Usable Security Through Isolation

Collège de France

April 6, 2011

Butler Lampson

Microsoft Research

Usable Security: Things Are Really Bad

- Users don’t know how to think about security
- User experience is terrible
  - Lots of incomprehensible choices
    - Just say “OK”
  - A few examples:
    - Windows Vista User Account Control
    - Windows root certificate store
    - User interface for access control on files
    - Password phishing
    - Client certificates for SSL
    - Signed or encrypted email
- In general, more secure = less usable

The Best is the Enemy of the Good

- Security is fractal
  - Each part is as complex as the whole
  - There are always more things to worry about
    - See Mitnick's *Art of Deception*, ch. 16 on social engineering
- Security experts always want more—
  - More options : There's always a plausible scenario
  - More defenses: There's always a plausible threat
- Users just want to do their work
  - If it's not simple, they will ignore it or work around it
  - If you force them, less useful work will get done

Usable Security Is About Economics

- Security is about risk management, not an absolute
  - There's benefit, and there's cost
    - We don't measure either one
    - Compare credit cards: fraud detection, CCVs, chip-and-PIN
    - The cost is *not* mostly in budgeted dollars
      - If you want security, you must be prepared for inconvenience.
        —General B. W. Chidlaw, 12 Dec. 1954
  - Tight security → no security
- Sloppy users are doing the right thing
  - With today's poor usability, the cost of security is high
  - And the benefits of better security are quite low
- Providers have no incentive for usable security
  - They mostly just want to avoid bad publicity

What Has Worked?

- Worked = gotten wide adoption
  - SSL
  - Passwords
  - Firewalls
  - Security life cycle
  - Safe languages

Technical Context

- **Security** is about
  - Secrecy — Who knows it?
  - Integrity — Who changed it?
  - Availability — Is it working?
  - Accountability — Who is to blame?
- **Privacy** is about controlling personal information
  - What is known—very hard
  - How it is used—mainly by regulation
- Two faces of security: Policy vs. bugs
  - **Policy**: user's or org's rules for security / privacy
  - **Bugs** : ways to avoid policy

Assurance and Threats

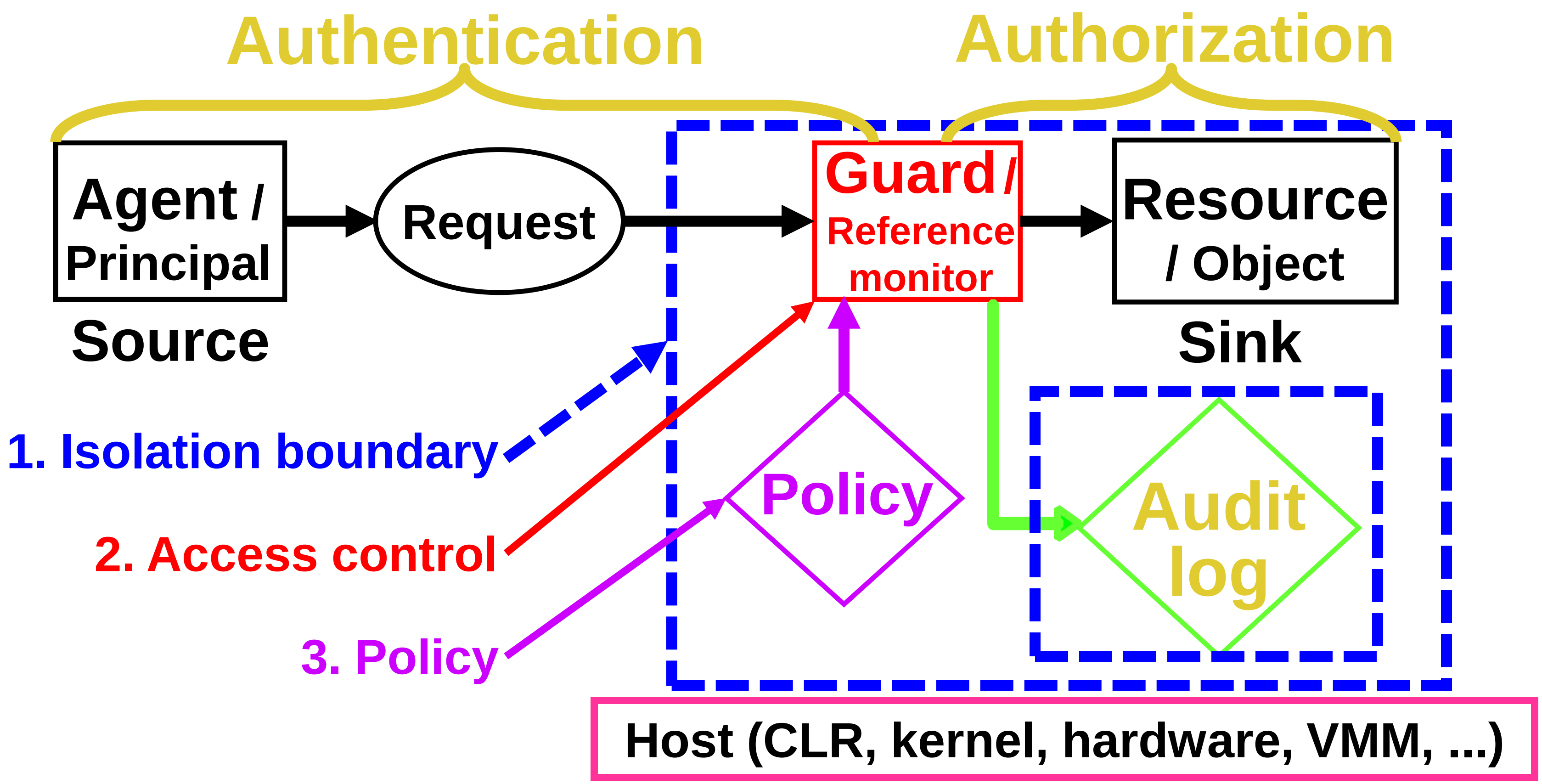
- Assurance:
  - **Policy**: Computer settings agree with user's or org's rules for security / privacy
  - **Bugs** : There is no way to avoid policy
- Assurance depends on the **threat model**—What the adversary can do.
- This depends on the adversary.There's a range:
  - User of downloaded tools

    ↓

  - National intelligence agency

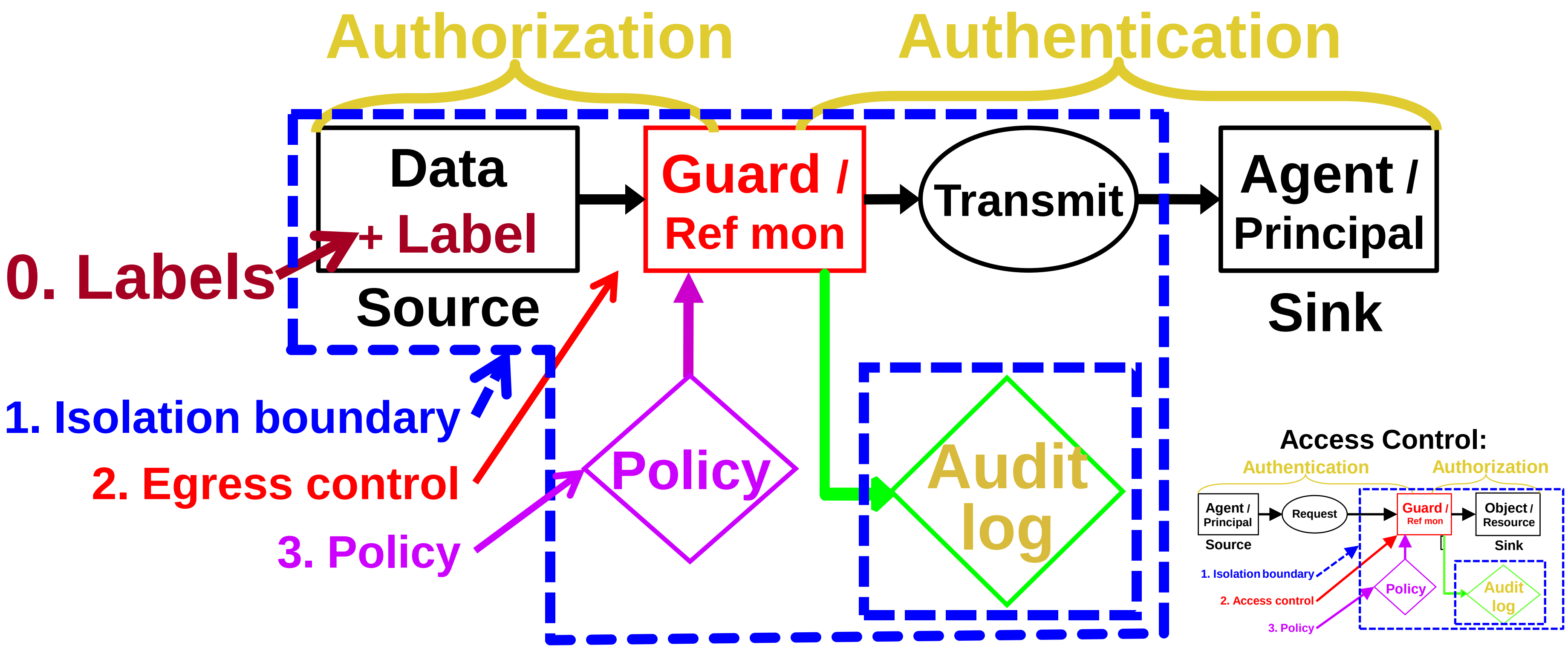
Context: The Access Control Model

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



Context: The Information Flow Model

0. **Labeled** information
1. **Isolation boundary** limits flows to channels (no bugs)
2. **Flow control** based on labels
3. **Policy** says what flows are allowed

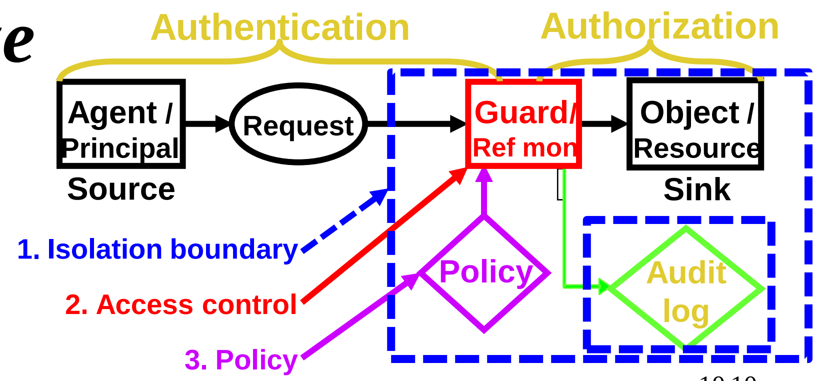


Access Control: The Gold Standard

- **Authenticate** principals: Who made a request
  - Mainly people, but also channels, servers, programs
    (encryption implements channels, so key is a principal)
- **Authorize** access: Who is trusted with a resource
  - *Group* principals or resources, to simplify management
    - Can define by a property, e.g. “type-safe” or “safe for scripting”
- **Audit**: Who did what when?

Lock = Authenticate + Authorize

Deter = Authenticate + Audit



Accountability

- Real world security is about deterrence, not locks
- On the net, can't find bad guys, so can't deter them
- Fix? End nodes enforce **accountability**
  - Refuse messages that aren't accountable enough
    - or strongly isolate those messages
  - Senders are accountable if you can **punish** them
    - With dollars, ostracism, firing, jail, ...
  - **All trust is local**
- Need an ecosystem for
  - Senders becoming accountable
  - Receivers demanding accountability
  - Third party intermediaries

Accountability vs. Access Control

- “In principle” there is no difference

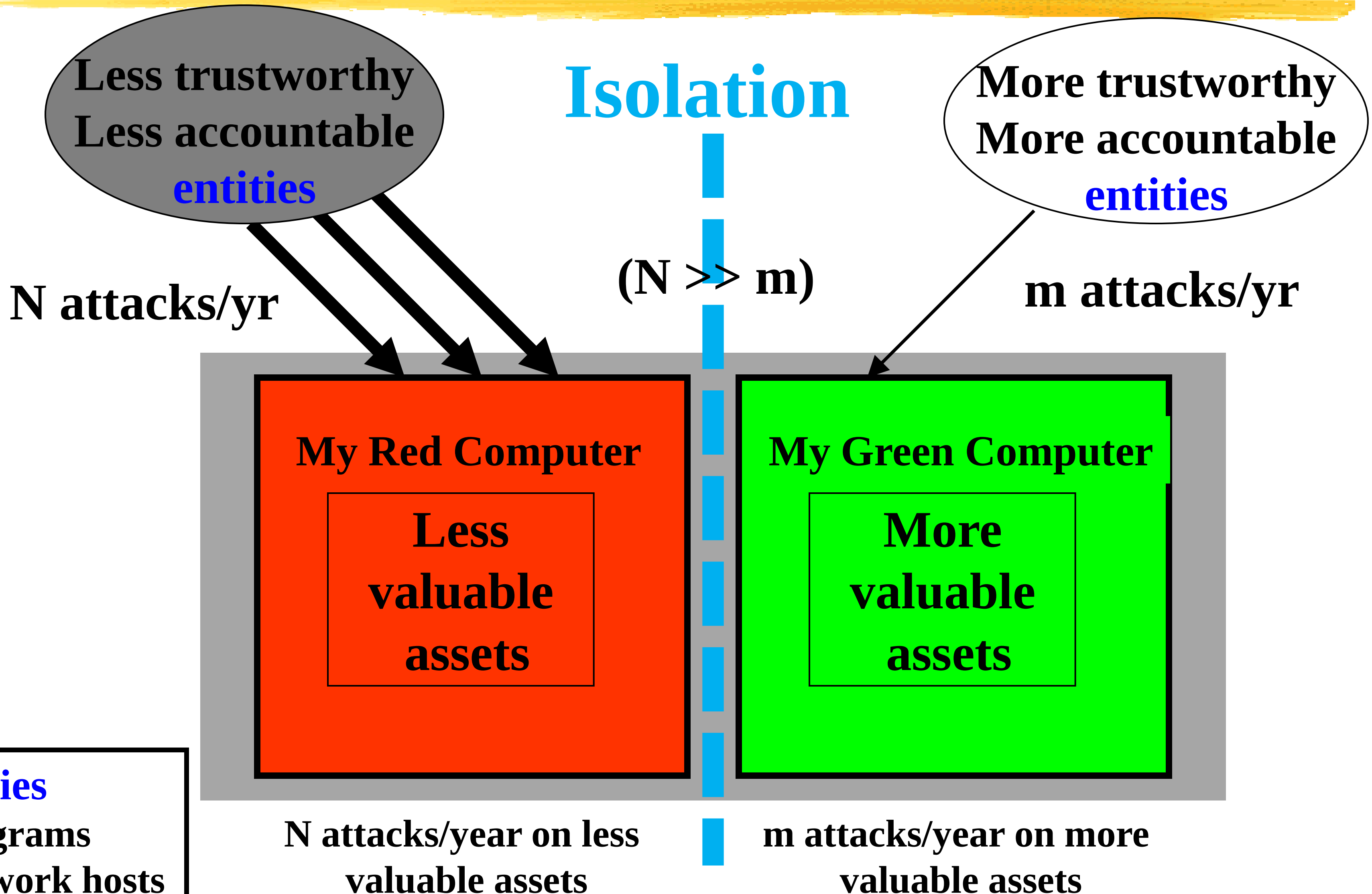
  but

- Accountability is about **punishment**, not access
  - Hence audit is critical
  - But coarse-grained control is OK—fix errors later

Freedom with Accountability?

- Partition world into two parts:
  - Green: More safe/accountable
  - Red   : Less safe/unaccountable
- Red / green has two aspects, mostly orthogonal
  - User experience
  - Isolation mechanism
- Green world needs professional management

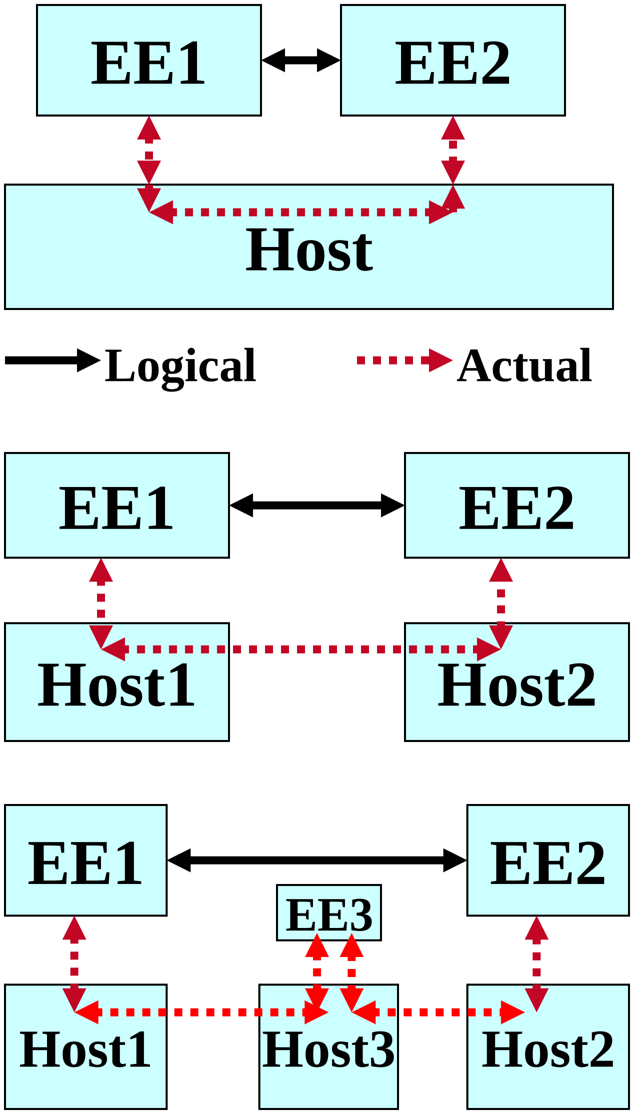
Red | Green



Entities
- Programs
- Network hosts
- Administrators

Hosts and Channels

- Host runs Execution Environments (EEs) and channels between EEs
- Host itself is an EE running a resource manager
  - EEs and channels are its resources
  - Recursive: It has its own host
    - Or it's a physical machine
- If EEs are on different hosts, use inter-host channel
  - Recursive: Host is an EE
  - Channel made by hosts' host, if any
    - Otherwise, by physical network
- No direct channel? Use middleman
  - Host3/EE3 is "host" for the network
    - It decides if Host1 and Host2 can talk



Definition of Isolation

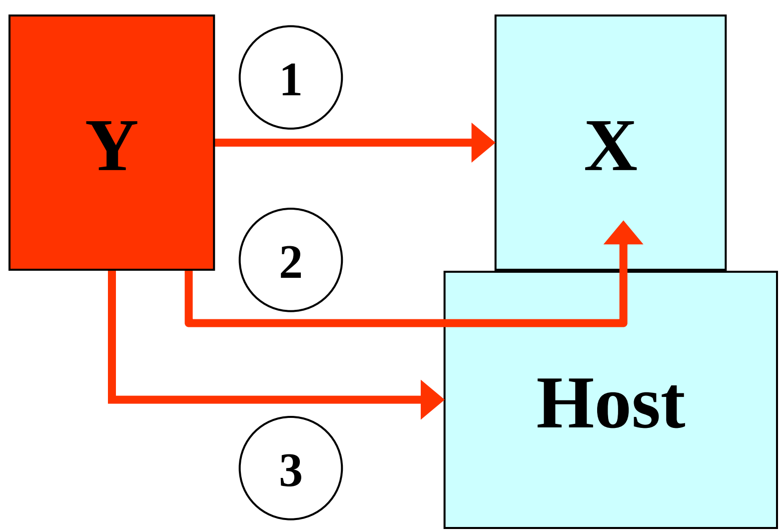
- X is ***isolated from*** Y if
  Y can't make X "go bad" (violate its spec)
  - Not symmetric; doesn't imply Y isolated from X
- To be isolated, you must
  - Isolate yourself: You handle anything correctly
    and/or
  - Be isolated: Your host only passes safe stuff to you

Attacks on Isolation

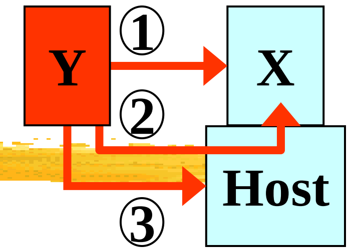
X is ***isolated from*** Y if Y can't make X "go bad" (violate its spec)

Attacks: How can Y make X go bad?

1. Send X some bad input
2. Use an unsafe function provided by X's host H
3. Make X's host H go bad

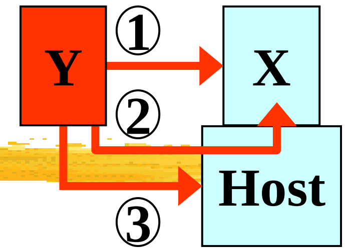


Y Attacks X: Details



Attack	Source	Example
1a. **Direct** bad input Y to X on a **channel**	Inputs trusted too much	Buffer overflow Malformed data Hostile code
1b. **Indirect** bad input Y to X via a **service**	Inputs trusted; Bugs in service	Y writes a file, X reads it Y corrupts shared service
2. Use unsafe host functions	Code injection	Debugging, extensibility (e.g. windows hooks)
3. Make the host go bad	Bugs in host	Y exploits bug in hosted EE or inter-host channel
Any of the attack classes	Human error (often from complexity)	Bad configuration (admin) Bugs (developer) Unsafe choice (end user)

Y Attacks X: Defense



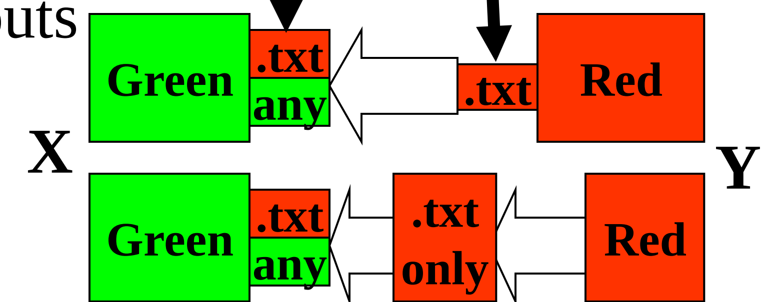
Attack	Defense
Direct bad input	No channels from Y to X
Y to X on a **channel**	X can't receive bad input
	X can handle all inputs from Y
	No inputs are bad
Indirect bad input	Service obeys host isolation policy
Y to X via a **service**	If not, host forbids service to have channels from both X and Y
	Assumption: Service is isolated from Y
	Assumption: Service access control policy enforces host's isolation policy
Unsafe host functions	Host forbids Y to use these functions
Make the host go bad	Host is isolated from Y

Isolation Policy: Labels

- Each EE has a label
  - The label is a principal
    - E.g., Red & Green, Secret & TopSecret, etc.
  - Trusted EEs can have more than one
- If client and server have no compatible labels, then channel isn't allowed
  - Identical labels are compatible
  - Some pairs of labels allow flow in one direction only
    - TopSecret can receive from Secret
    - Medium Integrity can send to Low Integrity
  - Compatibility is decided by policy

Isolation Policy: Safety

- Don't have to be so conservative:
  Not all inputs to X will cause it to go bad
  - An input to X is **safe** if it won't cause X to go bad
- Y's spec can says what **type** of outputs it produces
  - Such outputs are its **legal** outputs
- X's spec can say what input **types** are safe for it
  - E.g., `.txt` is safe, something more complex isn't
- Using safety: H allows Y $\rightarrow$ X only if
  Y's legal outputs $\subseteq$ X's safe inputs
  - H can trust Y's declaration of outputs
    - H could use Y's label to decide
  - Or, H can use its own database
    - E.g., IE Zones
  - Or, H can add a **filter**
    - In a trusted EE



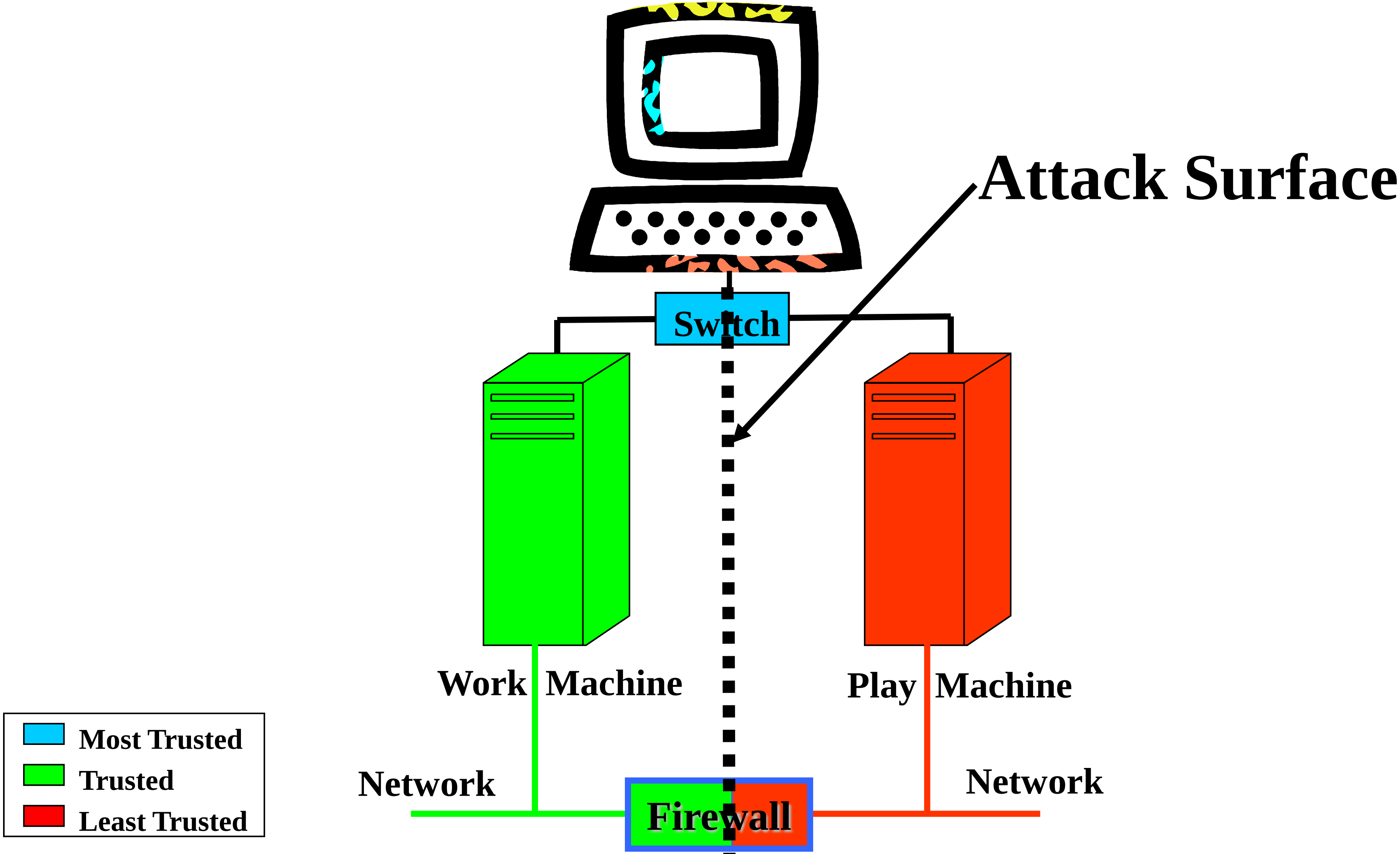
Isolation Policy vs. AuthZ Policy

Isolation Policy is authorization policy
It is the authorization policy of the host

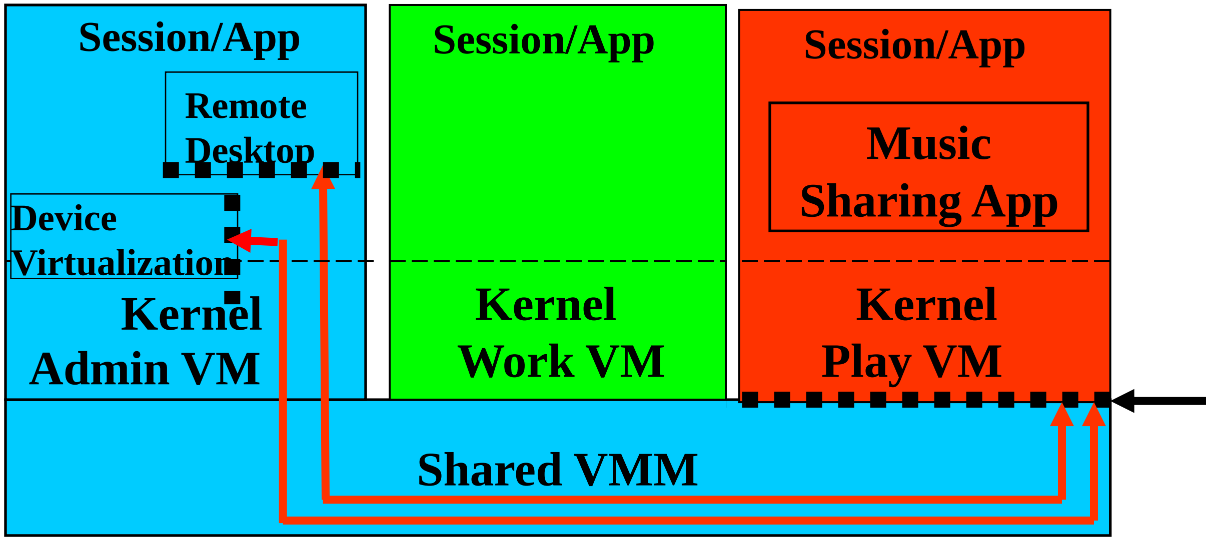
- **Isolation Policy**
  - Non-discretionary
  - Interpreted and enforced by the Host
  - Objective:
    - Allow/disallow creation/use of **channels** based on **EE** attributes
- **Access Control Policy**
  - Discretionary
  - Interpreted and enforced by the resource manager
  - Objective:
    - Allow/disallow creation/use of **resources** based upon **principal** attributes

This pattern is repeated at every layer of host

Switch Based Isolation

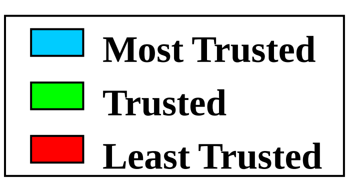


VMM Isolation

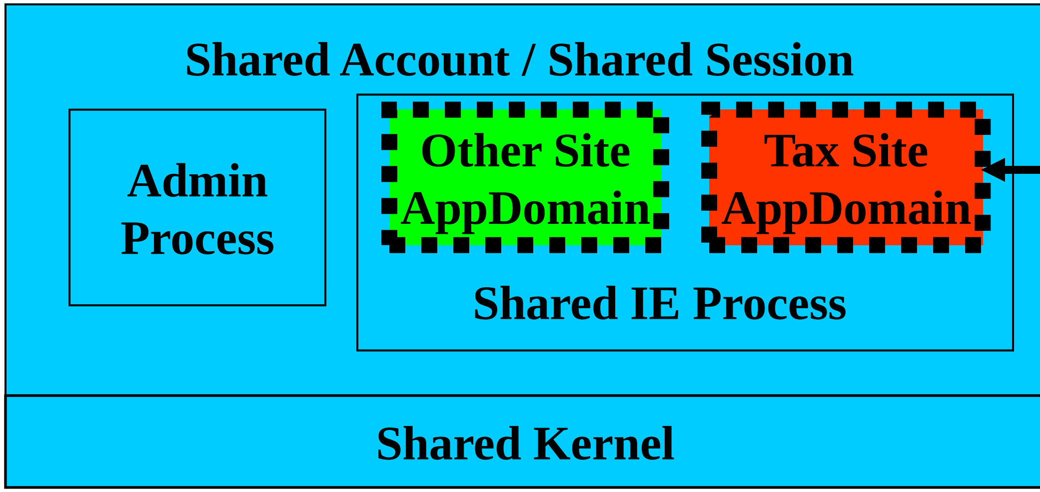


Attack Surface:
Remote desktop, network connectivity, **VMM APIs, Device Virtualization…**

- VMM emulates multiple physical machines
- Separate virtual disks
- Communication over virtual network
  - Virtual firewall in host



Browser / CLR Isolation



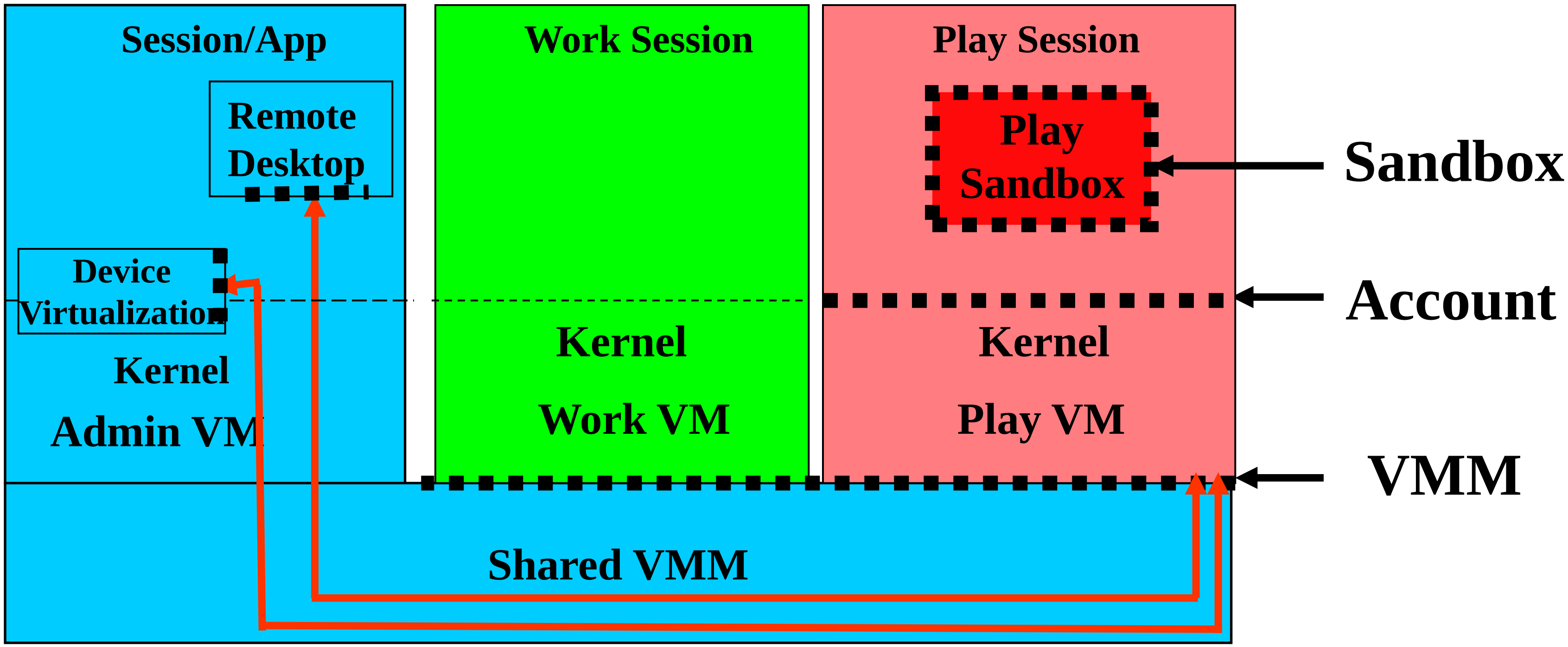
Attack Surface: APIs exposed to the sandbox; shared cookie state (cross-site scripting)

Note: Session and Account surfaces no longer exposed

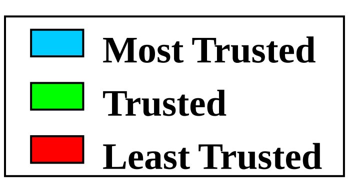
- Isolation mechanism in widespread use today – most secure because we've invested so much
- "Applications" (web pages) have very limited access to local resources. File access by user selection.
- Functionality could be expanded, but not practical for "full blown" applications

Most Trusted
Trusted
Least Trusted

Defense in Depth



Unless there are bugs that *line up* at multiple levels, the bugs are not exploitable.



Conclusions

- Things are really bad for usable security & privacy
  - Need to focus on essentials, not on frills
  - KISS: Keep It Simple, Stupid
- Isolation gives you:
  - Simple policy: Labels + safe inputs
  - Protection against bugs
- Need isolation at every level of host
  - Including the physical machine
- There are many ways to implement it