Carnegie Mellon

CyLab
www.cylab.cmu.edu

Dancing with the Adversary: a tale of Wimps & Giants

Virgil D. Gligor
Carnegie Mellon University
Pittsburgh, PA 15213
gligor@cmu.edu

Layered Assurance Workshop
30th ACSAC
New Orleans
December 9, 2014

Outline

Challenge 1: Business case for **high assurance** in ***commodity*** operating systems?

Challenge 2: Adversary definitions

- **compete** and **correct**

=> separation and composition ("dance") of **Wimps** & **Giants**

Challenge 3: Composition of adversary definitions

- adversary anatomy and basic metrics

Extending the Wimp-Giant metaphor to Human Protocols

- can a **User** retain personal control of her private data after outputting it to a **Giant**?

Challenge 1: Commodity systems

Carnegie Mellon
CyLab
www.cylab.cmu.edu

Commodity Software Markets -> **Rapid Innovation** -> **Low/No-Assurance Software**

Some Characteristics
cost of entry ≈ **0**
regulation ≈ **0**
liability ≈ **0**

Producers
- high measured productivity
  e.g., lots of S/W functions, apps,
- few configuration restrictions
  e.g., use others' code

Consumers
- low price
- mass access to lots of functions and apps

High Assurance?
- high latency, opportunity cost
  => fewer functions
- strict configuration control
  (provenance ctrl: cannot use others' unverified code)

- higher consumer cost
  e.g., maintenance cost

Niche Software Markets

(at most one "0"s
e.g., aerospace, defense, vehicular, IPC systems)

Challenge 2: Adversary definitions

Carnegie Mellon
CyLab
www.cylab.cmu.edu

A system *without* an Adversary definition
possibly be (*in*)*secure*; it can only be as

⇒ System security is meaningless without
complete and *correct* Adversary definitions

Conversely,
complete and correct Adversary definitions
=> security properties: counter *all* Adversary *behaviors*
& deny any advantage over defender

+ Consequence:
metrics and cost of security become possible
e.g., assurance: evidence of how well security
properties counter Adversary's behaviors

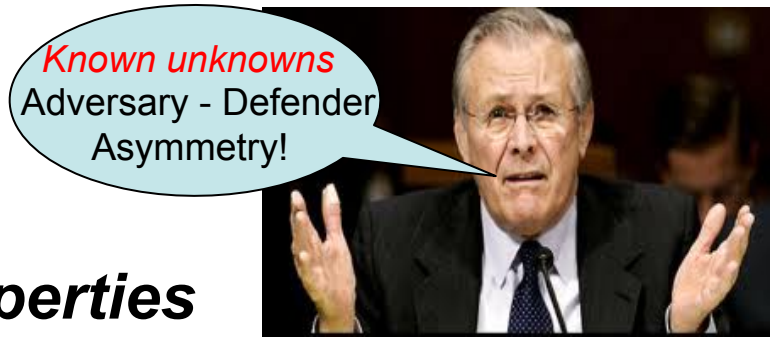
Complete & Correct? How?

Carnegie Mellon
CyLab
www.cylab.cmu.edu

- **Find *all vulnerabilities* in *all* types of system components**
  => a chance to define all attacks; e.g., crypto, OS, net, apps, human protocols

- **Define *all attacks* enabled by each type of vulnerability and combinations thereof**
  => a chance to define all attack strategies; aka. adversary behaviors

- **Define *all* Attack *strategies***
  => a chance to define security properties



- **Define *all necessary security properties***
  => a chance to define security assurances, metrics, and costs

Incomplete or Incorrect Adversary Definitions => *Persistent* Adversary Advantage

Complete & Correct? Possible in Practice?

Carnegie Mellon
CyLab
www.cylab.cmu.edu

1. **No, not for *commodity* systems. They will *always* have some functionally rich, complex, large & perhaps diverse-origin components ("giants")**

Why not?

1. "3 zeros" -> *rapid innovation* -> software ***giants*** [SPW'10]

2. *"Software components: only the **giants** survive"* [Lampson04]

=> opportunity cost of high assurance (e.g., time, money, talent) is prohibitive;
EAL6 ~ $10K/SLoC [SOSP09]

=> Adversary will *always* have an advantage over Defender

Complete & Correct? Possible in Practice?

Carnegie Mellon
CyLab
www.cylab.cmu.edu

1. No, not for *commodity* systems. They will *always* have some functionally rich, complex, large & perhaps diverse origin components ("giants")

2. Yes. *But only* for comparatively small, relatively simple, functionally-limited components ("wimps")

How do we design wimps?

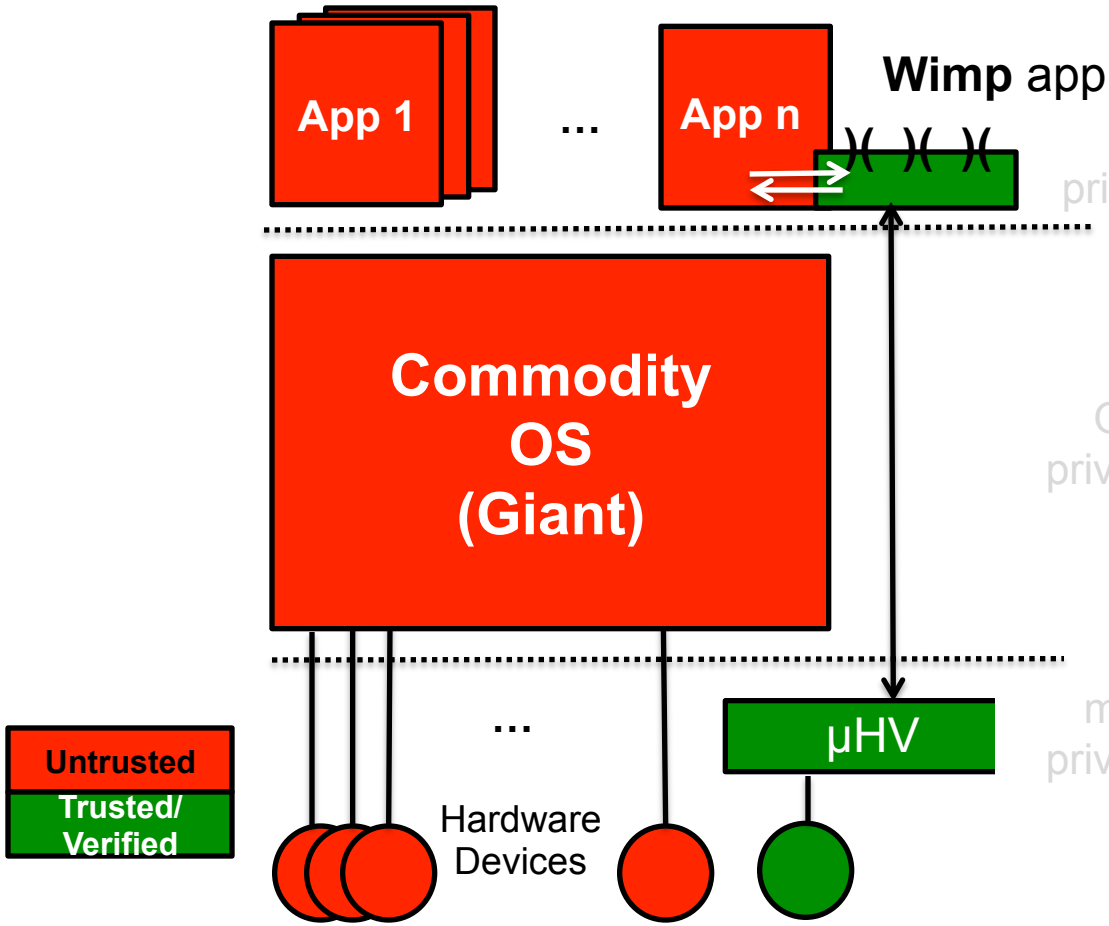
An example – components form **giant**:

- decrease their size & complexity until adversary
  - can be completely & correctly defined; has very small attack surfaces
- security properties = negations of "adversary behaviors"
- cost of high assurance is no longer prohibitive

=> Adversary does *not* have any advantage over Defender

Insecure Giants need Secure Wimps

- architectures for **Wimp separation** exist: e.g., hypervisors, security kernels, separation kernels, micro-hypervisors (μHV)**; + secure channels**



e.g.,

crypto libraries

- *key management*
  - SKs of web servers/CAs
- *secure crypto channels*
  - channel isolation

app reference monitors

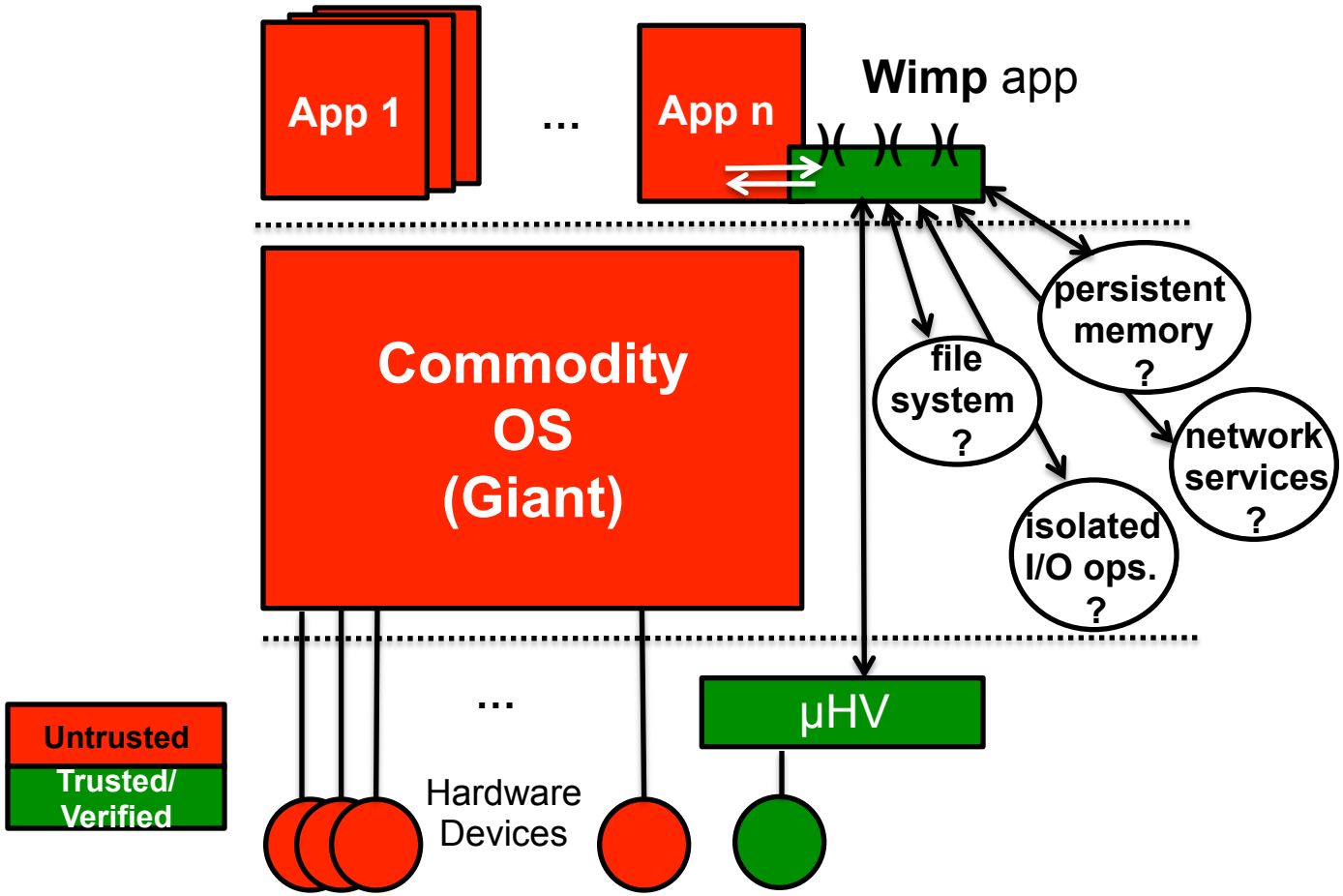
- isolated execution
- access policies

integrity checking

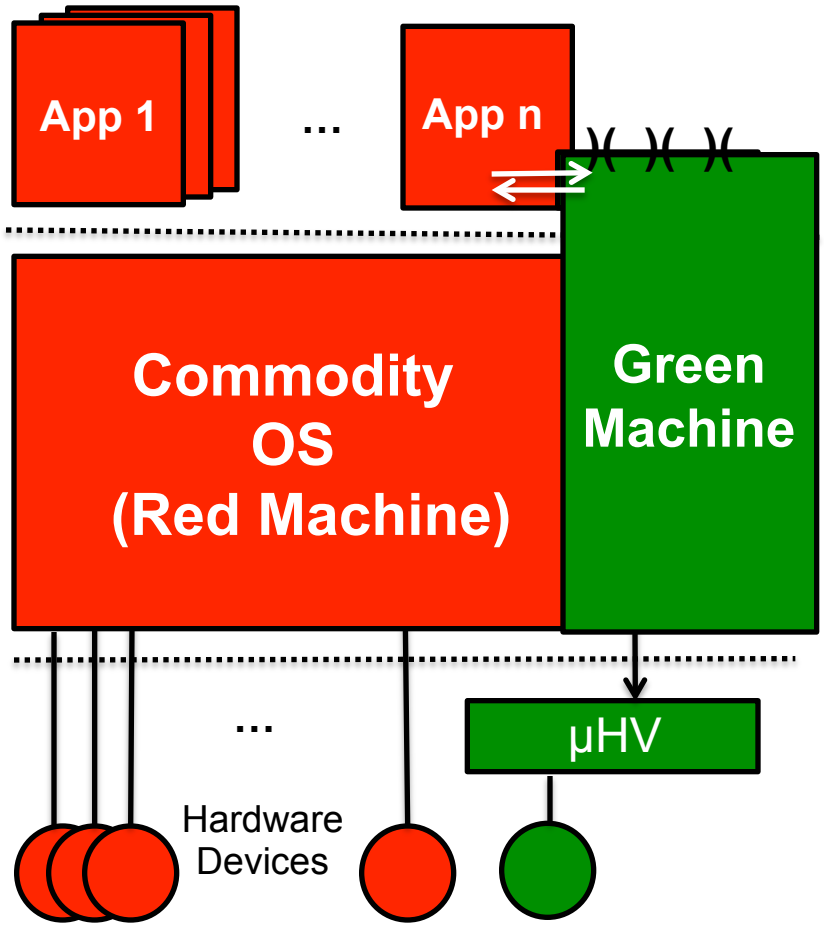
- **giant** services

e.g., **TrustVisor** & **XMHF**
[IEEE S&P '10 & S&P '13]

Problem: Wimps lack basic services

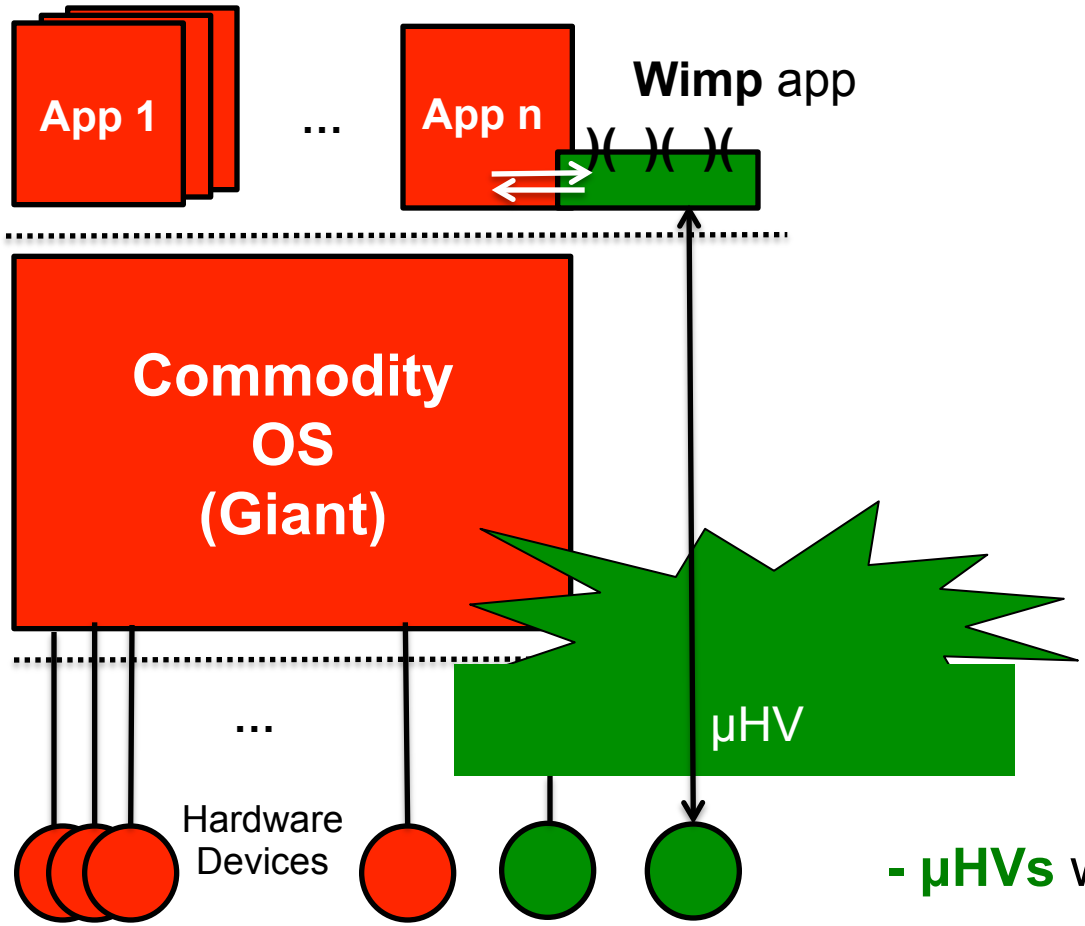


Adding Services to Wimps is Undesirable



- **Wimps** would become **Giants**
viz., Lampson's **Red**-**Green** machine
[CACM Nov. '09, SPW '10]

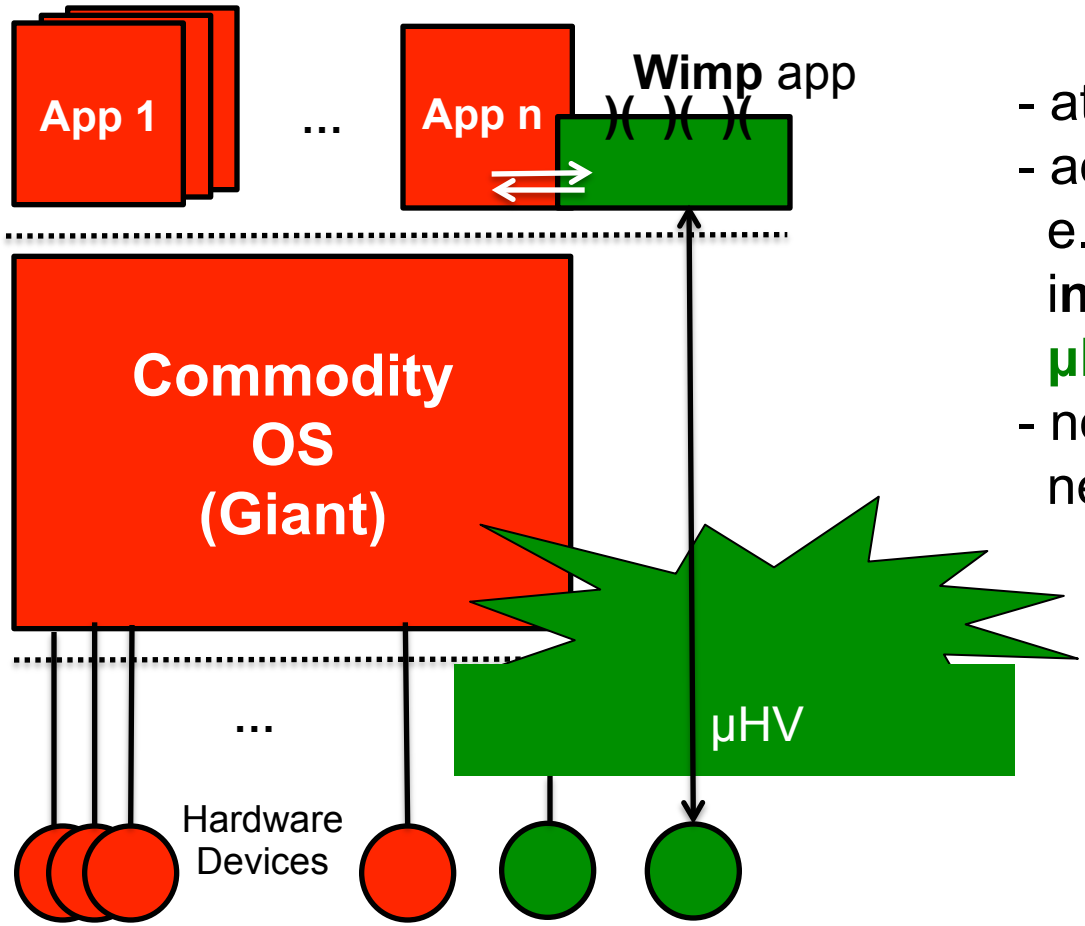
Adding Services to µHV is Undesirable



- µHVs would become Giants

Adding Services to µHV is Undesirable

Example: adding **USB** service to micro-hypervisor (**µHV**)

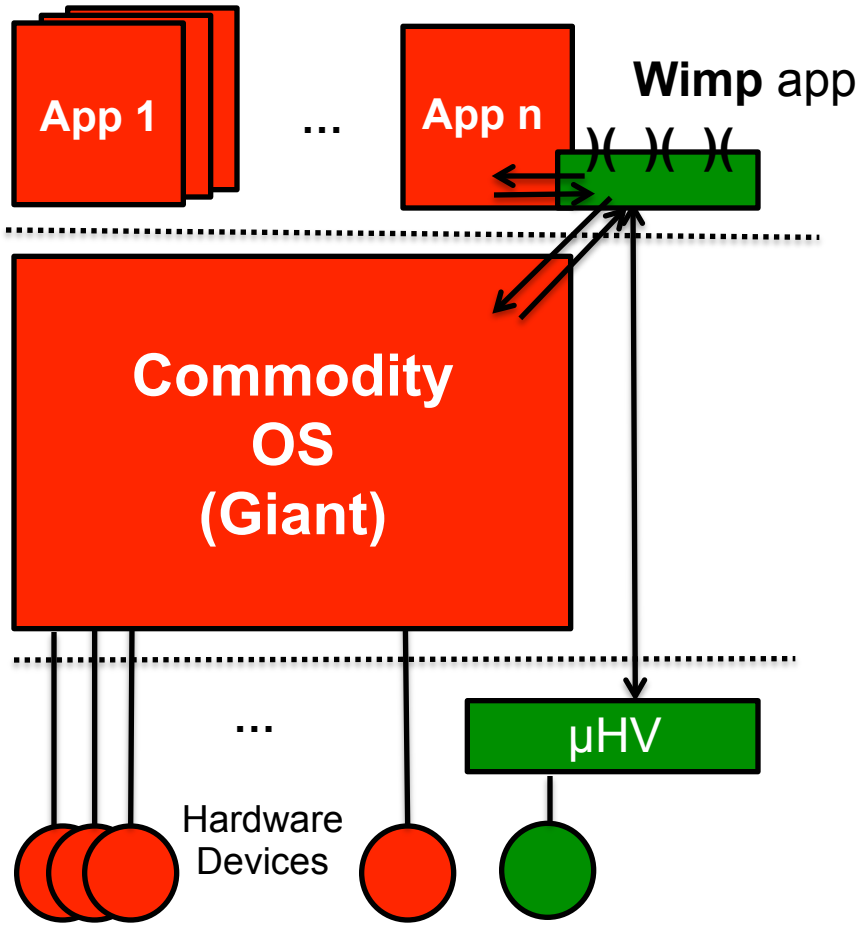


- at least **doubles size**
- adds **complexity**; e.g., interrupt handling **invalidates proofs** of **µHV** isolation
- not all **Wimp** apps need **USB** service/devs

+ other I/O services? e.g., PCI, Bluetooth, HDMI, Firewire . . .

"Paradox:" Secure Wimps need Giants

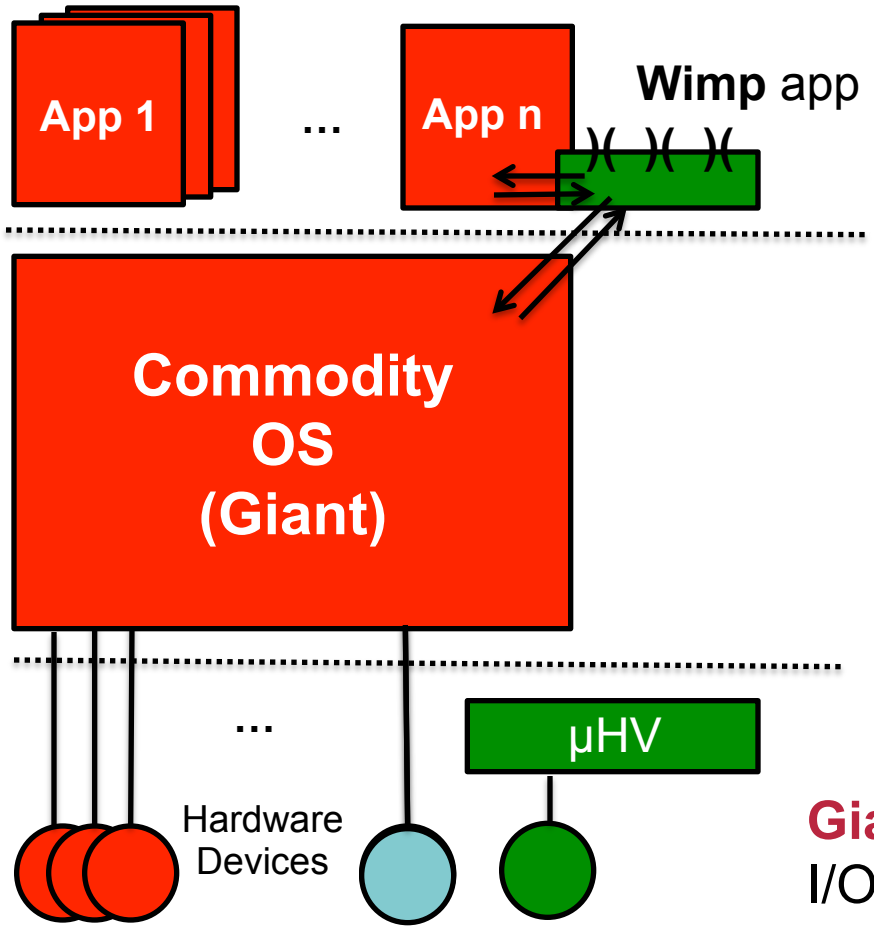
Example: Export Wimp objects to Giants



- **Files, VM pages** [GL79]
- **Database relations** [Denning84, Graubart84]
- **Network messages** [Chen08]
- **Persistent storage** [Parno11]

"Paradox:" Secure Wimps need Giants

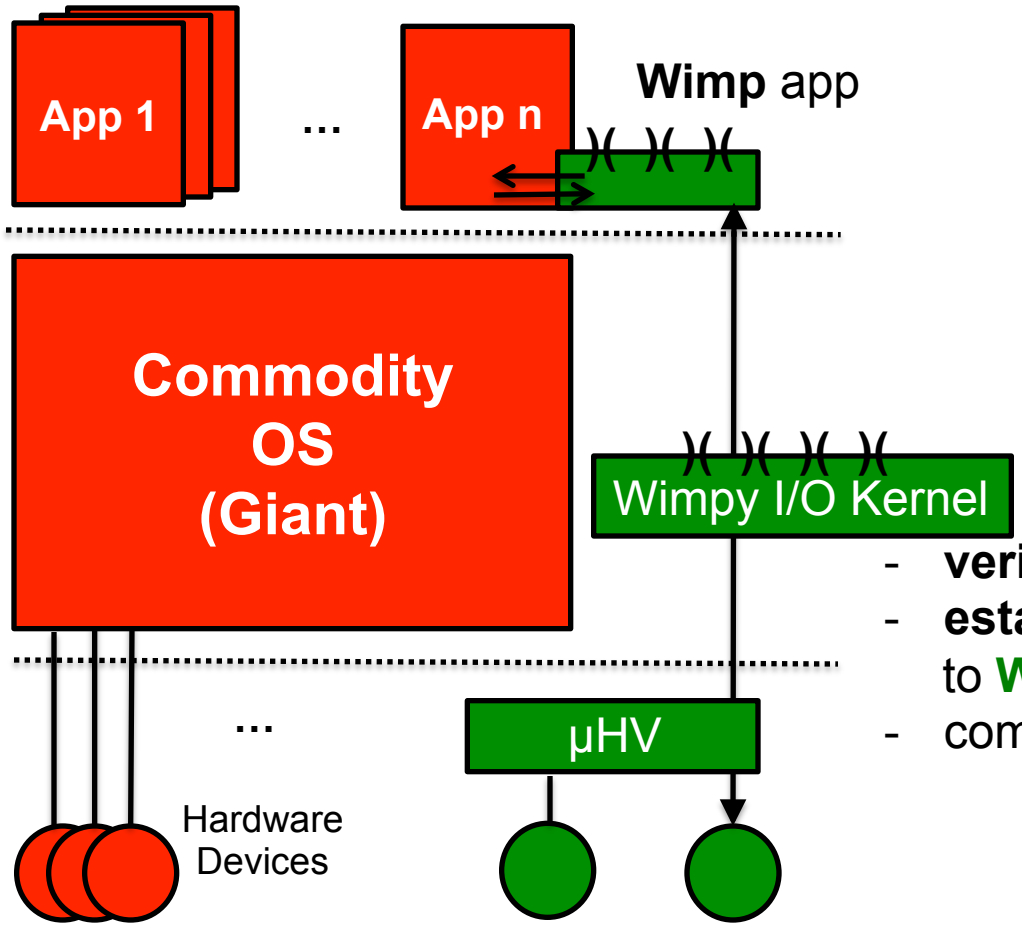
Example: Export **Giant's devices** to **Wimps** on-demand



Giant initializes and **configures** I/O device demanded by **Wimp**

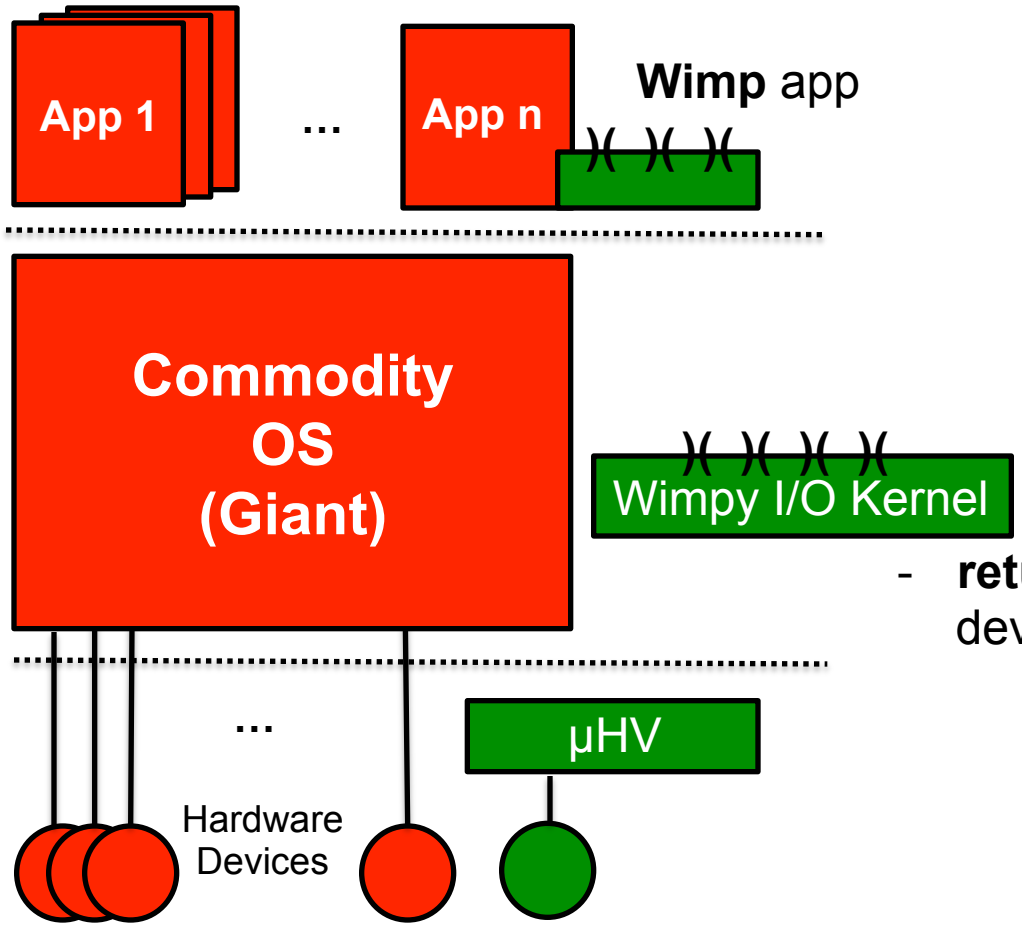
Export **Giant's devices** to **Wimps** on-demand

Need a **Wimpy I/O Kernel**



- **verifies** the I/O configurations
- **establishes isolated channel** to **Wimp** app
- composes with the **μHV**

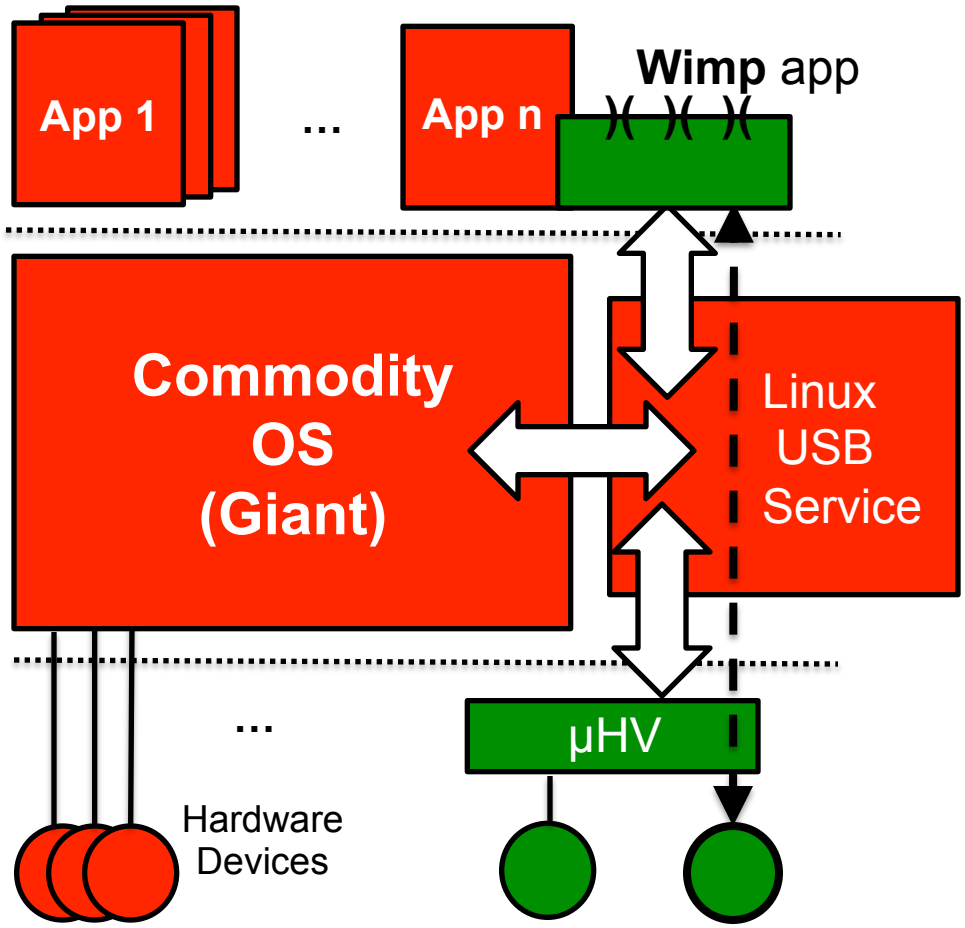
Ex: Export **Giant's devices** to **Wimps** on-demand



- **returns control** over the device back to **Giant**

How to Design a Wimpy I/O Kernel?

Example: **Linux USB** service => break it off from the **Giant**

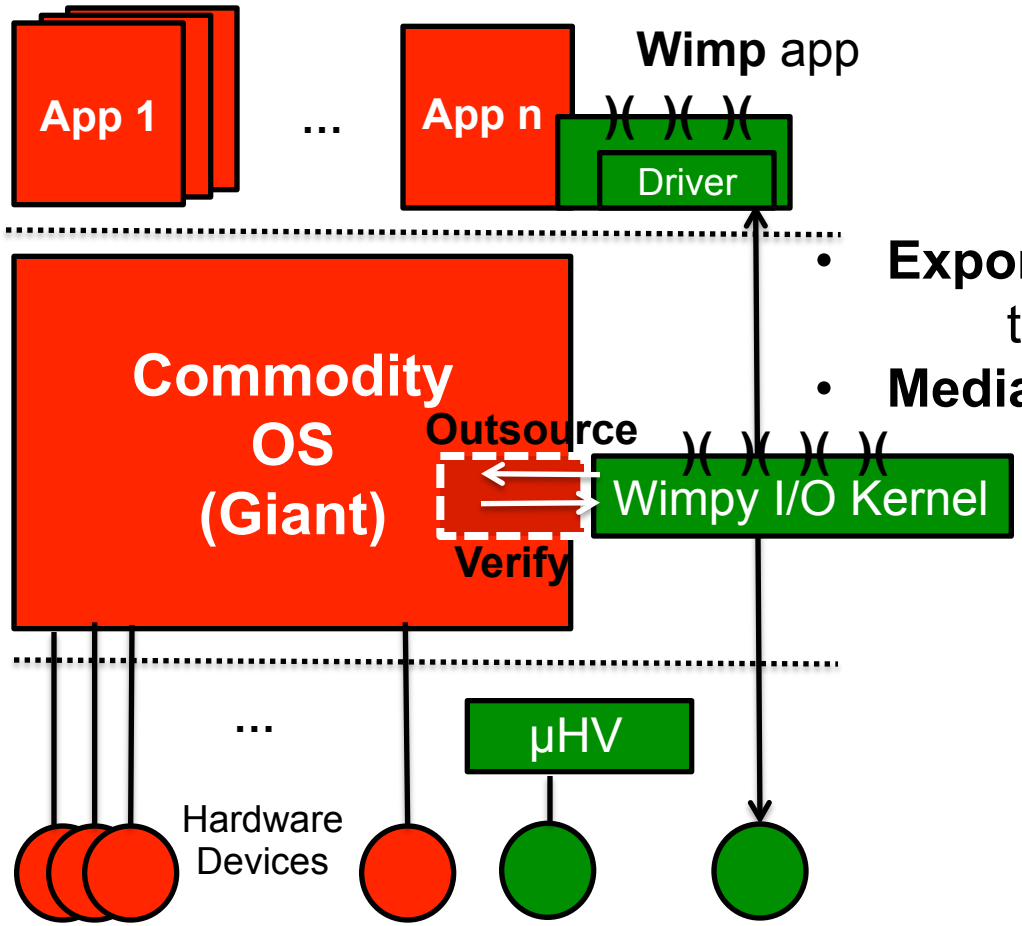


large & complex
- core ~ 19,820 SLoC
- drivers ~ 206,376 SLoC

- **unverifiable**
  I/O channel isolation
- **cannot be composed**
  with **µHV**

How to Design a Wimpy I/O Kernel?

Decrease Linux USB Service Size & Complexity until it becomes a **Wimpy I/O Kernel**



- **Export** driver & I/O operations to **Wimps** (ex. 1974 ->)
- **Mediate** USB requests

Code Reduction Results of USB Subsystem

USB 2.0 Code (SLoC)

in Wimpy Kernel					in Linux				
Bus Verification		USB Host & Hub Drivers		Request Mediation	Total	USB Core		USB Device Drivers		Total
93	+	1844	+	107	= 2,144	19,820	+	206,376	=	226,196

Wimpy Kernel:

- uses **a few standard** operations of USB **host controllers and hubs**
- **excludes drivers of other devices** that share same bus

  - size of request mediation code is small
  - over **99%** of reduction for USB 2.0 code base

Challenge 3: Composition of Adversary Definitions

Carnegie Mellon
CyLab
www.cylab.cmu.edu

1. **Why?**
   … Cryptographic Protocols, OSes, Nets, Apps, …, Human Protocols

2. **What do we compose?**
   - uniform adversary (and attack) anatomy

3. **How do we measure the composition result?**
   - basic metrics

Examples: Operating Systems

Carnegie Mellon
CyLab
www.cylab.cmu.edu

1. OS kernel corruption
2. crash the system
3. access any file

1-2. invoke ***internal*** *(unexposed)* ***kernel*** function

3. execute w/ "root" privileges → (***intermediate goal***)
"buffer overflow"
→ …

<Goal Capabilities Strategies>

{Attacks} ⇒ **System Interface**

Power & Privilege

user processes
- types of calls
  how many, how fast,
- types of storage, etc.
  how much, how fast

Capabilities to
Attack Surface
- **E**ntry **P**oints: calls, instructions
- **D**ata **I**nput: parameter sets
- **Ch**annel: invocation method

Size: single/multiple **EP**s? Restrictions?

Strategies: Capabilities → Goal
- all exploitable calls?
- all parameter combinations?
- concurrency?
- . . .

Examples: Cryptographic Modes

Carnegie Mellon
CyLab
www.cylab.cmu.edu

1. distinguish encryptions
2. forge auth. tags, ciphertexts
- . . .

1 - 2. Find Enc/Dec **oracles in crypto protocols** for verifiable/predictable/known/chosen plaintext … chosen ciphertext, & **oracle invocation**

<Goal Capabilities Strategies>

{Attacks} ⇒ **Protocol Interfaces**

Power & Privilege

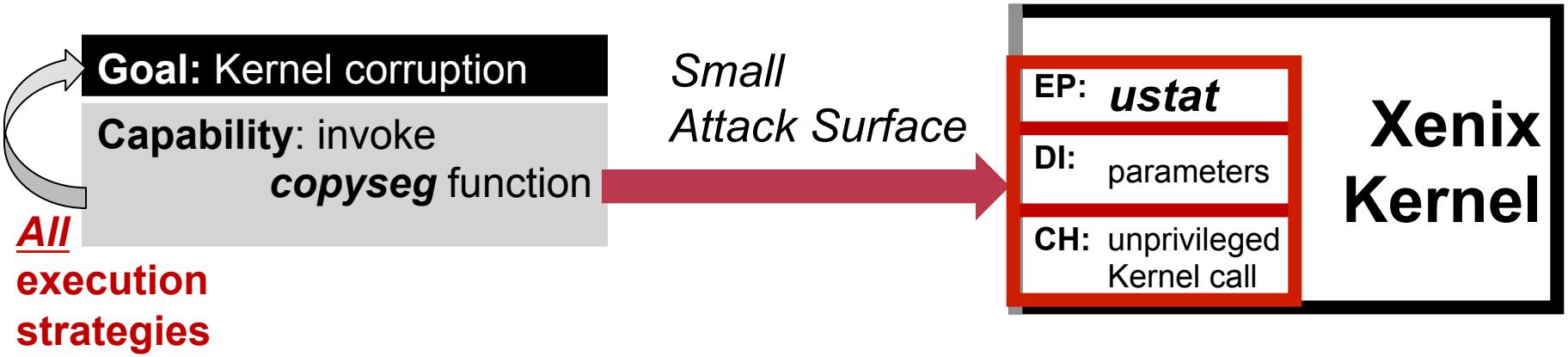
PPT RAM
- types of operations,
  how many, how fast,
- types of storage, etc.
  how much, how fast

Capability to
Attack Surface
- **E**ntry **P**oints: Enc/Dec **oracles**
- **D**ata **I**nput: verifiable/known/chosen…
- **Ch**annel: invocation methods

Size: single/multiple **EP**s? Restrictions

Strategies: Capabilities → Goal
- adaptive?
- interactive?
- concurrent?
- . . .

Old Examples: *All* Strategies are Found



All **execution strategies**

1 unprivileged system call (of **110**),
- all parameter combinations

2 independent kernel code paths

Old Examples: *All* Strategies are Found

Goal: crash system repeatedly

Capability: invoke ***panic*** function

All **execution strategies**

Large Attack *Surface*

EP: ***ustat***
DI: parameters
CH: unprivileged Kernel call

⋮

EP: …
DI: parameters
CH: unprivileged Kernel call

Xenix Kernel

[1989]

38 system calls
- all parameter combinations for each call

15 independent kernel code paths

Giants: *Not* All Strategies are Found

Carnegie Mellon
CyLab
www.cylab.cmu.edu

Goal$_0$: access any user's file

Capability: control-flow hijacking to get root access

All **? strategies**

Goal$_1$: buffer overflow

Capability: modify return address

All **? strategies**

System Interfaces

EP:	***rmdir***
DI:	fname, ret. address
CH:	unprivileged Kernel call

Unix
[1989]

EP:	***fingerd***
DI:	. . . ret. address
CH:	unprivileged Kernel call

[1988]

. . . .

statd
[1996]

. . . . .

Wimp's Security Properties

set of program behaviors that counters ***all*** attacks

- ***deny*** *execution* ***strategies*** *or* ***capabilities (=> deny goals)***
  Example: verified penetration-resistance; access control; authentication
  Example: verifiable execution; roots of trust
- **deter** an adversary
  Example: audit policies => punishment
- **limit** adversary success
  Example: recover secure system states (and force attack retry);
  undo; insurance

Example: Insiders

- authorized, by definition
  - **deny** execution strategies and **limit** success by **separation of duties**
  - **deter** the insider by **continuous, real-time, audit analytics**

Wimp's Security Properties

Basic Metrics based on Partial Orders

Attack$_1$
<Goal$_1$, Capabilities$_1$, Strategies$_1$>

$>$ <=> $\not\approx$

Attack$_2$
<Goal$_2$, Capabilities$_2$, Strategies$_2$>

Implication (**A_1 => A_2**):
System security properties that counter attack **A_2** also counter attack **A_1**

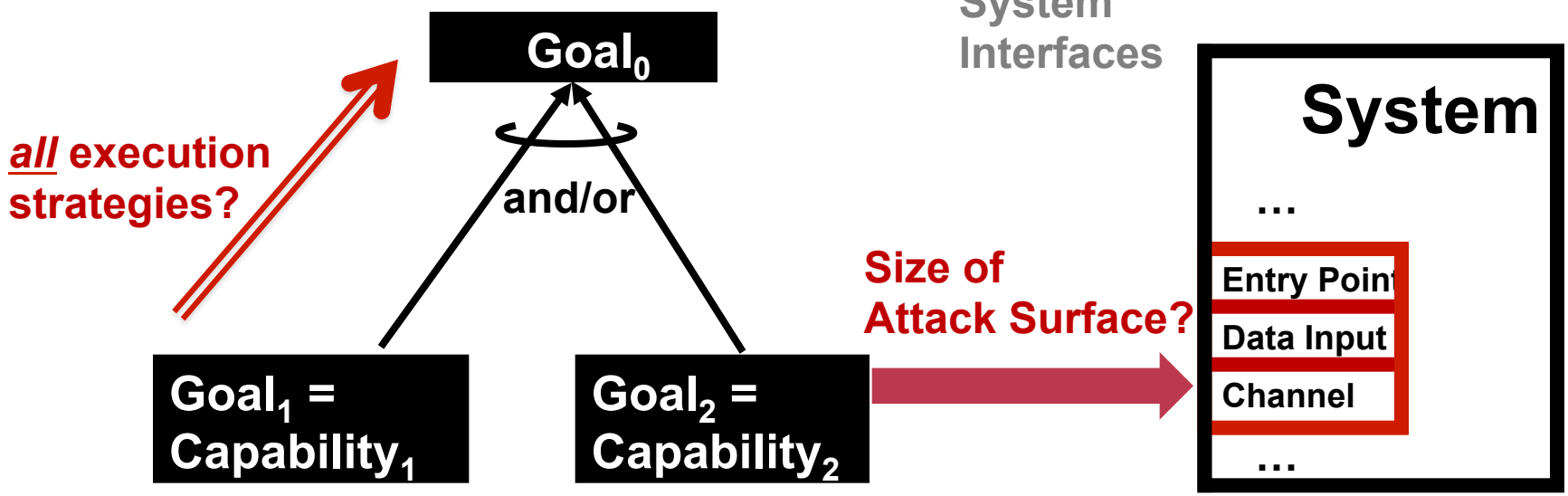
Separation (**A_2 ≠> A_1**):
Not all system security properties that counter attack **A_1** also counter attack **A_2**

Strict Dominance: $(A_1 > A_2) = (A_1 \Rightarrow A_2) \wedge (A_2 \not\Rightarrow A_1)$
Equivalence: $(A_1 \Leftrightarrow A_2) = (A_1 \Rightarrow A_2) \wedge (A_2 \Rightarrow A_1)$
Incomparability: $(A_1 \not\approx A_2) = (A_1 \not\Rightarrow A_2) \wedge (A_2 \not\Rightarrow A_1)$

Traditional Attack Representations

Carnegie Mellon
CyLab
www.cylab.cmu.edu

- attack tree



J. D. Weiss ('91), E. Amoroso ('94), B. Schneier ('99), and Swiderski & Snyder ('04), Mauw and Oosdijk ('05), and **attack-defense trees** Kordy et al ('10), etc.

Missing: *All* Execution Strategies (capability ordering, dependencies, timing)
: **Attack-Surface *Minimization* concern**
=> OK techniques, but not useful for defining Wimps

Traditional Attack Representations

Carnegie Mellon
CyLab
www.cylab.cmu.edu

"kill chain"	Goal Example: Get Root Access to a system's sensitive data
1. Reconnaissance	Discover **Entry Points (E)**: *exposed* system/app interfaces (APIs), *unexposed* interfaces/instructions, network protocol oracles, user interfaces, insiders
2. Weaponization	Malicious **Data Input (I)**; e.g., content of instruction fields/registers, parameters, messages, binaries, scripts, email attachments, links
3. Delivery	Instruction/System/App/Protocol **Channel (C)**; e.g., Spear phishing email
4. Exploitation	Use discovered **Vulnerability**; e.g., Target (i.e., System/App/Protocol/ User) fails to/cannot validate malicious data input; control flow highjacking; elevation of privilege
5. Installation	**Exploit:** Target inputs malicious data & acts on it: e.g., overwrites critical OS kernel data; executes malicious code/script, clicks on a link & downloads malicious payload
6. Execution	**Command & Control**; Centralized; e.g., local or remote access tool) Decentralized; e.g., P2P tool

Legend: "kill chain" = 1 attack surface(E,I,C) 1-3 + 1 execution strategy (4-6)

Wimps and Giants in Human Protocols

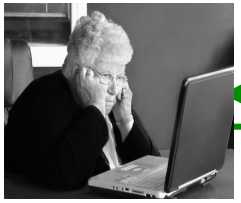
Carnegie Mellon

CyLab
www.cylab.cmu.edu

Certificates
- ID, group, role …

Attributes . . .

Action: click & invest, send credit card, account no., request for code …
Service: receive money, adds, code, …



Sender's **ID, Credentials**
Take Action => Receive Service

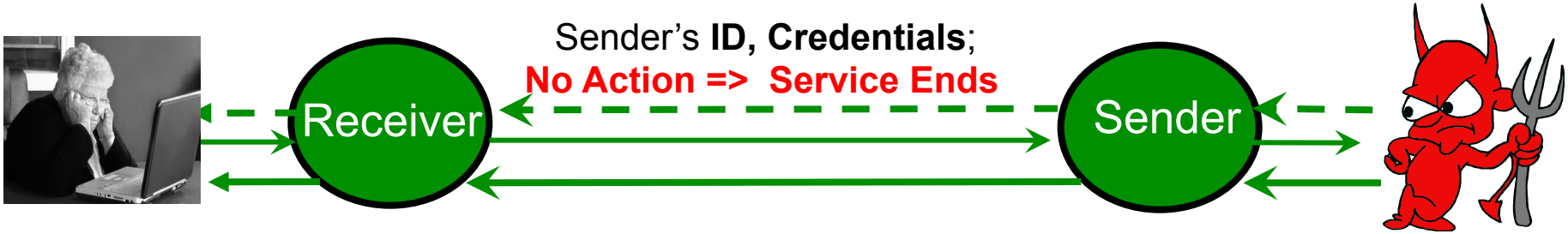
Receiver

Sender

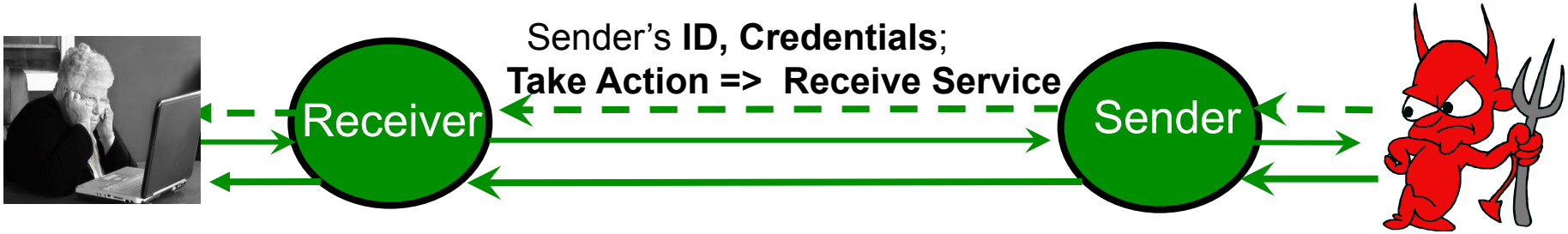
Either **Takes Specified Action** or **Rejects**

Verifies **Receiver's Action;**
Provides Specified Service or **Does Not**

Human Protocols: Wimps and Giants



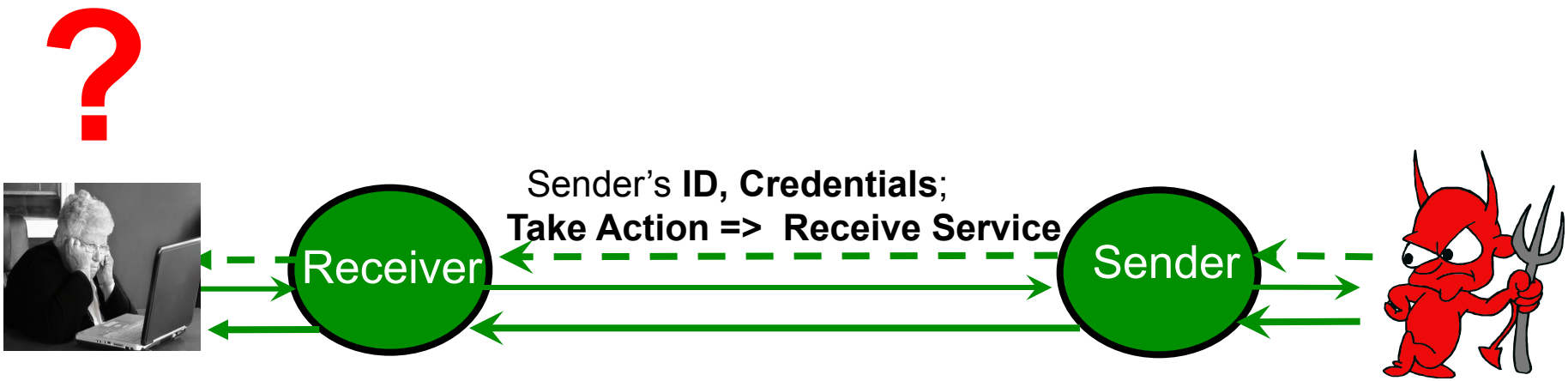
Human Protocols: Wimps and Giants



Value Proposition

- ***Both parties are honest => Both are <u>better off</u>*** **after session**
- **Future sessions: rational Receiver Takes Action again**

Inputting Data from Giants



Should I accept this ***unknown*** Sender's **Certificate?**

Typical non-answers . . .

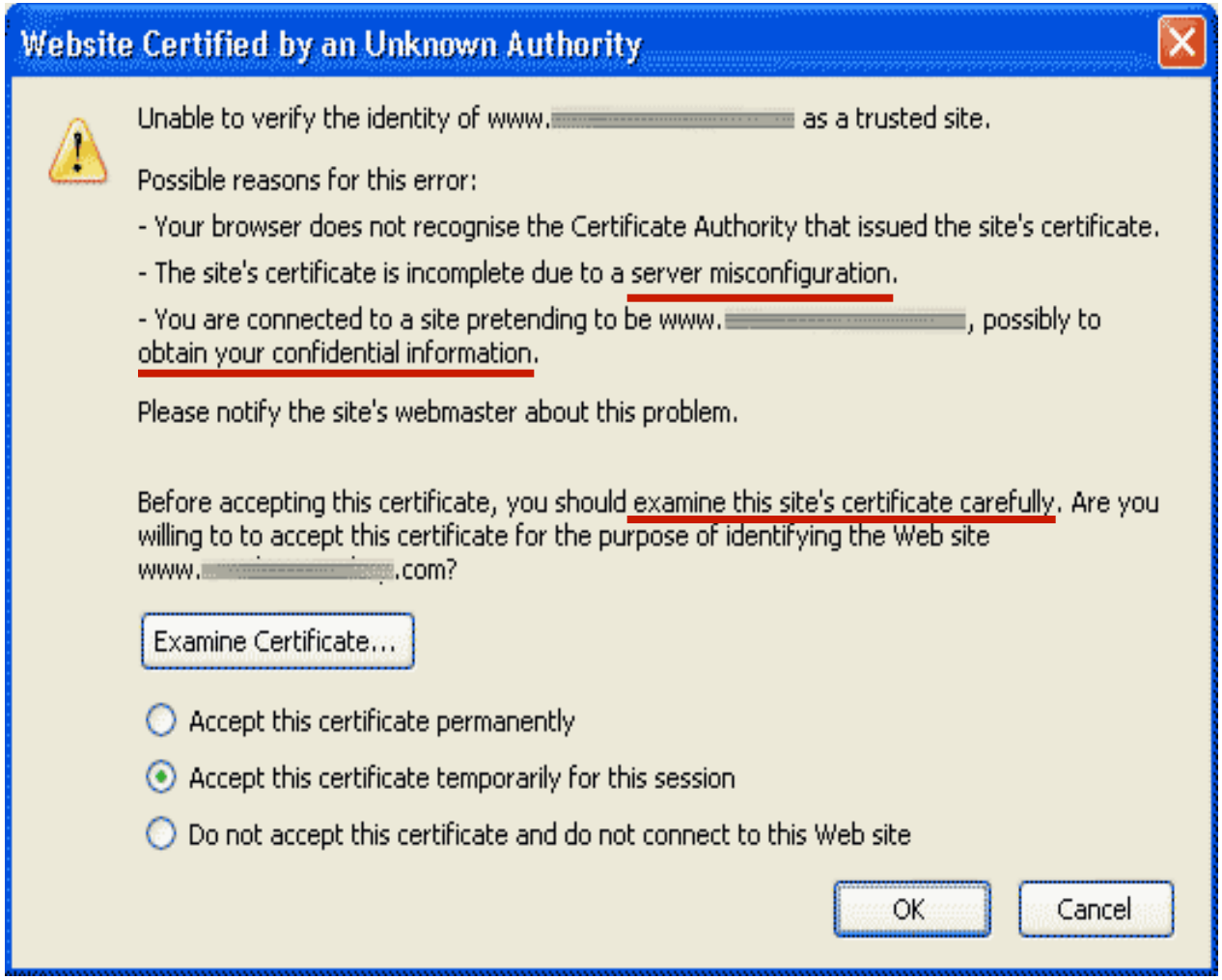
Uninformative

Good, Useless Advice

Typical non-answers . . .

Speculative, adds a little fear

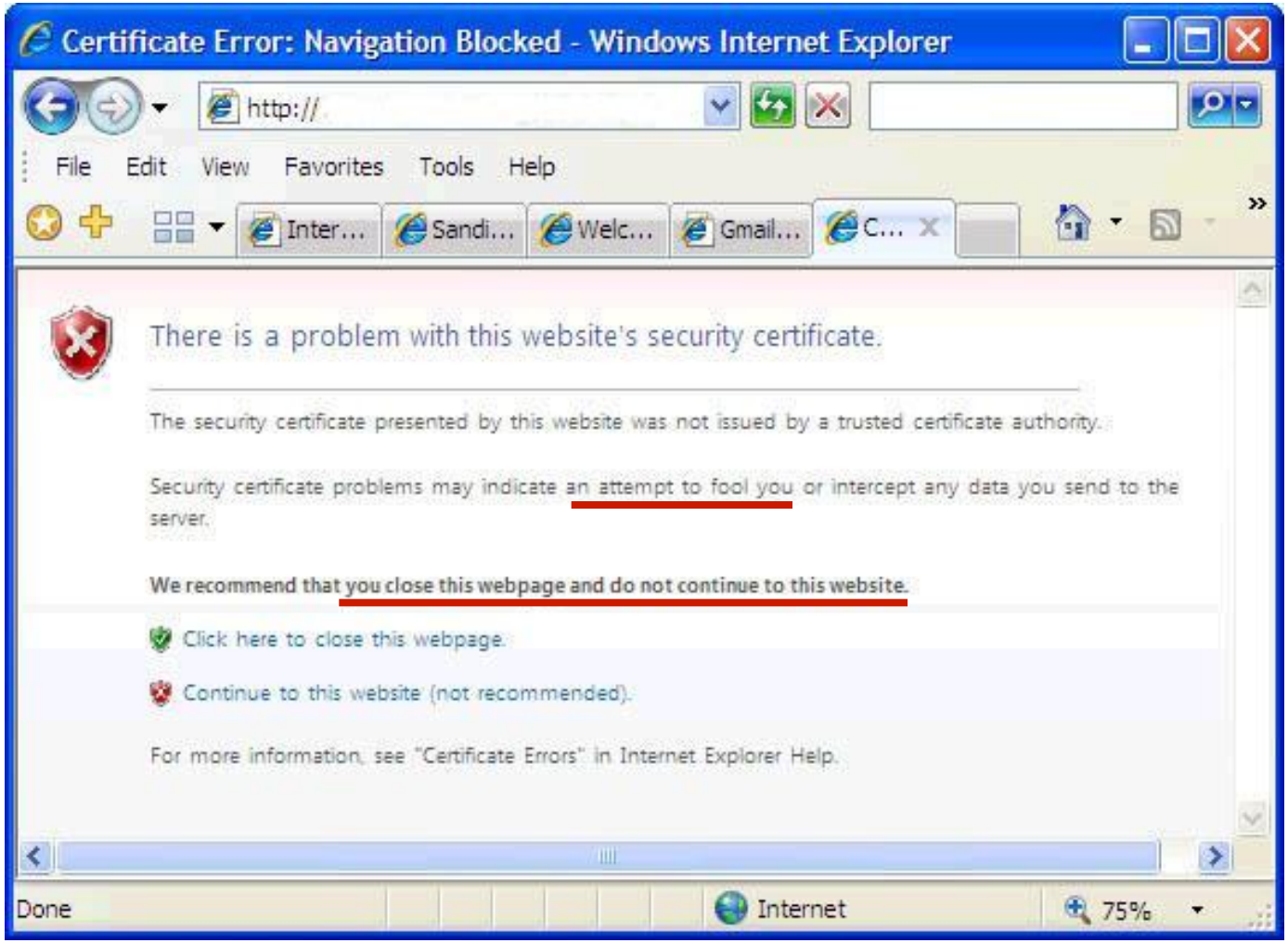
Good, Useless Advice



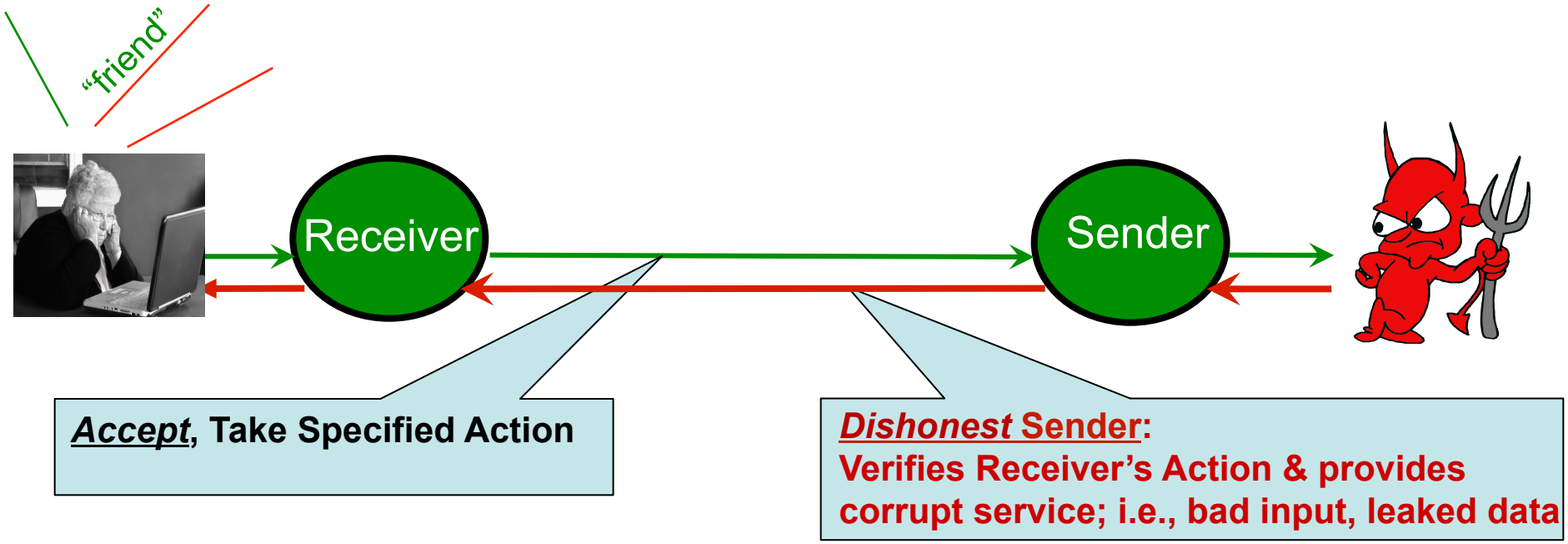
Typical non-answers . . .

Ego busting

"Just say No"



Human Protocols: Wimps and Giants



Asymmetry

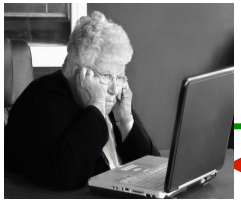
- *Dishonest* Sender is *better off than an Honest Sender* &
  Honest Receiver is *worse off* after session
- *Dishonesty* discovered => rational Receiver rejects *all* future sessions

Asymmetry Reduction?

Carnegie Mellon
CyLab
www.cylab.cmu.edu

~~Isolation~~ from Sender:
verify input, output compliance

~~Trustworthiness~~ Evidence (past)
never sent bad input, leaked output



Receiver

Sender

~~Recovery~~ from
bad ~~input~~, leaked output

Deterrence against
sending bad input, leaking output
=> ~~Punishment~~ (Regulation, Liability)
⇒ ~~Accountability~~
- Trusted Third Parties (TTPs)?

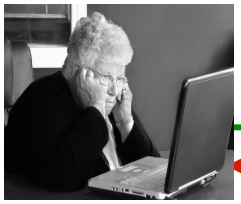
Machine?	Human?
Correct **Sender** & Machine Code	Correct **Sender** & **User Behavior**

Closure

Carnegie Mellon

CyLab
www.cylab.cmu.edu

No Behavioral Trust => **Isolation**



Receiver

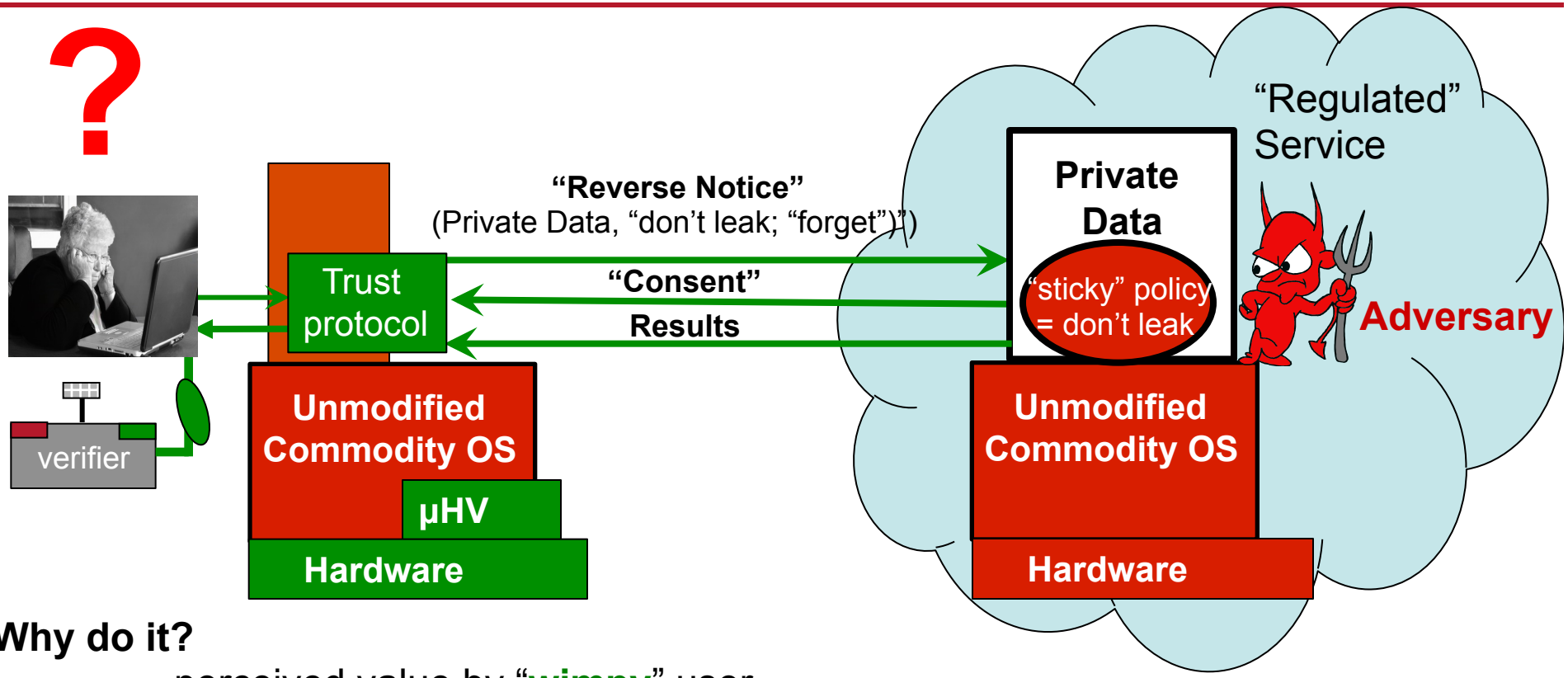
Sender

Behavioral Trust

- **Beliefs**
  - **trustworthiness** => **Trustworthiness Evidence (future)**
- **Preferences/Aversions**
  - **Risk** => **Recovery**
  - **Betrayal** => **Deterrence =>punishment =>*accountability***

we need: <= <=

Outputting Private Data to Giants



Why do it?

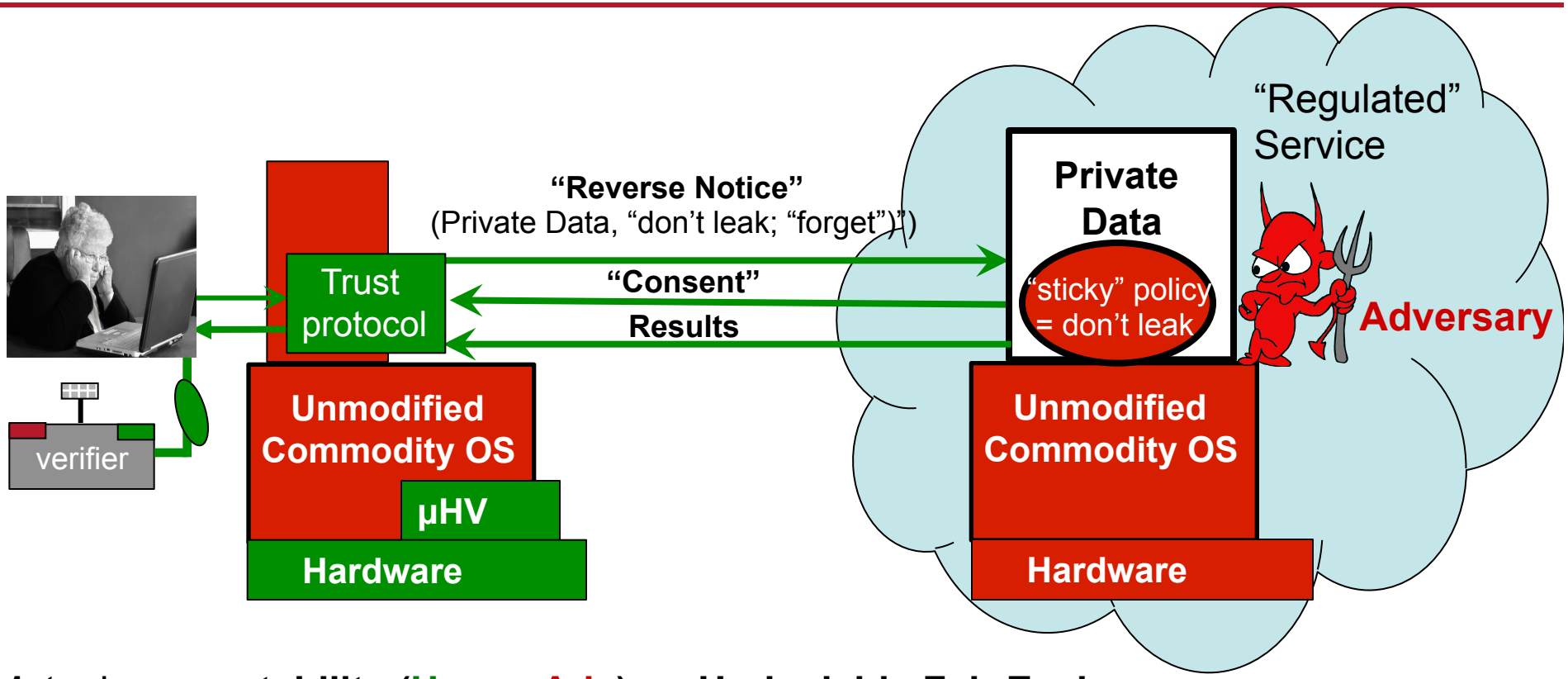
- perceived value by "**wimpy**" user

Problem

- asymmetric protocol => loss of personal control over sensitive data
- asymmetry elimination not possible with security & crypto methods alone
  isolation (e.g., outsource-and-verify) no longer works

Challenge: are there any "safe states?" e.g., Adversary can but doesn't leak user's data

Outputting Private Data to Giants



Mutual **accountability (Users, Adv) => Undeniable Fair Exchange**

Users *cannot* Trust Adv: *accountability* + *regulation* **≠>** *deterrence* of Priv. Data Misuse, returning Results + Malware

Users *can* Trust Adv: *accountability* + *regulation*: reduce **User's Betrayal Aversion**
anonymity, unlinkability: reduce **User's Risk Aversion**
recomm., repute: increase **User's Beliefs of Trustworthiness**

Summary

1. Correct and Complete Adversary Definitions in *Commodity* Systems => Wimps

2. Composition of Wimps and Giants becomes necessary

3. Uniform Adversary Definition for Wimps

4. Human Protocols

e.g., interactive trust protocols, social collateral models, deterrence

- automated detection of scams, deception, manipulation
- models of trust networks