Composition of Critical Properties
Lessons From Other Fields

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Composition

- Just a fancy word for modular construction
- We build systems from components
- Some of which are standard, some bespoke
- All tied together with "glue logic"
- We do this for economy, efficiency, quality
  - Good components are a reusable asset
- Our reasoning about the system is based on what we know about the components and the way they are put together
- What's wrong with that?

Modern Automobiles

- Engine and its control system were traditionally developed separately from transmission and its control system
- But they actually have to work together
  - e.g., transmission tells engine to retard ignition while it changes gear
- Given the two components, it can take six months to integrate them
  - Not due to intended interactions such as the one above
  - But unanticipated consequences of interactions
  - And low-level details, like bus timing
- Delayed introduction of automated cruise, lane monitoring/following, automated parking, integration of powertrain with steering and suspension
- Reputations of some luxury manufacturers damaged

Composition Is Easy, No It's Hard

- It's easy when there is little interaction among components
  - e.g., 1960s automobiles, OB1
- It gets harder the more they interact
- Emergent behavior
  - Behavior of the system not found in any component
  - That's why we build systems—e.g., medical device PnP
- But then there's unanticipated, undesired emergent behavior
- In the limit, we get Normal Accidents (Perrow)
  - High interactive complexity
  - Tight coupling
- Challenge is to eliminate bad emergence, keep the good

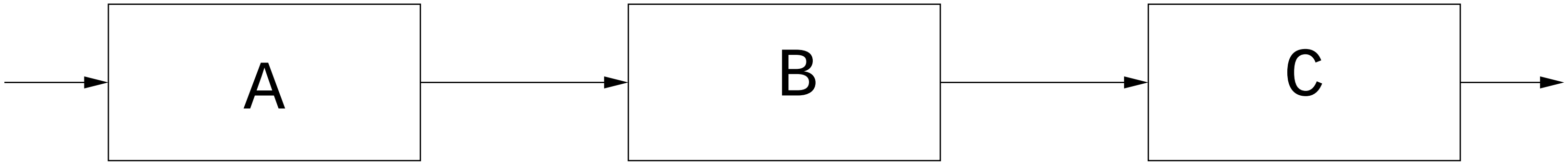
System Properties

- Properties like safety, security, real-time guarantees
- These are properties of the whole system
    - e.g., where's the component that makes an airplane safe?
    - But a single component can easily make it unsafe

  i.e., these are emergent properties
- That's why the FAA certifies only airplanes and engines
    - OK, propellers too
    - Components certified only as part of an airplane or engine
    - Because you need to examine them in their context of interaction
- But this is becoming ruinously expensive, even infeasible
- Our goal is compositional development and assurance

It's About Interactions

- We must have no unintended interactions
- But in most systems almost every component can affect every other indirectly
- So we need to focus on the direct interactions



- And the nature (channels) of interaction
- And we need to be concerned about unintended consequences of intended interaction channels
- Particularly in the presence of faults, malice

Lessons From Other Fields

- The most sophisticated treatment is in embedded systems
  - They need real time, fault tolerance, safety
- The have the following concepts
  - Error-propagation boundaries
  - Elementary and composite interfaces
  - Fault-containment regions
  - Composability
- And from integrated modular avionics (IMA), we get
  - Partitioning
  - Determinism, time-triggered scheduling
- And from EU safety certification, we get
  - Argument-based assurance cases

Error-Propagation Boundaries

- Errors in a component should be detectable at its interface, before they propagate to other components
- Two kinds of errors: in control and in data
- Interfaces move data and use control (e.g., a protocol) to accomplish it
- Control errors are particularly destructive in real-time systems because they affect workload in the victim and hence ability to meet deadlines
- Control errors can be detected if there is redundant information
  - e.g., static common knowledge such as fixed schedules
  - Then have less or no need for data in control messages
    - ⋆ e.g., destination of message
- Consequences of control errors depend on interface

Elementary and Composite Control Interfaces

- Composite interfaces are those where control is bidirectional
  - Even when data flow is unidirectional
  - e.g., producer-consumer, queues

  Problem is they allow errors to propagate in both directions

- Elementary interfaces have unidirectional control
  - Same direction as data flow
  - e.g., wait-free, lock-free, atomic registers
  - Such as Simpson's 4-Slot, Non-Blocking Write (NBW)

  Errors propagate in only one direction

- Choice of data affects type of interface
  - Event vs. state messages
  - Events require confirmation: therefore composite

Fault-Containment Regions

- Two kinds of data errors
  - Send wrong value, send it at wrong time
- Value errors require redundancy and selection/voting
  - A host of delicate issues, understood by very few
    - ⋆ So you get homespun designs
    - ⋆ e.g., read incident/accident reports such as NTSB A07-65 through 86 (Predator), or A08-46, 47 (Eclipse)
  - A key idea is that of fault containment region
  - Required so that faults in redundant values will be independent
- Timing errors (e.g., babbling) are very destructive
  - Guaranteed elimination requires fixed schedules
  - e.g., static common knowledge, enforced by bus guardians
    - ⋆ Such as TTEthernet (used by Project Orion)

Composability

- aka. preservation of prior properties
- A property established for a component or subsystem will not be invalidated by system integration
- Even when other parts of the system have faults
- i.e., if components A and B do their thing within allocated processor and bus utilizations, rest of system must never invalidate this
  - Even when another component causes a processor exception
  - Or babbles on the bus
- Composability is a stepping stone to compositionality
- DO-297 talks of "tiers of integration"

Partitioning

- Allows components to share processor, bus resources
- By eliminating unintended interactions
- Space partitioning
  - Cannot read/write another component's memory
- Time partitioning
  - Cannot affect another component's access to processor/bus allocation
- Robust partitioning is key technology for integrated modular avionics (IMA)
- Now COTS technology from RTOS vendors
- And avionics buses: AFDX (weak), TTEthernet (strong)

Assurance and Certification

- We have claims or goals that we want to substantiate
  - Concerning some system property
- We produce evidence about the product and its development process to support the claims
  - E.g., analysis and testing of the product and its design
  - And documentation for the process of its development
- And we construct an argument that the evidence is sufficient to support the claims
- This is the intellectual basis for all certification regimes
- Claims and argument generally implicit in standards-based assurance, which focus on evidence to be produced
- Argument-based safety/security/dependability cases require explicit claims, evidence, argument

Compositional Assurance/Certification

- Assurance case may not decompose along architectural lines
  - Insight due to Ibrahim Habli & Tim Kelly
- Goes to the heart of what is an architecture
- A good one supports and enforces the assurance case
- We've now got enough background to see how to do this

Synthesis

We need:

- Robust partitioning to share processor and bus resources
- Determinism to control faults in the time domain
- Redundancy to tolerate faults in the data domain
- And fault containment regions so faults are independent
- Elementary control interfaces to provide error-propagation boundaries
- Composability so we can build things piecewise (layers)

Relationship to Security

- Adversary models for security are generally stronger than fault hypotheses for safety and fault tolerance
  - Active malice rather than Mother Nature
  - Though Mother Nature is assumed to be a strong cryptographer
    - ⋆ e.g., checksums and nuclear triggers
- Disclosure is more subtle than (most) faults
  - Any observable variation in behavior can be a side channel or covert channel that discloses sensitive information
- Manifestation of this is that security is not even a property

Properties and Security

- A property is a (possibly infinite) set of behaviors
  - Safety property: no bad thing happens
  - Liveness property: good things do happen (eventually)
  - Any property is the intersection of a safety property and a liveness property
- The only things we can enforce are safety properties
- Information flow security is not a property
  - It's a hyperproperty
  - Sets of sets of behaviors
- But every hyperproperty can be enforced by a safety property
  - e.g., information flow enforced by access control
  - May exclude some good behaviors
- We'll enforce safety properties, do end-to-end analysis as hyperproperties

The MILS Idea

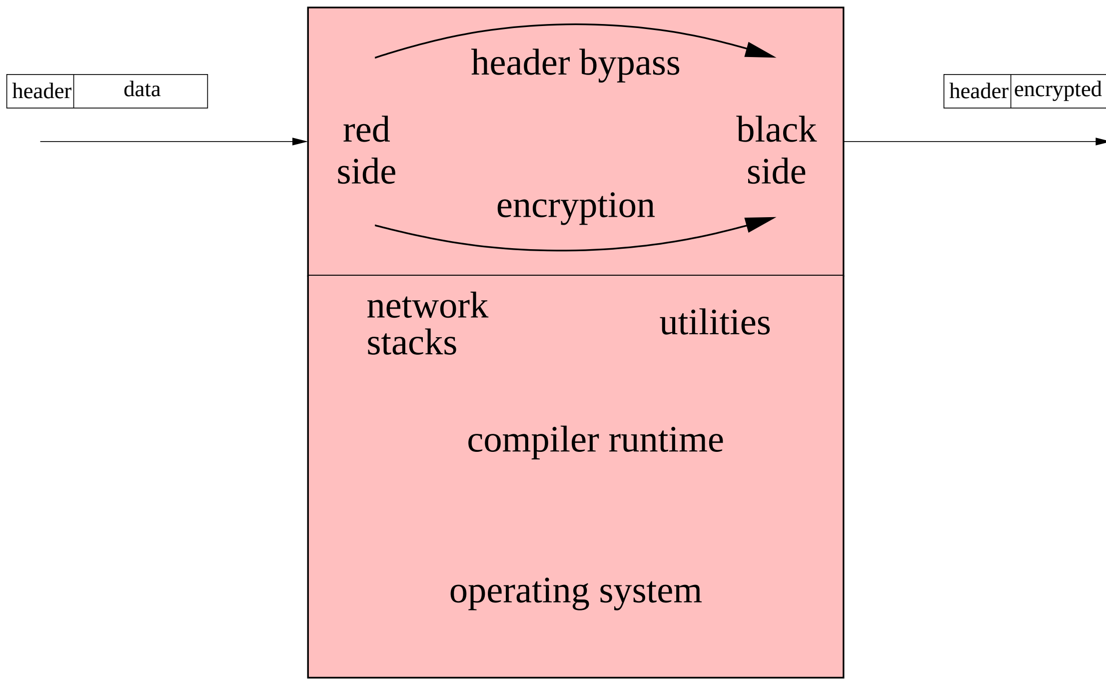
- Construct an architecture so that security assurance does decompose along structural lines
- Two issues in security:
  - Enforce the security policy
  - Manage shared resources securely
- The MILS idea is to handle these separately
- Focus the system architecture on simplifying the argument that policy is enforced correctly
  - Hence policy architecture
- Policy architecture is the interface between the two issues

Policy Architecture

- Intuitively, a boxes and arrows diagram
  - There is a formal model for this
- Boxes encapsulate data, information, control
  - Access only local state, incoming communications
  - i.e., they are state machines
- Arrows are channels for information flow
  - Strictly unidirectional
  - Absence of arrows is often crucial
- Some boxes are trusted to enforce local security policies
- Want the trusted boxes to be as simple as possible
- Decompose the policy architecture to achieve this
- Assume boxes and arrows are free

Crypto Controller Example: Step 1

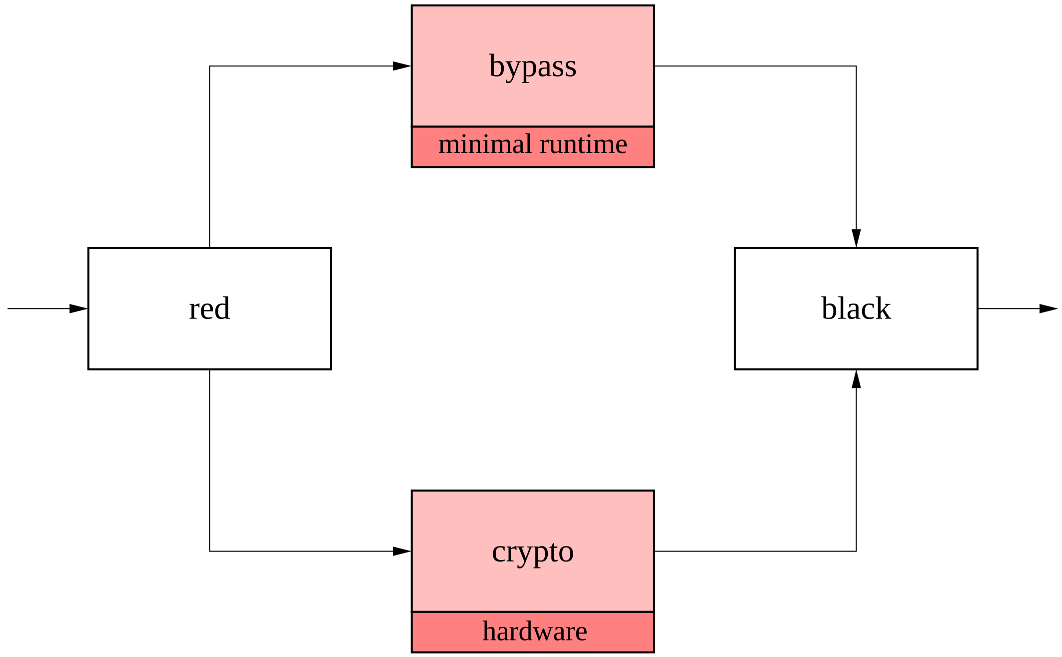
Policy: no plaintext on black network



No architecture, everything trusted

Crypto Controller Example: Step 2

Good policy architecture: fewer things trusted



Local policies (notice these are intransitive):

Header bypass: low bandwidth, data looks like headers

Crypto: all output encrypted

Policy Architecture: Compositional Assurance

- Construct assurance for each trusted component individually
  - i.e., each component enforces its local policy
- Then provide an argument that the local policies
  - In the context of the policy architecture

  Combine to achieve the overall system policy
- Medium robustness: this is done informally
- High robustness: this is done formally
  - Compositional verification

Enforcing Assumptions Of The Policy Architecture

- Primarily separation
- Five basic mechanisms available
  - physical: separate boxes
    - ⋆ But even they may need wrapping
  - temporal: classic periods processing
  - cryptographic: encryption and checksums
  - logical: verify no interference
    - ⋆ Only works when you have all the code
  - separation kernel: runtime enforcement
- Also need unidirectional arrows
  - Data diodes etc.
- Generally want to combine separation with resource sharing

Resource Sharing

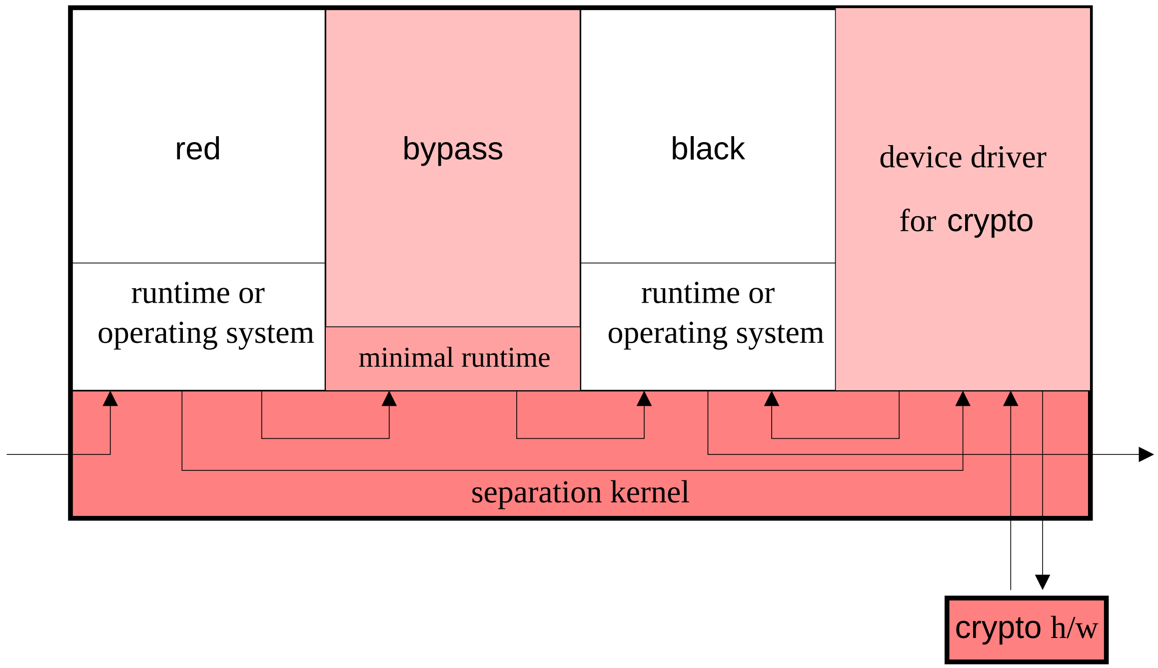
- Next, we need to implement the logical components and the communications of the policy architecture in an affordable manner
- Allow different components and communications to share resources
- Need to be sure the sharing does not violate the policy architecture
  - Flaws might add new communications paths
  - Might blur the separation between components

Secure Resource Sharing

- For broadly useful classes of resources
  - e.g., file systems, networks, consoles, processors
- Provide implementations that can be shared securely
- Start by defining what it means to partition specific kinds of resource into separate logical components
- Definition in the form of a protection profile (PP)
  - e.g., separation kernel protection profile (SKPP)
  - or network subsystem PP, filesystem PP, etc.
- Then build and evaluate to the appropriate PP

Crypto Controller Example: Step 3

Separation kernel securely partitions the processor resource



The integrity of the policy architecture is preserved

Resource Sharing: Compositional Assurance

- Construct assurance for each resource sharing component individually
  - i.e., each component enforces separation
- Then provide an argument that the individual components
  - Are additively compositional
  - e.g., **partitioning(kernel)** + **partitioning(network)** provides **partitioning(kernel + network)**

  And therefore combine to create the policy architecture
- Medium robustness: this is done informally
- High robustness: this is done formally
  - Compositional verification

Summary

- MILS (and HAP) are in the mainstream of architectures promoting compositional development and assurance for critical systems
- Ahead in some areas
  - e.g., the policy architecture, COTS cultivation
- Behind in some others
  - e.g., use of elementary control interfaces
  - tool support for assurance
- The challenge ahead is compositional certification
- And regulatory adjustment to enable this

Thanks

- Joyce Brookins and others at USAF Cryptomod
- Wilmar Sifre and others at AFRL
- Carolyn Boettcher at Raytheon
- Rance DeLong