Retroactive Security

Schneider Symposium on Trustworthiness

Butler Lampson

Microsoft Research

December 5, 2013

Why Retroactive?

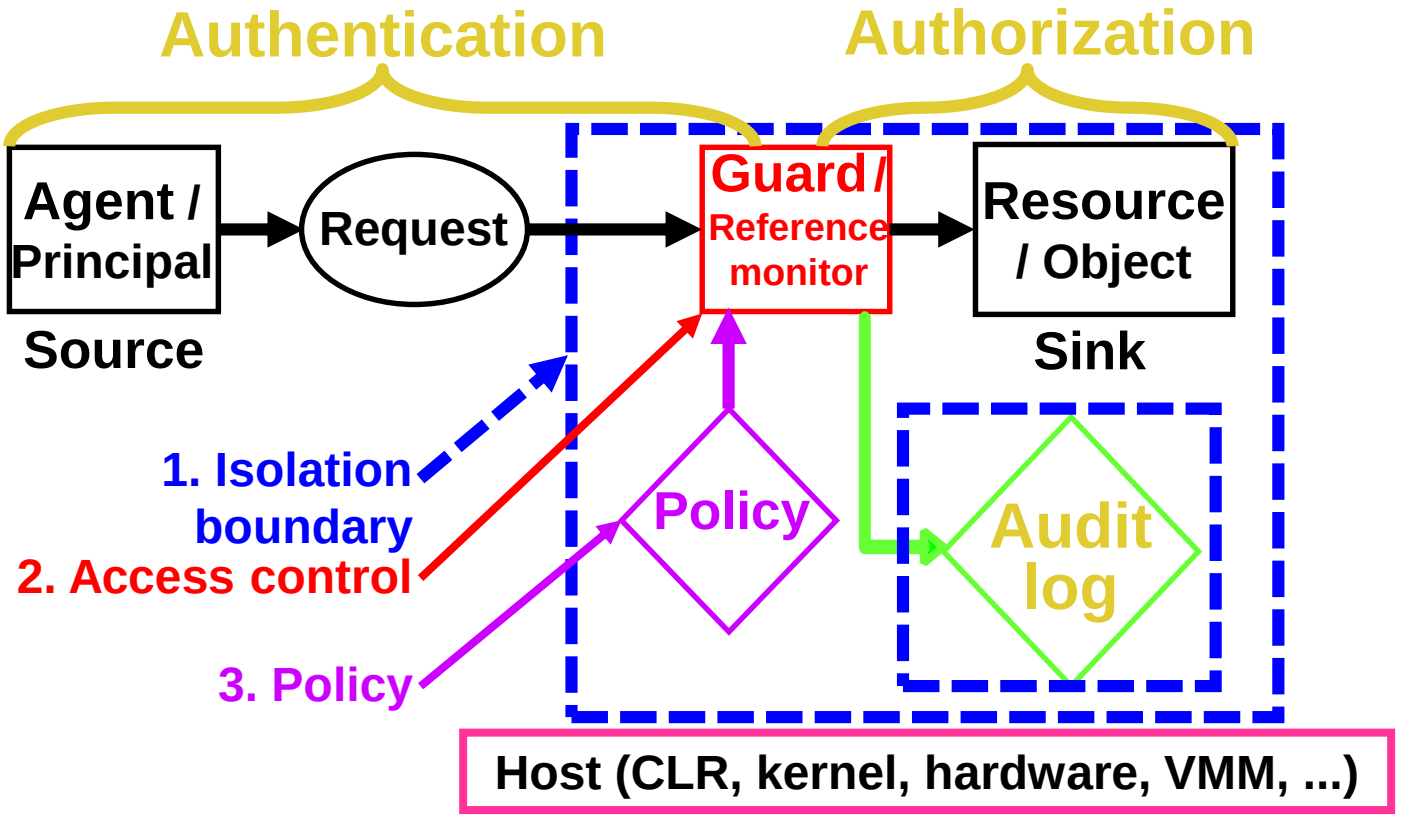
- Access control doesn't work
  - 40 years of experience says so
  - Basic problem: its job is to say "No"
    - This stops people from doing their work
    - and then they weaken the access control
    - usually too much, but no one notices
    - until there's a disaster
- Retroactive security focuses on things that actually happened
  - rather than all the many things that *might* happen

Why Retroactive?

- 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
  - Basic advantage: work on real, not hypothetical cases
- The best is the enemy of the good
  - Retroactive security is not perfect
  - But it's better than what we have now

Access Control

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



Aspects of Retroactive Security

- What about enforcing rules? Blame and punishment
  - Assigning blame? Auditing
  - Imposing punishment? Accountability
- What about integrity? Selective undo
- What about secrecy? Undo publication
- What about bugs? Accountability and isolation
- What about freedom? Red/Green

What About Punishment? 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**

What About Blame? Auditing

- Use access control just to keep out people you can't punish
  - End nodes enforce accountability
- Otherwise
  - Make common sense rules
  - Let people override the machine's enforcement
  - Log all accesses: who and what
  - For problems you notice, use the log to find culprits
  - Mine the record for unusual behavior, esp. overrides
- Needs authentication, and admin-friendly audit log

What About Integrity? Selective Undo

- A better form of "reinstall &reload from backup"
- Log all state changes, their inputs and their outputs
- To fix a corrupted system:
  - Reset the system to an old good state
  - Install patches and block known intrusions
  - Replay the logged actions (except the blocked ones)
    - Unchanged actions with unchanged inputs don't need replay
- This doesn't always work, but it often does
  - Sometimes it needs user advice to resolve conflicts
- Kaashoek, Zeldovich et al

What About Secrecy? Undo Publication

- How to stop the Internet from remembering forever
- When you post something, tag it as yours
- Well-behaved apps and services *respect* the tags.
  - Carry the tag along with the data
  - Consult the current policy for the tag
- To take something back, change the policy
- Enforcement by social norms or regulation
  - Works for Google, Facebook, MS Office, etc.
    - Of course doesn't work for everything

Ownership Tags

- Enough information to find the current policy
  - URL or search query for source of policy
  - HTTP request to retrieve policy
  - Public signing key to authenticate policy
- Current policy?
  - Cache retrieved policy
  - Check for changes—perhaps once per day or once per week
- Need the tag to last for decades

Ownership for Medical Data

- Same idea: tag data with patient identity
- Patient controls use of data
  - Who gets to see it
  - How it can be used in research
- Question: Can you take data back even after it's been used?

- See PITAC report on Health IT

From Ownership To Provenance

- Provenance: How this data came into being
  - Input, with owner(s)
  - Computed, by $f(x_1, x_2, ...)$
- Trace the chain of responsibility / ownership
- Recompute when inputs or program change

- Problems:
  - Cost
  - Process
  - Non-determinism

What About Bugs? Control Inputs

- Bugs will always subvert access control
  - Can't get rid of bugs in full-function systems
    - There's too much code, changing too fast
    - Timeliness and functionality are more important than security
- A bug is only dangerous if it gets tickled
  - So keep the bugs from getting tickled
  - Bugs get tickled by inputs to the program
  - So refuse dangerous inputs
    - or strongly isolate those inputs
  - To control possible inputs, isolate the program
    - VM, Drawbridge, process isolation, runtime or browser sandbox

Stopping Dangerous Inputs: Accountability

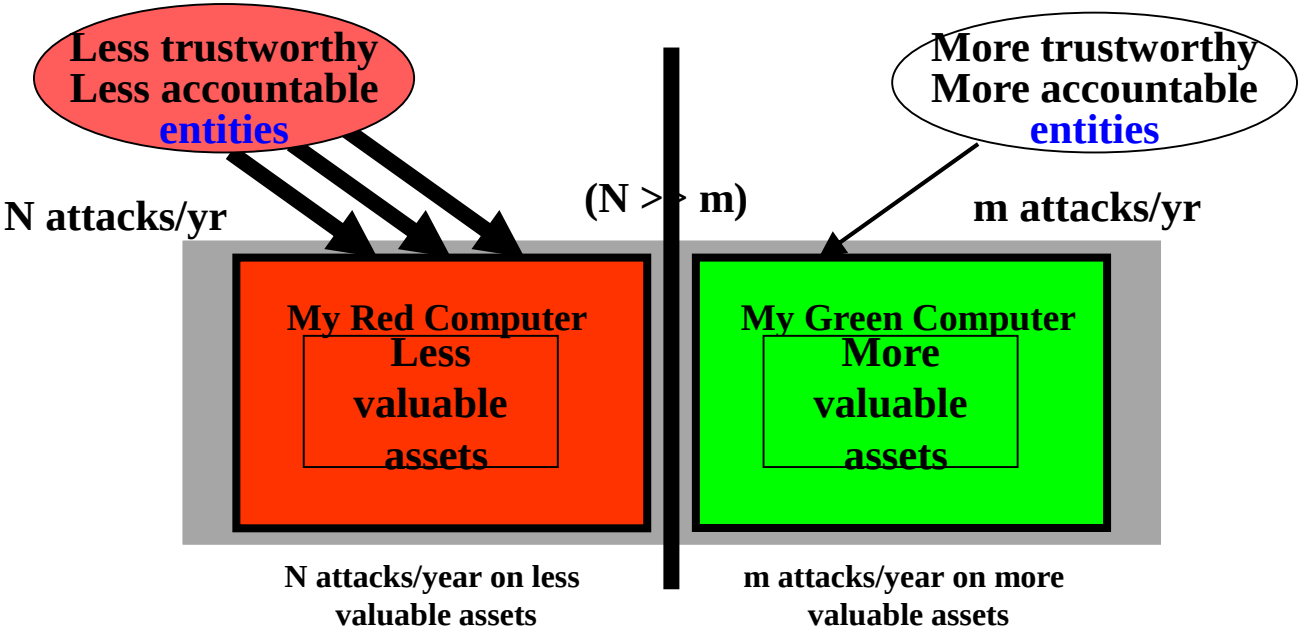
- Inputs from accountable senders are safer
  - Senders are accountable if you can **punish** them
    - With dollars, ostracism, firing, jail, ...
  - Accountability deters senders from tickling bugs
- Bad guys are not accountable
- So keep bad guys from tickling the bugs
  - Refuse inputs that aren't accountable enough
    - or strongly isolate those inputs
- End nodes enforce accountability
  - Need all the machinery of authentication and isolation
    - But coarse grained

What About Compromise?

- Stuff happens, so good guys can be compromised
  - Though less likely with accountability
  - Need careful management of accountable machines
- Second line of defense: Sanitizing
  - For each data type, define a safe subset
  - A sanitizer forces a value to be safe
  - Only accept safe inputs

What About Freedom? Red/Green

- Partition world into two parts:
  - Green: More safe/accountable
  - Red : Less safe/unaccountable
- Green world needs professional management



Why R|G?

- Problems:
  - Any OS will always be exploitable
    - The richer the OS, the more bugs
  - Need internet access to get work done, have fun
    - The internet is full of bad guys
- Solution: Isolated work environments:
  - Green: important assets, only talk to good guys
    - Don't tickle the bugs, by restricting inputs
  - Red: less important assets, talk to anybody
    - Blow away broken systems
- Good guys: more trustworthy / accountable
  - Bad guys: less trustworthy or less accountable

Data Transfer

- Mediates data transfer between machines
  - Drag / drop, Cut / paste, Shared folders
- Problems
  - Red → Green : Malware entering
  - Green → Red : Information leaking
- Possible policy
  - Allowed transfers (configurable). Examples:
    - No transfer of ".exe" from R to G
    - Only transfer ASCII text from R to G
  - Non-spoofable user intent; warning dialogs
  - Auditing
    - Synchronous virus checker; third party hooks, ...

Conclusions

- Access control hasn't worked. Learn from real-world experience.
- Security should depend mostly on retroactive, after-the-fact response
  - Work on actual problems, not hypothetical ones
- For blame and punishment: auditing and accountability
- For integrity: selective undo
- For secrecy: ownership of published data and provenance
- For bugs: isolation, accountable inputs, and red/green