Perspectives on Security

Butler Lampson
Microsoft Research
Symposium on Operating Systems Principles
October 4, 2015

How did we get here?

- In the beginning, security was by physical isolation (1950-1963)
  - **Easy**: You bring your data, control the machine, take everything away
  - Still do this today with VMs and crypto (+ enclaves if VM host is untrusted)
- Timesharing brought the basic dilemma of security: (1963-1982)

Isolation vs. sharing

  - **Hard**: Each user wants a private machine, isolated from others
  - but users want to share data, programs and resources
- Since then, things have steadily gotten worse (1982-2015)
  - Less isolation, more sharing, no central management
  - More valuable stuff in the computers
  - Continued misguided search for perfection (following the NSA's lead)

Wisdom

- If you want security, you must be prepared for inconvenience.

  —General B.W. Chidlaw, 12 December 1954

- When it comes to security, a change is unlikely to be an improvement.

  —Doug McIlroy, ~1988

- The price of reliability is the pursuit of the utmost simplicity.
  It is a price which the very rich find most hard to pay.

  —Tony Hoare, 1980 (cf. Matthew 19:24)

- But who will watch the watchers? She’ll begin with them and buy their silence.

  —Juvenal, sixth satire, ~100

What we know how to do

- Secure something simple very well
- Protect complexity by isolation and sanitization
- Stage security theatre

What we don’t know how to do

- Make something complex secure
- Make something big secure if it’s not isolated
- Keep something secure when it changes
- Get users to make judgments about security
- Understand privacy—fortunately not an SOSP topic

Themes

- **Goals**: Secrecy (confidentiality), integrity, availability (CIA: Ware 1970)
- **Gold standard**: Authentication, authorization, auditing (S&S 1975)
- **Principals**: People, machines, programs, … (Dennis 1966, DEC 1991)
- **Groups/roles**: make policy manageable (Multics 1968, NIST 1992)

Oppositions

(*in deployment, not good vs. bad*)

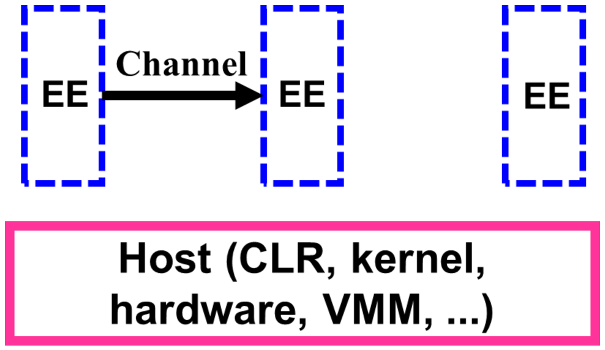
Winner		*Loser*
Convenience	vs.	Security
Sharing	vs.	Isolation
Bug fixes	vs.	Correctness
Policy/mechanisms	vs.	Assurance
Access control	vs.	Information flow

Timeline

	Themes	Systems
1960s	**Timesharing**; ACLs; access control matrix; VMs; passwords; capabilities; domains; gates	CTSS; Multics; CP/CMS; *Cal TSS*; Adept-50; Plessey 250
1970s	**TS**; LANs/Internet (e/e security); public key; multi-level sec.; ADTs/objects; least privilege; Trojans; isolation by crypto; amplification; undecidability	Unix; VMS; VM/370; IBM RACF; Clu; Hydra; Cambridge CAP
1980s	**Workstations; client/server**; Orange Book; global authentication; Clark and Wilson	A1 VMS; SecureID; Morris worm; IX
1990s	**PCs; Web**; sandboxes; Java security; crypto export; decentralized information flow; Common Criteria; biometrics; RBAC; BAN; SFI; SET	Browsers; SSL; NT; Linux; PGP; *Taos*
2000s	**Web; JavaScript**; buffer overflows; DDoS	TPM; LSM; SELinux; seL4; HiStar
2010s	**Web; big data**; enclaves; homomorphic crypto	Singularity; CryptDB; Ironclad ...

Foundation: Isolation

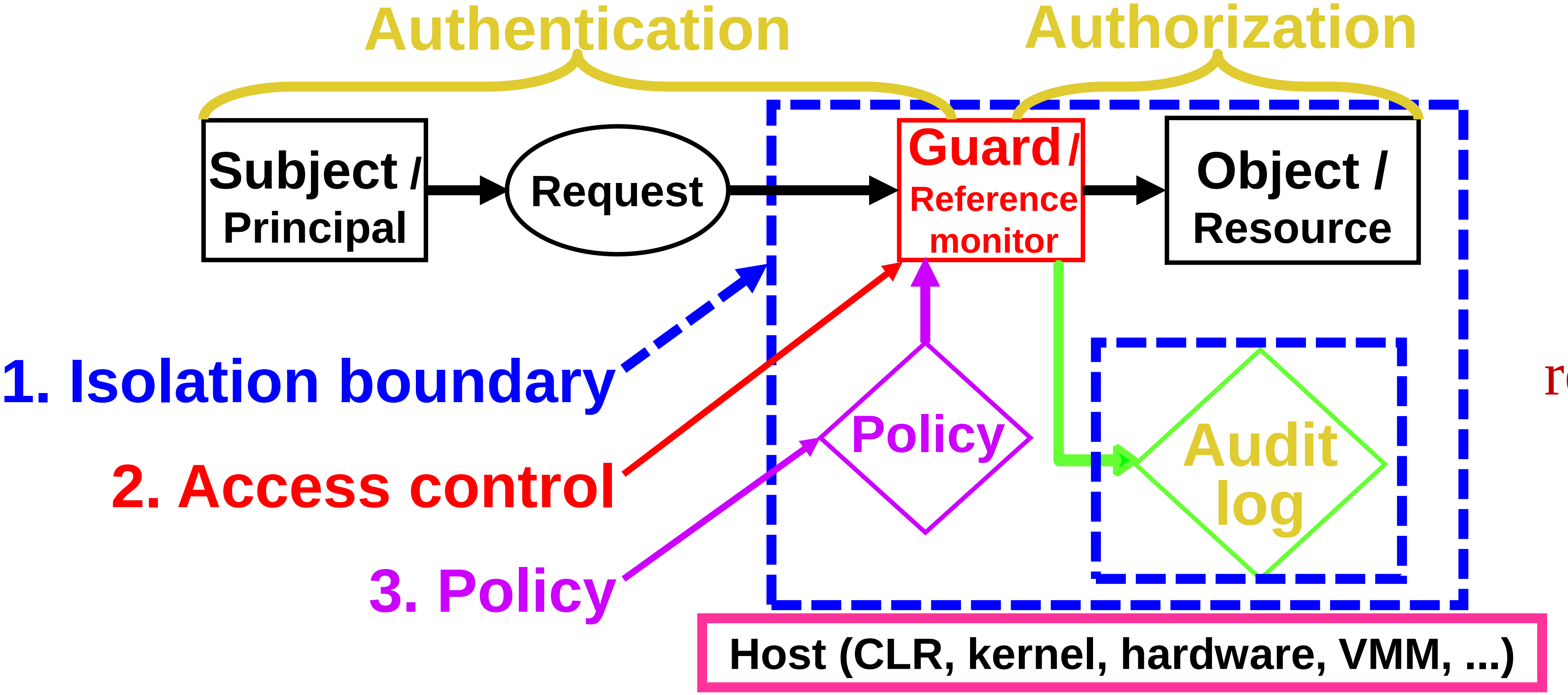
- A host isolates an execution environment
  - The basis for any security. Must trust the host
- Many ways to do it (and many bugs):



Mechanism	Host	
Java/JavaScript sandboxing	JVM/JS engine	Java 1995
Modules/objects	language/runtime	Clu 1974
Software fault isolation	process	Wahbe et al 1993
Processes	OS	CTSS 1961
Virtual machines	hypervisor	CP/40 1966
Limited comm (wires or crypto)	network	DESNC 1985
Air gaps: physical separation	physics	1950; Tempest ~1955

Safe Sharing: Access Control

1. **Isolation boundary** limits attacks to channels (no bugs)
2. **Access Control** for channel traffic
3. **Policy** sets the rules

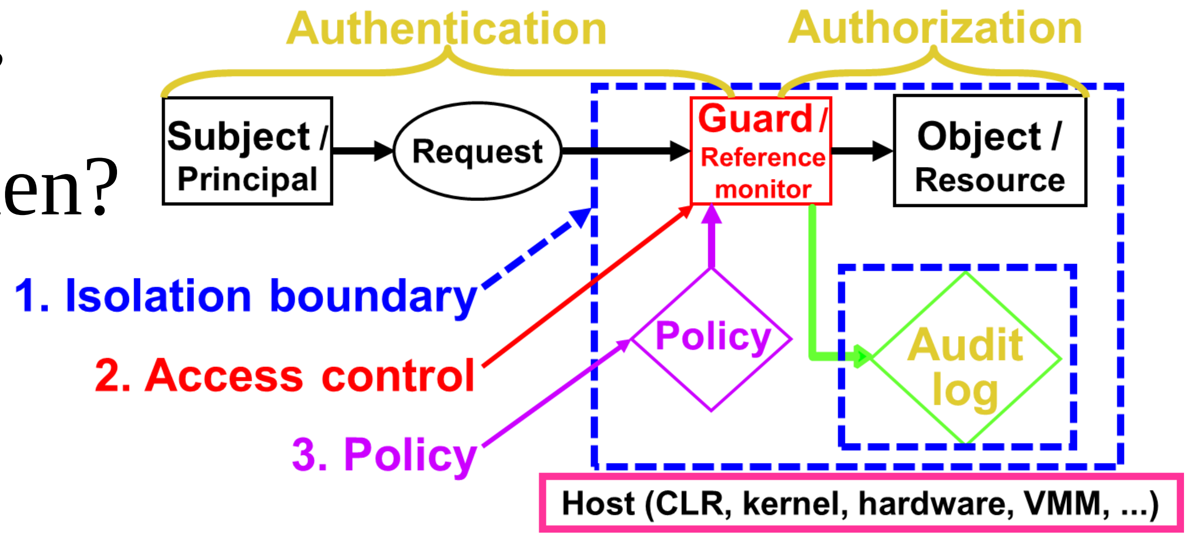


Anderson report 1972

Access Control: The Gold Standard

- **Authenticate** principals: Who made a request?
  - People, but also channels, servers, programs

    (encryption implements channels, so the key is a principal)
- **Authorize** access: Who is trusted with a resource?
  - *Group* principals or resources, to simplify management
    - Can define a group by a property, e.g. "type-safe" or "safe for scripting"
- **Audit** requests: Who did what when?



Policy: What sharing is allowed?

- The guard evaluates a function: permissions = policy(subject, object)
  - If functions are too mathematical, call it an access matrix (Lampson 1971)

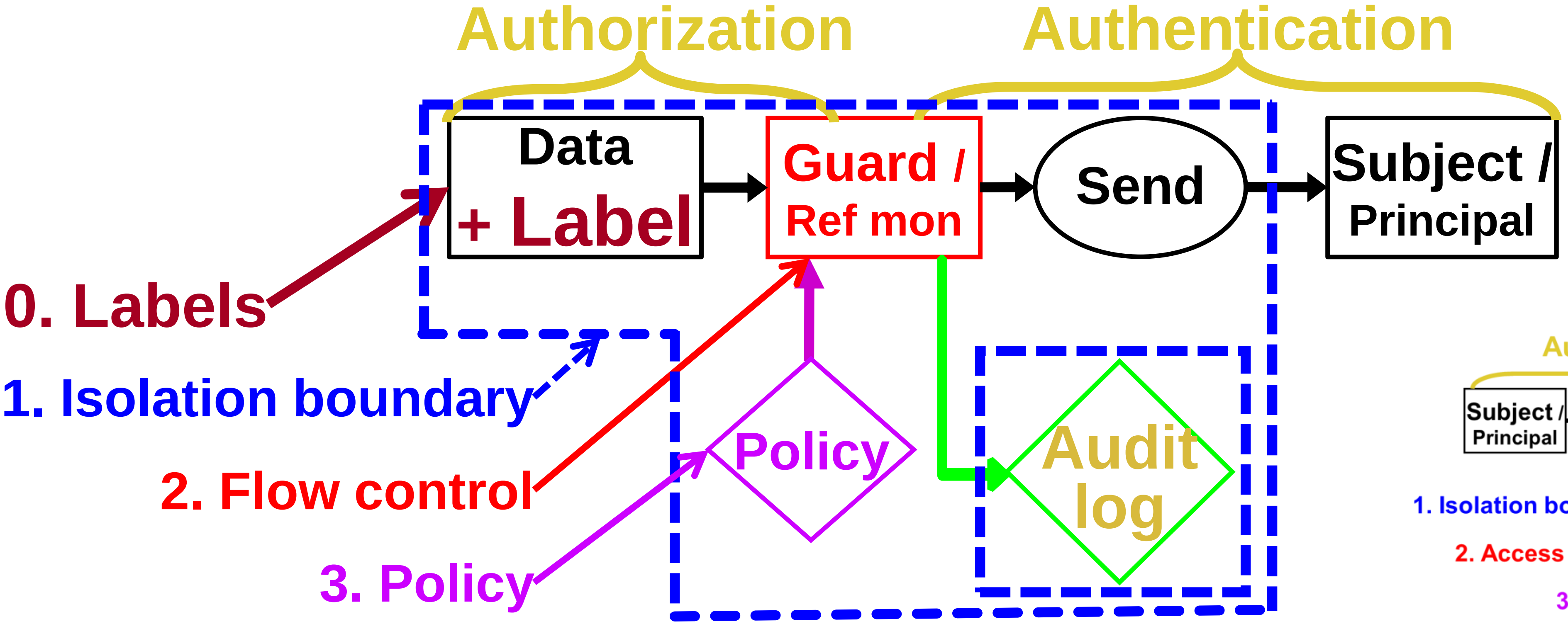
Subject/principal	Object/resource	
	File foo	Database payroll
Alice	*read*, *write*	*write paychecks*
Bob	*read*	-

- Permissions kept at the object are ACLs; at the subject, capabilities
  - Capabilities work for short term policy
    - File descriptors/handles in OS; objects in languages (Unix/Windows; Java, C#)
  - ACLs work for long-term policy; tell you who can access the resource
    - **Groups** of subjects and objects keep this manageable (Multics 1968)

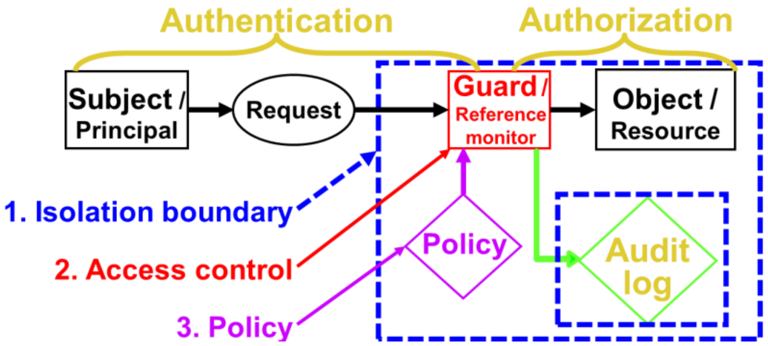
Keeping Secrets: Information Flow Control

0. **Labels** on information
1. **Isolation boundary** limits flows to channels
2. **Flow control** based on labels
3. **Policy** says what flows are allowed

Adept-50 1969
Orange Book 1985



Compare access control



Information Flow Control

- Invented to model military classification (Adept-50 1969)
  - **Label** every datum: top secret/nuclear ≥ top secret ≥ secret
    - Labels form a lattice, and propagate: If d_1 is input to d_2, then d_2's label is ≥ d_1's
  - Enforce with access control: label subjects, containers (Bell/LaPadula 1973)
    - No read up, write down; can be dynamic or static (Adept-50; Denning 1976)
- **Decentralized** flow control (Myers and Liskov 1998)
  - Anyone can invent labels. If you own a label, you can declassify it
    - Can do this in a language or in an OS (Jflow 1999; HiStar 2006)
- So far, none of this has been practical
- And then there are **covert** (side) channels: timing, radiation, power ...
  - Abstractions don't keep secrets (Tempest 1955, Lampson 1972)

Who controls policy? DAC, MAC, RBAC

- How to decide:
  - Is the user or the program **malicious**? Insiders, Trojan horses
  - Is the user **competent**? Policy can be tricky
  - Is the user **motivated**? Security gets in the way of work and play

- Discretionary access control (DAC) : the object's owner (CTSS 1963)
- Mandatory access control (MAC) : an administrator (1969; 1985)
  - MAC ≠ flow control
- Role based access control (RBAC): the app designer (NIST 1992)
  - Administrator just populates the roles

Distributed Systems: Cryptography

- Systems **communicate**, so need secure channels
  - Host, secure wire, …, but usually cryptography: it's general, **end-to-end**
- Basic crypto functionality: mathematical magic that implements:
  - **Sign** with K^{-1}/ verify with K : integrity
  - **Seal** with K / unseal with K^{-1}: secrecy

  } You can only do it if you know the key
  - This gives you an end-to-end secure channel
- Public key crypto: $K \neq K^{-1}$; I can sign, anyone can verify (RSA 1977)
- **Homomorphic** crypto: compute on encrypted data (Gentry 2009)
  - This is too slow, but you can *simulate* it (CryptDB 2011)
- Zero knowledge and **verifiable** computation (Pinocchio 2013)

Distributed Systems: Understanding Trust

- Systems are **decentralized**, so we must reason about trust
  - We need proofs to justify such reasoning
- Principals: people, machines, programs, services, protocols, …
- Accountability: principal **says** statement
  - Alice **says** read from file Foo
- Trust: principal A **speaks for** principal B
    - Alice **says** Bob@microsoft **speaks for** Alice
    - Microsoft **says** Key63129 **speaks for** Bob@microsoft
    - Key63129 **says** read from file Foo
  - Extending this to authorization yields an end-to-end argument
    - file Foo **says** Alice **speaks for** file Foo ACL entry
    - So Foo **says** read from file Foo

DEC 1989, 1991

Does it actually work? Assurance (Correctness)

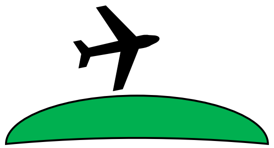
- Keep it simple—Trusted Computing Base (TCB) (Rushby 1981)
  - One way: a security kernel—apps out of the TCB. This works for sharing hardware
- Ideally, you **verify**: prove that a system satisfies its security spec
  - This means that *every* behavior of the system is allowed by the spec
    - Not the same as proving that it does everything in the manual
  - Today in seL4, Ironclad, … First tried in Gypsy (late 1970s)
  - What if the spec is wrong? Keep it simple
- Usually verifying is too hard, so you **certify** instead
  - Through some “independent” agency. Alas, process trumps substance
    - First by DoD for Orange Book, later international Common Criteria (1985, 1999)
- Or you can verify **some** properties: isolation, memory/type safety
- Or you can apply bandaids

Bandaids for Bugs (Defense in Depth)

- No guarantees, but at least the bad guy has to work harder
  - **Firewalls** to keep intruders out, look for suspicious traffic (DEC 1988)
  - **Signature** hacks to detect malware (~1990)
  - **Memory safety** hacks to catch writes outside array bounds (Phrack 1996)
  - **Intrusion detection** hacks to look for anomalous behavior (SRI 1986)
  - **Control Flow Integrity** to block jumps not in the normal flow (MSR 2005)
  - **Taint tracking** to keep unsanitized input away from execution (CMU 2005)
  - **Process** to enforce use of the tools (MS SDL 2004)
- "I don't have to outrun the bear; I just have to outrun you."
  - These are not bad things, but they are hacks

What about *my* system? Configuration (Policy)

- If the code is correct, the configuration may still be wrong
  - You write the code once, but every system has its own configuration
  - It’s boring, and it changes. So either it’s small, or it’s wrong.
    - But it’s not small; there’s always another feature, another plausible scenario
      - There are 12 levels of indirection in Windows printing, each with its own security
- And configuration is usually done by amateurs
  - With MAC and RBAC at least it’s done by pros
- **Conflict**: want fine grained policy, but can only manage coarse grain
  - Not much work on this, and it remains unsolved
    - Solution (never adopted): Lower aspirations, budget for complexity



What has worked? What hasn't?

Worked ~ gotten wide adoption

Worked

- VMs
- SSL
- Passwords—universal
- Safe languages
- Firewalls
- Process—SDL

Failed

- "Secure systems"
- Capabilities (except short term)
- Metrics for security
- MLS/Orange book
- User education
- Intrusion detection

Why don’t we have “real” security?

- **A**. People don’t buy it
  - Danger is small, so it’s OK to buy features instead
  - Security is expensive
    - Configuring security is a lot of work
    - Secure systems do less because they’re older
  - Security is a pain
    - It stops you from doing things
    - Users have to authenticate themselves
  - Goals are unrealistic, ignoring technical feasibility and user behavior
- **B**. Systems are complicated, so they have bugs
  - Especially the configuration

What next?

- **Lower aspirations**. In the real world, good security is a bank vault
  - Hardly any computer systems have anything like this
  - At best we can only make simple things secure
- Access control doesn't work—40 years of experience says so
  - Basic problem: its job is to say "No"
    - This stops people from doing their work, and then they relax the access control
    - usually too much, but no one notices until there's a disaster
- **Retroactive** security: focus on things that actually happened
  - rather than all the many things that *might* happen
  - Real world security is retroactive
    - Burglars are stopped by fear of **jail**, not by locks
    - The financial system's security depends on **undo**, not on vaults

Biertan fortified church, Romania

Jail

Lock