* of their access scopes. An intersection that is the null set defines the *separation* property of that intersection. The application of an *access mechanism* to a pertinent resource map *activates* an *access scope*. (It may be preferable to say an access mechanism *actuates* an access scope since resource *behavior* may be more naturally understood as animation or activity.)

An access scope may include instructions (transformational resources) and/or storage. The quintessential example of transformational resource activation is the application of the mechanism of *instruction execution cycle* to an access scope delineating an otherwise inanimate collection of instruction specification fusing their association into an executing *process*. Storage resources in the access scope remember the instructions, which one is current and the residual state at each instruction's completion.

A common example of storage activation is a virtual memory mechanism. The physical memory pages enumerated in a process's page table delineate its access scope of storage. Swapping activation from one process to another occurs by replacing the page table entries for one process by those of another process. (Saltzer & Schroeder 1975: p. 1286)

Classes distinguish resources whose function in the environment engenders control. Since all "activity" in an environment results from the application of an access mechanism to a resource map, those two specific resources jointly realize the instrument of "animation," the progression of state changes or "execution." Any resource that effects the initiation, sequencing, alteration or suspension of "animation" exhibits the property of *control*. Access to a process's resource map denotes the opportunity of controlling that process.

Detecting attempts to access control resources is a key to managing computer systems security. This is the purpose of the *privilege* property in designating resources susceptible to potentially harmful application.

4. HOST PROCESS ARCHITECTURE

This section describes some common design choices found in contemporary computer systems. They illustrate how the special ontology describes a specific process execution environment. In most computing literature the unit of animation, execution, in a computer system is called a *process* and the environment in which the process executes is called the *host computer, host machine* or *host* (reminiscent of calculating machines). (A generous survey of computer architectures and their properties is found in (Blaauw & Brooks, 1997)).

Mono-processing is the execution of only a single process on a host. Process animation results from the association of access mechanism with resource map – the instruction execution cycle, a collaboration of resources combining state information with transformation. The state information specific to execution is delineated in the *process status vector* (psv) that indicates residual state information: (e.g. conditional comparison flags, error conditions, and what the next instruction to execute should be, etc.). Each instruction execution occurs in the state of the machine resulting from the previous instruction's execution, hence a "cycle." The psv also delineates the subset of storage resources accessible by the process. The psv realizes the special ontology's resource map delineating both the transformational and storage resources accessible by the process. (As indicated in the set theoretic ontology description in Appendix A, it is also common for the resource map to be realized as two discrete elements: one focusing on instruction execution and a second mapping the accessible storage space.) In the common case where external resources exist (i.e. input/output devices, networking), their design may be treated as additional namable resources. (An example of such a connection is the mapping of I/O interfaces as storage locations in the DEC PDP11 UNIBUS configuration.) (Blaauw & Brooks, 1997: p. 967)

Multiprogramming occurs in the presence of more than a single process residing on one host. Although there are multiple processes, there may exist only a single instruction execution cycle that is shared (by multiplexing) among them. This processing protocol requires a managing process usually called an operating system (OS) that includes supervisory and service components arranged as a collection of agent processes. These agents manage the association of the instruction execution cycle with the various processes one at a time. (The protocol for delegating execution among the various processes is called process scheduling and may be based upon various priority schemes to manage the progress of the individual processes respectively.) The agents may themselves be processes – each with its own resource map.

What distinguishes a process that is an OS is the privilege of an access scope including any and all host resources. Where the resource map of a non-OS process is denoted psv for process state vector, the resource map of an OS process is denoted csv, control state vector. The distinction highlights the OS as *in control* of the entire host.

A control resource is one that permits a process to assign the instruction execution cycle to a particular process or to access/modify the resource map of a process, including itself. The most common mechanism for enforcing the OS's prerogatives (privileges) is the *interrupt* that extends the behavior of the instruction execution cycle and permits the OS to gain access to the instruction execution cycle at will.

The normal course of the instruction execution cycle proceeds with each succeeding instruction determined by the current process's psv (i.e. the process's execution continues without interruption.) An *interrupt* mechanism is the detection and response to a condition signaling the need for a departure from the current sequence of execution (suspending the current process) and designating execution of the next instruction in a different process (activating another process). In a nominal OS design the interrupt would cause the sequence of instruction execution to transfer to an agent of the OS called the *interrupt handler*. Once an interrupt handler is activated its instructions determine the system's ensuing behavior. The result is a transfer of the sequence of instruction execution from the interrupted process to another, a *process swap*.

Interrupts are designed to support a variety of supervisory tasks. Interrupts may occur due to: external signals (i.e. input/output connections), execution exceptions (i.e. erroneous instruction specifications), attempts to access resources designated as privileged, access exceptions (i.e. attempts to access resources not delineated in the process's resource map), or solely to relinquish the instruction execution cycle. The interrupt mechanism's design incorporates state information (i.e. psv) to designate the precise process swap behavior (e.g. what state information is to be set in the suspended process). Hence, the psv is (in itself) a control resource. Whether or not interrupts are enabled or suppressed in the current process and which process is to be activated in the process swap is indicated in the csv, hence these are controlled by the OS.

Multiprocessing differs from the multiprogramming protocol in that more than one instruction execution cycle resource is available to associate with the processes present on the host. Multiple processes may concurrently execute their instructions. Each instruction execution cycle must manage a separate interrupt protocol. Individual process management in the presence of multiprocessing is not significantly different from multiprogramming unless concurrently executing processes are allowed to communicate and/or share resources. In which case inter-process communication and sharing require addition supervisory services that monitor and mediate process interactions.

5. OS AND PROCESS SECURITY

Monoprocessing is the least complex form of process management. It depends exclusively on the process's fidelity of programming to its specifications. The primary threat is programming error. (A monoprocessing system often includes OS services to offload I/O or job-scheduling tasks from individual processes. It earns trust from the quality assurance it receives. Regardless, any error in programming can compromise the system.)

Multiprogramming requires an OS that shares its attention with more than one process. Where the OS protects itself from the user process, it now must also protect user processes from the misbehavior of one another. *Encroachment* is the unauthorized access of one process's resources by another. Encroachment occurs due to errors/malice among the cohabitant processes.

Although the added instruction execution capabilities in a multiprocessing environment make resource management and coordination among processes more complicated, the basic principles of process closely resemble multiprogramming.

Separation is the major protection mechanism in a multiprogramming environment – to prevent the undesirable behavior of one process from causing the failure of or undue interference with the other processes. *Separation* is achieved by: 1) devising resource maps that delineate resources according to process, and 2) enforcing *complete mediation* where each access mechanism enforces adherence to its delineated resources by prohibiting unsanctioned access and, usually, by raising an interrupt condition when an attempt occurs (Bishop, 2003, p. 345). Effective separation eliminates the risk of encroachment.

Privilege denotes the supervisory authority of an OS to control the resources of individual processes. The supervisory functions of the OS (the *kernel* or *nucleus*) exercise *control* over every process in the system. Complete mediation also controls the number of consecutive instructions that a process may execute (based on either a real or virtual "clock," as prescribed in host csv). That prohibits any process from monopolizing the instruction execution (e.g. the infinite loop!). The OS kernel protects itself by managing the csv of the host to retain *control* over every process while at the same time it remains *separated* from all processes on the host except itself – a *secure operating system*.

6. VIRTUAL MACHINES IN THE CLOUD

The monoprocessing environment is the least complex and therefore, the most easily quality assured – trustworthy. That explains the growing preference for multiprocessing virtual machines as the security framework for cloud computing. (Rosenblum & Garfinkel, 2005)

A virtual machine is an execution environment in which the process that executes cannot determine if it resides on a (physical) host machine or is a "guest" on a simulation of that host. Each guest process can be treated as the sole process on that virtual machine. A layer of supervisory software, the virtual machine

monitor (VMM), manages the execution environment of each guest by configuring the resource maps and the interrupt behavior of the access mechanisms to intercept any guest attempts to directly access control resources. When a guest needs access to resources that might compromise "safety," that access must be mediated by the VMM through a *reference monitor* that simulates the process's resource access by virtualizing that resource while protecting the VMM's control of the host. These protocols rely on *control* and *access mechanism* design. (Smith, J.E. & Nair, R., 2005)

Self-virtualizable: When a virtual machine appears identical to its underlying host, it can provide direct access to the exact same instruction set and access mechanisms of that host. We call this host architecture *self-virtualizable* because it can provide a complete replica of itself for guest process(s). (Waguespack, 1985) A guest process may execute at the same level of efficiency as it would if it were the sole process on the underlying host except for those resources that must be virtualized to maintain the VMM's control.

Host hardware design decisions make virtualization straightforward or complex. (Garfinkel & Rosenblum, 2005) Process control protocols prior to the advent of virtual machines focused primarily on facilitating a host-based OS's capacity to maintain control. Design decisions for processor hardware architecture did not always support *complete mediation*. When host instruction set designs include instructions that access control resources but are not designated as privileged they do not raise interrupt conditions even when executed in a process without privilege. There are two options that support virtual machines on these hosts. In one approach the VMM pre-scans all the process's instructions before execution and replaces "unsafe" instruction sequences with "safe" instructions – usually including instructions that do cause interrupts to allow the VMM to intercede and simulate the indicated service. Alternatively, the VMM simulates all instructions running on the virtual machine; an approach that results in a dramatic reduction in effective execution speed since simulation is orders of magnitude slower than direct host hardware execution. (Complete simulation is the common approach used to execute Java code. (Lindholm, Yellin, Bracha, & Buckley, 2013).)

The most efficient virtual machine realization requires that all control resources include the privilege property with appropriate interrupt conditions to allow a VMM to enforce complete mediation. Secondly all resource access attempts that would otherwise be directed to "hardware resources" (i.e. I/O and communication devices) need to be intercepted allowing the VMM to virtualize those resources through reference monitors. This combination of intercession provides the maximum guest performance with the minimum of VMM overhead.

Multiple OS's: The earliest implementations of VMM technology supported guest processes that themselves were OS's, guest OS's. The ability to run multiple versions of OS concurrently on the same physical host provided flexibility and cost savings. In the description of process swapping and control above the key resource designating process activation is the process state vector. (Waguespack, 1985) The privilege of designating which psv(s) is active determines control over the entire environment. Each virtual machine environment is a simulation of the physical host and thus as these guest OS's manage the processes that reside on them the OS's believe they are controlling the host. But their "control" is virtualized by the VMM with virtualized csv access. The VMM manages the VM's as its processes retaining control over all the physical host resources. This permits a multi-layered arrangement of virtual machines some supporting mono-processing, others supporting multiprogramming OS's and finally some supporting replicas of the VMM itself telescoping the illusion of host environment layer after layer limited only by physical of performance and resource virtualization capabilities.

Security through virtualization: Virtual machines provide a convenient architectural foundation for security. Each guest process (or virtual machine) resides in a naturally separated execution environment. Complete mediation through VM instruction execution and virtualized resource access facilitates connectivity beyond the boundaries of the virtual machine, but only through contractually defined protocols, VMM mediated services. (Garfinkel & Rosenblum, 2005)

Although initially virtual machines were intended to support concurrently executing OS's on a single host, guest VM's may also be

VMM's in their own right. Such an arrangement permits a hierarchically encapsulated progression of secure execution environments. Virtualization enables sharing and protection protocols administered through complete mediation that both enables and structures security. VM architecture is the basis of cloud computing. VM's make possible the protections and scalability of system security in the cloud. (DeKeyrel, Waldbusser & Jones, 2012)

7. CONCLUSION

Security is an essential topic in IS education today. But, finding room for it in an already crowded curriculum is a challenge. The "shrinking real estate" for technical content in IS curricula will continue to require compactly designed primers (similar to this one on computer security structures) if our IS graduates will have any practical grounding to discern credible information systems capabilities and performance potential.

This primer addresses the key architectural security issues concisely. The primer is a useful pedagogical foundation for computer, network and Internet security discussions. It is suitable for embedding in a generic systems, networking or business process course for upper level undergraduates or graduate IS or MBA students. Teachers may use it as a survey, tutorial or as an outline for a student research project. It is appropriate for either an individual or a comparative system security study. The special ontology is applicable to the full range of computing architectures from Turing and von Neumann through the DEC, Cray, IBM, Motorola and Intel generations – as well as architectures of loosely coupled processors (e.g. the World Wide Web). (Turing, 1936, Bell & Newell, 1971, Blaauw & Brooks, 1997)

8. ACKNOWLEDGEMENTS

Thanks for helpful comments from the referees. Special thanks are due my colleagues David Yates and Bill Schiano at Bentley University for their insightful discussions and comments on these ideas. And thanks to the students who have labored through the development of this learning unit.

9. REFERENCES

Anderson, R. (2001), "Why Information Security is Hard – An Economic Perspective," Proceedings of the 17th Computer Security Applications Conference, pp. 358-365.

Basin, D., Doser, J. & Lodderstedt, T., (2006). "Model Driven Security: From UML Models to Access Control Infrastructures," *ACM Transactions on Software Engineering and Methodology*, 15(1), pp. 39-91.

Bell, C.G., & Newell, A., (1971). Computer Structures: Readings and Examples, McGraw-Hill, New York, NY, pp. 92-119.

Bell, D.E., & LaPadula, L.J. (1973). "Secure Computer Systems: Mathematical Foundations," Technical Report Mitre-2547, Vol. 1, Bedford, MA, USA.

Best, B., Jürjens, J. & Nuseibeh, B. (2007). "Model-Based Security Engineering of Distributed Information Systems Using UMLsec," ICSE '07 Proceedings, pp. 581-590.

Bishop, M. (2003), Computer Security: Art and Science, Addison-Wesley, Boston, USA.

Blaauw, G.A., & Brooks, F.P., Jr., (1997). Computer Architecture: Concepts and Evolution, Addison-Wesley, Reading, MA, USA.

DeKeyrel, M., Waldbusser, M. & Jones, A. (2012). "Secure virtual machine instances in the cloud: security considerations when provisioning in IBM SmartCloud Enterprise, http://www.ibm.com/developerworks/cloud/library/cl-cloudvmsecurityrisks/ (retrieved 5/23/2013).

Denning, D.E. (1976). "A Lattice Model of Secure Information Flow," *Communications of the ACM*, 19(5), pp. 236-243.

Garfinkel T., & Rosenblum M., (2005). "When Virtual is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments," 10th Workshop on Hot Topics in Operating Systems.

ISO/IEC (2012). "ISO/IEC 27000:2012 2ed," ISO/IEC, Geneva, Switzerland, www.ISO.org.

Landwehr, C.E. (2001). "Computer security," *International Journal on Information Security,* Vol. 1, No. 1, pp. 3-13.

Lindholm, T., Yellin, F., Bracha, G., & Buckley, A. (2013). The Java® Virtual Machine Specification: Java SE 7 Edition, http://docs.oracle.com/javase/specs/jvms/se7/html/index.html, (accessed 5-23-2013).

Lipner, S. (1982). "Non-Discretionary Controls for Commercial Applications," 5th Symposium on Operating Systems Principles, pp. 192-196.

McLean, J. (1985). "A Comment on the 'Basic Security Theorem' of Bell and LaPadula," *Information Processing Letters*, 20(2), pp. 67-70.

McCumber, J.R. (1991). "Information Systems Security: A Comprehensive Model," Proceedings of the 14th National Computer Security Conference (Annex to NSTISSI No. 4011).

Topi, H., Valacich, J.S., Wright, R.T., Kaiser, K.M., Nunamaker, J.F. Jr., Sipior, J.C., & de Vreede, G.J. (eds.) (2010). IS2010: Curriculum Guidelines for Undergraduate Degree Programs in Information Systems, Association for Computing Machinery (ACM) & Association for Information Systems (AIS).

Rosenblum, M. & Garfinkel, T. (2005). "Virtual Machine Monitors: Current Technology and Future Trends," *IEEE Computer*, 38(5) pp. 39-47.

Saltzer, J.H. & Schroeder, M.D., (1975). "The Protection of Information in Computer Systems," Proceedings of the IEEE, 63(9) pp. 1278-1308.

Smith, J.E. & Nair, R. (2005). "The Architecture of Virtual Machines," *IEEE Computer*, 38(5), pp. 32-38.

Turing, A.M. (1936). "On computable numbers, with an application to the Entscheidungsproblem," Proceedings of the London Mathematical Society, Series 2, vol. 42 pp. 230-65, correction ibid. 43 (1937) pp. 544-6.

Waguespack, L. (1975). Virtual Machine Multiprogramming and Security, Doctoral Dissertation, Computer Science Department, University of Louisiana at Lafayette.

Waguespack, L. (1985). "A Structural Computer Resource Model for Teaching Computer Organization," Proceedings of SIGCSE'85, pp. 63-67.

Wikipedia (2013). "Denial of Service," https://en.wikipedia.org/wiki/Denial-of-service_attack, (accessed 25 July 2013.)

Appendix A – Set Theoretic Representation of the Special Ontology (Waguespack, 1975, 1985)

E → environment "→" reads "…is defined as…"

S → storage resources (all state information)

T → transformational resources (all computation capable of modifying the state)

E → {T, S} (all transformational and storage resources)

M → a machine, {Tm, Sm}, (i.e. M ⊆ E)

tb → the "map" delineating an access scope of transformation resources

[] → a mechanism (function) to access a resource access scope

T[]tb → a resource subset of T defined by tb (i.e. T[]tb ⊆ T)

sb → the "map" delineating an access scope of memory/storage resources

S[]sb → a partition of the resource S defined by sb (i.e. S[]sb ⊆ S)

{tb, sb} → {eb}; environment base: a composite access scope

E[]eb = {T[]tb, S[]sb} a program is a specification of transformational and storage resources

psw → the process status word of a program (process not yet initiated)

psv → the process status vector (word) as "idle" or suspended process

⧖ → the transformational resource that executes instructions in a process

⧖ (psv) → the process state word of an "active" program under execution

{T[]tb, S[]sb, psw} → an "idle" program (not yet initiated)

{T[]tb, S[]sb, psv} → an "idle" process (having been initiated but not currently active)

{T[]tb, S[]sb, ⧖ (psv)} → an "active" process currently executing

CSV → the control state vector enabling the control of a machine

Sm → all the memory/storage of M

Sm[]uj → the storage of M accessible by user j

Sm = {Sm[]csv, Sm[]u1, Sm[]u2, Sm[]u3, ... }

Sm[]csv → the storage that contains/accesses the CSV of M

Sm[]psv → the memory/storage that covers/accesses the PSV of a process

Sm[]u → {Sm[]csv ∩ Sm[]u = ∅} the memory/storage not including the CSV of M

Tm → instruction set of M

Tm[]c → instruction set of M that can control M (also named C)

Tm[]u → { Tm - Tm[]c} (non-control) "user" instruction set of M (also named U)

E[]csv → {Sm[]csv, Tm[]c} the control state vector of E that designates "control" of E

{ E[]csv , ⧖ (csv)} → the "active" process that controls the entire environment, (e.g. VMM)

Computer Systems Architecture and Security

Without a Language or Syntax!

What is computer systems security world all about?

Special Ontology of Computer Systems Architecture

This ontology is consistent with the practice in computer science and information science categorizing a domain of concepts (i.e. individuals, attributes, relationships and classes). This ontology is based upon a set-theoretic model of computer system resources defining the architectural elements of information systems characterized by executable program code and accessible storage resources. The ontology identifies the elements of a computerized information system and their relationships relevant to the definition and protection of resource security – *the state of being free from danger or threat.*

1. Individuals

A ***resource*** is a distinguishable element within the domain of discourse concerned with security. An ***environment*** is a stakeholder prescribed domain of interest defining the context and relevant elements of security concern; it is the set of all resources under consideration. A ***resource map*** is a specification of some subset of the *environment*. An ***access mechanism*** is an executive agent that effectuates *behavior* and/or mediates access to *resources* specified by a *resource map*.

2. Attributes

Each *resource* in an *environment* is uniquely distinguishable, the ***identity*** property. Each ***resource*** is either ***storage*** (the object of some behavior) or ***transformational*** (the instrument of some behavior). *Storage* resources exhibit the property of ***remembrance*** as they are associated with a value which may be retrieved or modified only through access to the *storage resource* itself. A *transformational resource* exhibits the property of ***behavior*** that acts upon a *storage resource* to modify its value (effect a change of "state").

3. Relationships

Relationships exist on two dimensions: behavioral and structural.

3.1. Behavioral Relationships

3.1.1. Activation

The application of an *access mechanism* to a *resource map* is an ***activation,*** resulting in animation within an *environment;* realizing behavior and any corresponding state changes. Any and all access to *resources* is realized through the application of an *access mechanism* to a *resource map*. The animation (or activity) realized by the composition of *access mechanism* to *resource map* is called a ***process*** or a ***thread.***

3.2. Structural Relationships

3.2.1. Composition

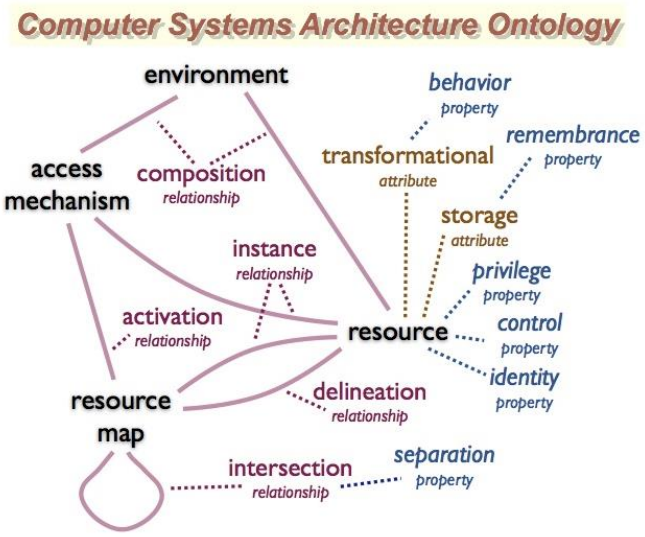
An *environment* is the composition, the union, of all *resources*, both *storage and transformational resources* (including all *access mechanism(s)*).

3.2.2. Instance

A *resource map* is an instance of *storage resource*. *Access mechanism* is an instance of *transformational resource* effectuating *behavior*.

3.2.3. Delineation

A *resource map* ***delineates*** a subset of resources called an ***access scope***; thus determining what storage resources are accessible and/or what behaviors are possible through *transformational resources*.



3.2.4. Intersection

Access scopes ***intersect*** if they share access to the same *resource*. The ***separation*** property denotes the absence of the *intersection* of accessibility. *Process* X is said to be ***separated from*** *Process* Y if the *resource map* of *Process* X is not delineated in the *access scope* of *Process* Y. If the *intersection* of two *access scopes* is null then they are said to be mutually ***separated***.

4. Classes

Resources enabling *activation* compose the class of ***control*** resources. All *resources* fall into either the class of ***control*** or ***non-control*** *resources*.

5. Security Apparatus

In the context of computerized systems, security is the state of being free from danger or threat ascribed to a stakeholder(s) whose interests are invested in the system behavior. That vestiture takes the form of agency – behavior within the system carried out on behalf of a stakeholder(s). Security breach includes disruption, incursion, or the hijacking of resources.

5.1. Process

A ***process*** results from *activating* the combination of the *transformational* and *storage resources* in its *access scope*. A *process* realizes behavior as the agent of a stakeholder(s). The ongoing application of host *access mechanism*(s) to a *process*'s *resource map*(s) denotes an ***active process***; withdrawing the host *access mechanism*(s) yields a ***suspended process***.

5.2. Process Control

Process Control refers to regulating *process* behavior (initiating, sequencing, suspending, and/or terminating behavior). *Process control* is effected through the manipulation of *access mechanisms* and/or a *process's resource map*. Process X is said to *control* Process Y if the *resource map* of Process Y is in the *access scope* of Process X. *Separation* precludes *process control*.

5.3. Encroachment

Encroachment (also called ***access violation***) is an access within a process's *access scope* by unauthorized *process*. *Process* X *encroaches* on *Process* Y if it accesses *Process* Y's *access scope* without *Process* Y's intentional cooperation.

5.4. Secure Process

Process X i***s protected from*** *Process* Y if *Process* X *is separated from Process* Y. A ***secure process*** is a *process* that *is separated from* all other *processes* in an *environment* and thus is free from danger of *encroachment.*

6. Operating System Principles

A ***host*** (sometime called ***machine***) is a computer architecture upon which a *process*(s) executes. An ***operating system*** is a collection of *processes* that *controls* and extends the native *resources* of a *host* architecture to facilitate one or more *processes* other its own. The *host* architecture may be physical or virtual.

6.1. Nucleus

The ***nucleus*** (sometimes called a ***kernel***) manages the *host resources* specific to *control* of the *host* realized as two functions: *interrupt handlers* (managing the instruction execution cycle) and *reference monitors* (mediating access to resources).

6.2. Secure Operating System

A *secure operating system* is a *secure process*(s) on a host with *control* of all *processes* on that *host* including itself.

6.3. Virtual Machine

A *virtual machine* is an execution environment in which any *process* that executes cannot determine if it resides on a (physical) *host* or on a simulation of that *host.*

6.4. Virtual Machine Monitor

A *virtual machine monitor* is a *nucleus* supporting *processes* that are themselves "guest" *virtual machines*. Each VM may either a single *process* or *operating system* managing multiple *processes*.

6.5. Self-Virtualizable Host

A *self-virtualizable host* supports *complete mediation* in the application of *access mechanisms* to *control resources* thus enabling a VMM to maintain its *control* as a *secure operating systems* and intercede by mediating "guest" access to virtual resources while allowing direct, benign access to native resources. A *self-virtualizable host* offers *virtual machine* efficiency comparable to that of the *host* for all unmediated *process* behavior.