Proof of Separability A Verification Technique for a Class of Security Kernels

Introduction

- Kernelized systems: isolate the functions essential to the system's security inside a **security kernel.**
  - Dual-Mode and Multimode Operation-OS Review: (Silberschatz, Abraham, Peter Baer Galvin, and Greg Gagne. Operating System Concepts. 9th ed. N.p.: Wiley, n.d. Print.)
    - Two modes of operation: **user mode** and **kernel mode**
    - Mode bit - determines mode: kernel (0) or user (1)

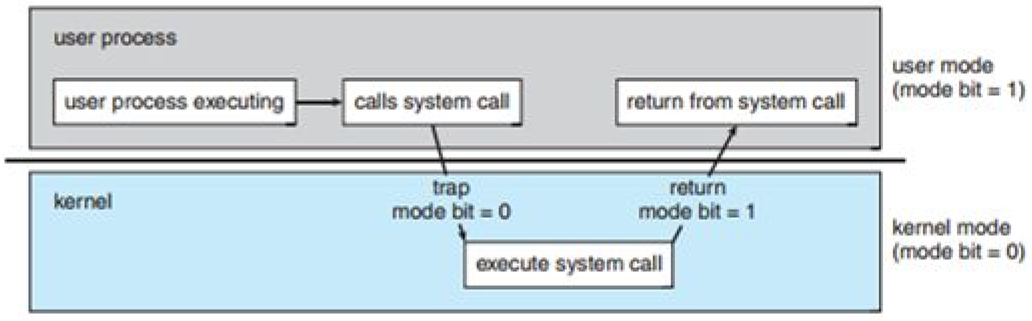


Figure 1.10 Transition from user to kernel mode.

    -
    - Privileged instructions - instructions that may cause harm. Only executed in kernel mode.
    - Privilege levels are an alternative to modes.
    - Hardware protection detects errors that violate modes.
  - Kernelized Systems Referenced in Paper
    - UCLA Secure Unix* (https://www.computer.org/csdl/proceedings/afips/1979/5087/00/50870355.pdf)

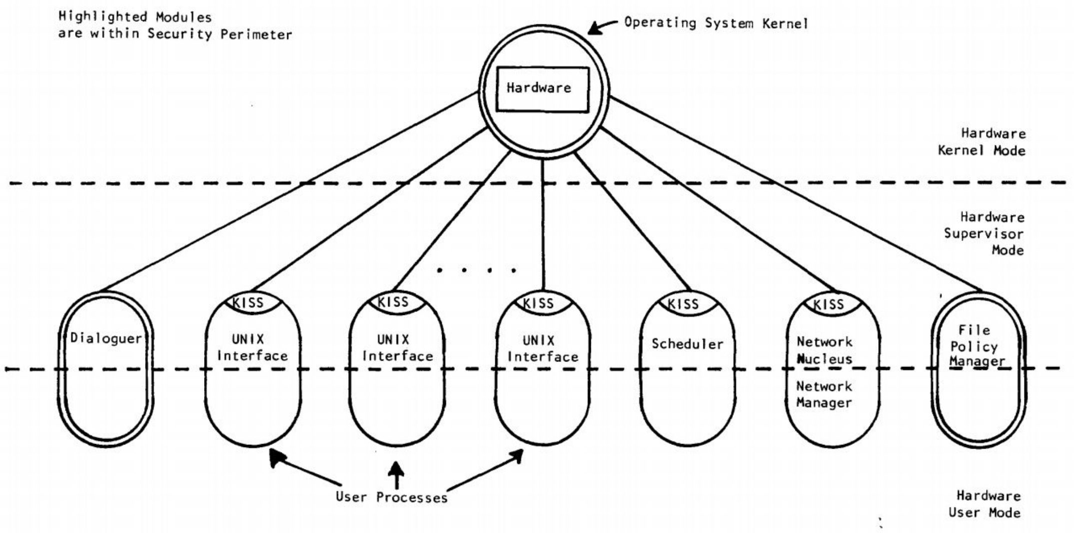


Figure 1—UCLA Secure Unix architecture.

        - KSOS (https://www.computer.org/csdl/proceedings/afips/1979/5087/00/50870345.pdf)

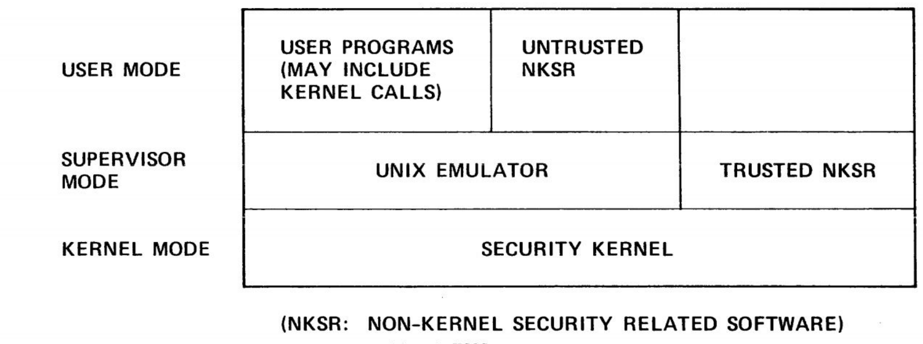
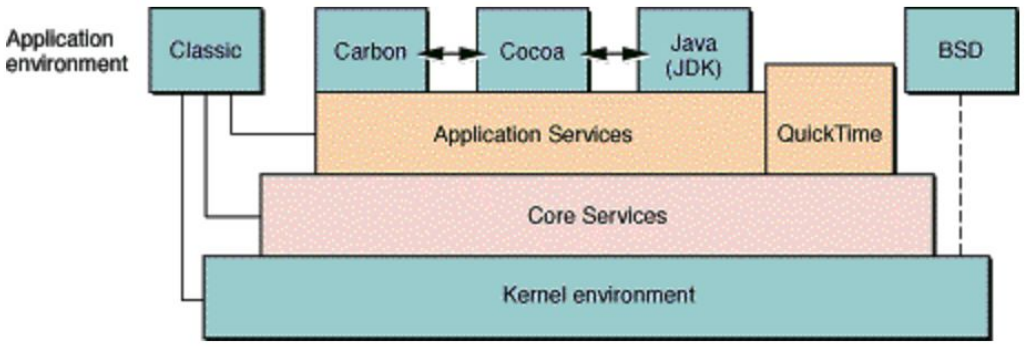


Figure 1—KSOS system structure.

    - Modern example: OSX architecture (https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/Architecture/Architecture.html#//apple_ref/doc/uid/TP30000905-CH1g-CACDAEDC)

Figure 3-1 OS X architecture



        - "In OS X, however, the kernel environment contains much more than the Mach kernel itself. The OS X kernel environment includes the Mach kernel, BSD, the I/O Kit, file systems, and networking components. These are often referred to collectively as the kernel. Each of these components is described briefly in the following sections. For further details, refer to the specific component chapters or to the reference material listed in the bibliography."
- Proposes type of distributed system that uses a kernel called **separation kernel** that masks interactions of different users on the same machine. (It still acts like a single-user machine.)
    - Each system component (user) has a virtual machine (called a **regime**) dedicated to it.
    - Components communicate through message passing (theoretically, but that's not accounted for here).
- Main goal: prove the system behaves as if it were composed of several totally separate machines AKA "Proof of Separability"
- He defines this proof by specifying "Secure Isolation"

Specifying Secure Isolation

- Assumptions made in proof:
    - Extreme isolation: these specifications don't allow for communication between users

    - Systems must have some means of making progress
    - Must consume inputs and produce outputs
    - Inputs are presented to the system only once, right at the start
    - More than one user shares the machine
- Definitions:
    - A system or machine $M = (S,\ I,\ O,\ Nextstate,\ Input,\ Output)$ where:
        - S = a finite, nonempty set of system states
        - I = a finite, nonempty set of inputs ($i \in I$ includes user inputs and the "program")
        - O = a finite, nonempty set of outputs
        - $Nextstate : S \to S$, where $Nextstate(s)$ is the successor of state $s \in S$
        - $Input :\ I \to S$
        - $Output : S \to O,$ where $Output(s)$ is the visible aspect of $s \in S$
    - The users of a system are members of a set $C = \{1, 2, ..., m\}$ of "colors". (Our machine is called a $C-$ shared machine)
    - The computation of an input $i \in I$ is an infinite sequence of the resulting states from inputting i. $Computation(i) = <s_0, s_1, ..., s_n, ...>$, where $s_0 = Input(i)$ and $s_{j+1} = Nextstate(s_j)$, $\forall\ j \geq 0$.
    - The result of a computation, given an input $i \in I$, is the sequence of outputs observable to an observer.
      $Result(i) = <Output(s_o),\ Output(s_1), ..., Output(s_n), ...>$
      As a "notational convenience", the author also defines: $Result(i) = Output(Computation(i)).$
    - The extraction of an input ($Extract(c, i)$) returns the components of some $i \in I$ that are c -colored (belonging to some user c) for some $c \in C$. Similarly, $Extract(c, o)$ returns the components of some $o \in O$ that are c-colored for some $c \in C.$
    - A condensed sequence eliminates any consecutive repeats. (i.e. $Condense(1, 1, 2, 3, 3, 1, 4, 3) = 1, 2, 3, 1, 4, 3$) This is useful because $Condense(Extract(c, Result(i)))$ eliminates any repeat outputs caused by waiting for the system while it services other users.
    - $X \cong Y$ if and only if either $X = Y$ or the shorter of X and Y is an initial subsequence (predecessor) of the other. This is useful because if $Extract(c, i) = Extract(c, j),$ but $Computation(i)$ faces a denial of service, say $result_1 = Condense(Extract(c, Result(i)))$ and $result_2 = Condense(Extract(c, Result(j)))$, then $result_1 \neq result_2,$ but $result_1$ is a predecessor of $result_2,$ so $result_1 \cong result_2.$
- Precise specification of **Secure Isolation**:
  $\forall c \in C, \forall\ i, j \in I,\ Extract(c, i) = Extract(c, j) \Rightarrow Condense(Extract(c, Result(i))) \cong Condense(Extract(c, Result(j)))$

Verifying Secure Isolation

- Establish this idea of secure isolation by proving that a user c is unable to distinguish the behaviour of a $C-$ shared machine and a private machine. For these purposes, the author defines an M -compatible private machine.
- A $C-$ shared machine can be defined as follows: $M = (S,\ I,\ O,\ Nextstate,\ Input,\ Output)$ is a $C-$ shared machine where $I = I^1 \times I^2 \times ... \times I^n$ and $O = O^1 \times O^2 \times ... \times O^n$ and $c \in C.$
- A Private Machine for $c \in C$ can be defined as follows: $M^c = (S^c,\ I^c,\ O^c,\ Nextstate^c,\ Input^c,\ Output^c)$ with the functions $Computation^c$ and $Result^c$
- **Definition 1 of an *M*-Compatible Private Machine:**
  Given a $C-$ shared machine $M,$ and a private machine M^c , then M^c is an $M-$compatible private

machine for c if $\forall i \in I,\ Condense(Extract(c, Result(i))) \cong Condense(Result^c(Extract(c, i)))$.
In English: The result when an $M-$compatible private machine takes the c component of an input must equal the c component produced by that same input on the $C-$shared machine.

- **Definition 2 of an *M*-Compatible Private Machine (Theorem 2):**
  (He introduced this definition because it's easier to derive a proof from it.)
  Given a $C-$shared machine M, a private machine M^c, and a function $Colour : S \rightarrow C$ that returns the colour associated with who is being serviced in that state, then M^c is an $M-$compatible private machine if $\exists\, \Phi^c : S \rightarrow S^c$ such that $\forall i \in I,\ \forall\ s\ \in S$:
  1) $Colour(s)\ =\ c\ \Rightarrow \Phi^c(Nextstate(s))\ = Nextstate^c(\Phi^c(s))$. If c is being serviced in state s in M, then the state in S^c corresponding to the successor of $s \in S$ is equal to the successor in S^c of the state in S^c corresponding to s. (In other words, there's a corresponding transition function in S^c)
  2) $Colour(s)\ \neq\ c\ \Rightarrow \Phi^c(Nextstate(s))\ = \Phi^c(s)$. If c is not being serviced in state s in M, then the state in S^c corresponding to the successor of $s \in S$ is equal to the state in S^c corresponding to s. (In other words, no transition happens in S^c)
  3) $\Phi^c(Input(i))\ =\ Input^c(Extract(c, i)$). Defines the mapping of the input.
  4) $Output^c(\Phi^c(s)) = Extract(c, output(s))$. The outputs should match at each state.

Proof of Separability

- A proof could be derived by constructing an $M-$compatible private machine M^c that fits these specifications. He decides that this is too arduous, so he suggests only slightly constructing an M^c by specifying operations of the private machine, and then constraining the behaviour of a $C-$shared machine so that the existence of an $M-$compatible private machine would be guaranteed. (Basically, the usefulness of this theorem is that this theorem relies on defining Φ without also having to define M^c.)
- Specifying operations:
  - In a real machine: first, an "operation" is selected by some "control mechanism", and then it is "executed" to yield the next state.
  - Operation: $Ops \subseteq S \rightarrow S$ is a set of operations, where each operation is a total function on states.
  - Control Mechanism: $Nextop$: $S \rightarrow Ops$
  - Execution: $Nextstate(s) = Nextop(s)(s)$
- <u>**Theorem 3:**</u>
  Let $M = (S,\ I,\ O,\ Ops,\ Nextop,\ Input,\ Output)$ be our new $C-$shared machine. Suppose there exist sets S^c of states and $Ops^c \subseteq S^c \rightarrow S^c$ of operations on S^c together with abstraction functions:
  $\Phi^c : S \rightarrow S^c$ and $Abop^c$: $Ops \rightarrow Ops^c$ which satisfy $\forall c \in C,\ \forall\ s, s' \in S, \forall op \in Ops,\ \forall i, i' \in I$:
  (Explanations are told in terms of a "red" user)
  1) $Colour(s)\ =\ c\ \Rightarrow \Phi^c(op(s)) = Abop^c(op)(\Phi^c(s))$. When an operation is executed on behalf of the red user, the effects which that user perceives must be capable of complete description in terms of the objects known to him.
  2) $Colour(s)\ \neq\ c\ \Rightarrow \Phi^c(op(s)) = \Phi^c(s)$. When an operation is executed on behalf of the red user, other users should perceive no effects at all.
  3) $Extract(c, i)\ =\ Extract(c, i') \Rightarrow \Phi^c(Input(i)) = \Phi^c(Input(i'))$. Two equal c-components in an input map to the same state in S^c. Only red I/O devices may affect the state perceived by the red user.

4) $\Phi^c(s) = \Phi^c(s') \Rightarrow Extract(c, Output(s)) = Extract(c, Output(s'))$. Two equal states in S^c map to the same c-components in an output in S. Red I/O devices must not be able to perceive differences between states which the red user perceives as identical.

5) $Colour(s) = Colour(s') = c$ and $\Phi^c(s) = \Phi^c(s') \Rightarrow Nextops(s) = Nextops(s')$. If c is the colour currently being serviced and they are equal in S^c then their corresponding operations are equivalent. The selection of the next operation to be executed on behalf of the red user must only depend on the state of his regime.

- Accounting for Input/Output: $Input$ changes so $Input : S \times I \rightarrow S$. Prior to the $Nextstate$ function at each step, the machine makes a nondeterministic choice whether or not to apply the $Input$ function. Replace part 3) in the theorem with:
  3a) $Extract(c, i) = Extract(c, i') \Rightarrow \Phi^c(Input(s, i)) = \Phi^c(Input(s, i'))$
  3b) $\Phi^c(s) = \Phi^c(s') \Rightarrow \Phi^c(Input(s, i)) = \Phi^c(Input(s', i))$. Two equal states in S^c will transition to the same state in S^c given the same input. I/O devices must not be able to cause dissimilar behavior to be exhibited by states which the red user perceives as identical.
- Further work: relax second condition in Proof of Separability to include communication between users.