Resilient Cyber Security and Privacy

Butler Lampson
Microsoft Research
Cyberforum
April 7, 2015
Security: 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
- Keep something secure when it changes
  - “When it comes to security, a change is unlikely to be an improvement.” —Doug McIlroy
- Get users to make judgments about security
Lots of hype

- Not much hard evidence of actual harm
  - As opposed to scare stories and uneasiness
  - Ex: Scale of identity theft, losses from cybercrime

- Most numbers come from interested parties
  - who are in business to sell you security stuff

- Rarely, we see business decisions backed by data
  - Verifying credit card transactions

- Most costs are in prevention, not in harm
Approaches to rational security

- Limited aspirations
  - In the real world, good security means a bank vault
    - There’s nothing like this in most computer systems
  - Requires setting priorities—what’s really important

- Retroactive security
  - React, don’t anticipate—work on actual problems
  - Deterrence and undo rather than prevention
    - Deterrence needs punishment
    - Punishment needs accountability
Deterrence, punishment, accountability

- Real world security is retroactive, about deterrence, not about 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

29 February 2016  
Lampson: Retroactive Security
Limiting aspirations: Red | Green

- Partition world into two parts:
  - **Green**: More safe/accountable
  - **Red**: Less safe/unaccountable

- Green world needs professional management

![Diagram showing Red and Green worlds with less and more valuable assets and attack rates.](image-url)
What about bugs? Control inputs

- Bugs will always subvert security
  - Can’t get rid of bugs in full-function systems
    - There’s too much code, changing too fast
    - Timeliness and functionality trump 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 or sanitize those inputs

- To control possible inputs, isolate the program
  - Airgap, VM, process isolation, sandbox
Privacy: Personal control of data

- You are empowered to **control** your data
  - Find it, limit its **use**, claim it
  - **Everywhere**—Across the whole internet
  - **Anytime**, not just when it’s collected
  - **Consistently** for all data handlers and devices
  - Remaining **anonymous** if you wish
Personal control of data: Mechanisms

**Ideal:** All your data is in a vault you control

- I bring you a query
- If you like the query, you return a result
  - Otherwise you tell me to go away

**Practical:** Data has **metadata** tag: link to policy

- Two kinds of players:
  - **Agents you choose**—like an email provider
    - Personal Agent on your device
    - Policy Service online
  - **Data handlers**, subject to regulation
    - Anyone who handles your data and follows the rules
    - Must fetch and obey your current policy
How it works

You are in control

NID+ is the metadata

(1) Set policy

Your policy service

Identity: NID

Policy:
<type, handler>→Y/N ...

(2) Provide data
data, NID+→

Handler h

Data items:
<NID +, type, bytes>
...

(3) Get policy
handler, type, NID

(4) Claim data
NID→
data items

Regulator makes rules

Your agent

Your policy service

You are in control

You are in control

Regulator makes rules

You are in control
Policy

- **Data-centric**, not device or service centric
  - Metadata stays with the data, points to data’s policy
- Standard policy is very simple
  - $7 \pm 2$ types of data: contact, location, transaction, ...
    - Can extend a type with an optional tree of subtypes
  - **Basic policy**: handler $h$ can/can’t use data type $t$
- **One screen** shows most policies (in big type)
  - **Templates** (from 3rd parties) + your exceptions
- **Encode complex policy in apps**
  - An app is a handler that tags its output suitably
Conclusions

- **Rational security**
  - Limited aspirations
    - Red | Green
  - Retroactive security
    - React—work on actual problems
    - Deterrence and undo over prevention

- **Personal control of data**
  - Data tagged with metadata: a link to your policy
  - Handlers must obey policy
Backup
Access Control

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

![Diagram](attachment:image.png)

- **Source**: Agent / Principal
- **Request**
- **Guard/Reference monitor**
- **Resource/Object**
- **Sink**: Audit log

1. Isolation boundary
2. Access control
3. Policy

Host (CLR, kernel, hardware, VMM, ...)

Lampson: Retroactive Security
Incentives

- **Perceived** threat of harm, or regulation
  - Harm: loss of money or reputation
  - For vendors, customer demand, which is weak

- Perception is based on *past experience*
  - not on possible futures
  - because too many things might go wrong
  - and you’ll have a different job by then

- Regulation is a blunt instrument
  - slow, behind changing technology and threats
  - expensive
  - prone to unintended consequences.
  - But it can work. Ex: US state laws on PII disclosure
Are people irrational? No

- Goals are unrealistic, ignoring:
  - What is technically possible
  - What users will actually do
  - Conflicting desires for
    - security, anonymity, convenience, features

- Actual damage is small
  - Evidence of damage is weak
  - Hence not much customer demand

- Incentives are lacking
  - Experience trumps imagination
  - Convenience trumps security
  - Externalities: who benefits ≠ who pays
What is technically possible?

- Security requires simplicity
- Most processes add complexity
  - SSL/TLS recently discovered bugs
  - EMV chip-and-PIN system
  - Windows printing system
  - SET “standard” for internet credit card transactions
- “Too complex” is a judgment call
  - Why? No good metrics for complexity or security
  - So desire outruns performance
What will users actually do?

What gets the job done
- Disabling or evading security in the process

What is easy
- 2-factor auth for banking → password + device
  - But in Norway, one time passwords for banking

What works everywhere
- For security, that’s nothing
- So “educating” users doesn’t work

What solves a problem they (or a friend) actually had

“If you want security, you must be prepared for inconvenience.”