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

<table>
<thead>
<tr>
<th>Commodity Software Markets</th>
<th>Rapid Innovation</th>
<th>Low/No-Assurance Software</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Producers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost of entry ≈ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>regulation ≈ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>liability ≈ 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- high measured productivity</td>
<td>high latency, opportunity cost</td>
<td></td>
</tr>
<tr>
<td>e.g., lots of S/W functions,</td>
<td>=&gt; fewer functions</td>
<td></td>
</tr>
<tr>
<td>apps,</td>
<td>- strict configuration control</td>
<td></td>
</tr>
<tr>
<td>- few configuration restrictions</td>
<td>(provenance ctrl: cannot use others’ unverified code)</td>
<td></td>
</tr>
<tr>
<td>e.g., use others’ code</td>
<td>- higher consumer cost</td>
<td></td>
</tr>
<tr>
<td><strong>Consumers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- low price</td>
<td>e.g., maintenance cost</td>
<td></td>
</tr>
<tr>
<td>- mass access to lots of</td>
<td>- higher consumer cost</td>
<td></td>
</tr>
<tr>
<td>functions and apps</td>
<td>e.g., maintenance cost</td>
<td></td>
</tr>
</tbody>
</table>

**High Assurance?**

- high latency, opportunity cost
- strict configuration control
(provenance ctrl: cannot use others’ unverified code)

**Niche Software Markets**

(at most one “0”s
e.g., aerospace, defense, vehicular, IPC systems)
Challenge 2: Adversary definitions

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

⇒ 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?

- 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?

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?

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

![Diagram showing separation of applications and trust levels]

- **Commodity OS (Giant)**
  - Untrusted
  - Trusted/Verified
  - Hardware Devices

- **µHV**
  - most-privileged

- **App 1, ... , App n**
  - non-privileged

- **Wimp app**
  - isolated execution
  - access policies
  - integrity checking

- Crypto libraries
  - key management
  - 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

- Wimps
- Lack basic services
- Isolated I/O ops.
- Network services
- File system
- Persistent memory
- Isolated I/O ops.

Commodity OS (Giant)

- Untrusted
- Trusted/Verified
- Hardware Devices

\( \mu \text{HV} \)
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

- Wimpy I/O Kernel
- **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**

- unverifiable I/O channel isolation
- cannot be composed with \( \mu \text{HV} \)

**Commodity OS (Giant)**

- core ~ 19,820 SLoC
- drivers ~ 206,376 SLoC

**Linux USB Service**

**Hardware Devices**

**Wimp app**
How to Design a Wimpy I/O Kernel?

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

- **Outsource** USB hierarchy initialization & configuration
- **Verify** USB hierarchy
- **Export** driver & I/O operations to Wimps (ex. 1974 -> )
- **Mediate** USB requests
Code Reduction Results of USB Subsystem

USB 2.0 Code (SLoC)

<table>
<thead>
<tr>
<th>in Wimpy Kernel</th>
<th>Total</th>
<th>in Linux</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Verification</td>
<td>93</td>
<td>USB Core</td>
<td>19,820</td>
</tr>
<tr>
<td>USB Host &amp; Hub Drivers</td>
<td>1844</td>
<td>USB Device Drivers</td>
<td>206,376</td>
</tr>
<tr>
<td>Request Mediation</td>
<td>107</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2,144</td>
<td></td>
<td>226,196</td>
</tr>
</tbody>
</table>

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

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

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

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

3. execute w/ “root” privileges \(\rightarrow\) *(intermediate goal)*
   “buffer overflow” \(\rightarrow\) ...

<Goal    Capabilities  Strategies>

{Attacks} \(\rightarrow\) System Interface

Strategies: Capabilities \(\rightarrow\) Goal
- all exploitable calls?
- all parameter combinations?
- concurrency?
- ...

Power & Privilege

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

Capabilities to *Attack Surface*
- Entry Points: calls, instructions
- Data Input: parameter sets
- Channel: invocation method

Size: single/multiple EPs? Restrictions?
Examples: Cryptographic Modes

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

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

Power & Privilege

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

Capability to
Attack Surface
- Entry Points: Enc/Dec oracles
- Data Input: verifiable/known/chosen…
- Channel: invocation methods

Size: single/multiple EPs? Restrictions
Old Examples: **All** Strategies are Found

**Goal:** Kernel corruption

**Capability:** invoke `copyseg` function

**Small Attack Surface**

**EP:** ustat

**DI:** parameters

**CH:** unprivileged Kernel call

---

**Xenix Kernel**

---

**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

38 system calls
- all parameter combinations for each call

15 independent kernel code paths

Xenix Kernel

[1989]

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

Large Attack Surface

VDG Dec 2014
Giants: *Not* All Strategies are Found

**Goal**\(_0\): access any user’s file

**Capability**: control-flow hijacking to get root access

**Goal**\(_1\): buffer overflow

**Capability**: modify return address

---

**System Interfaces**

**Unix**

**EP**: `rmdir` [1989]

**DI**: `fname, ret. address`

**CH**: unprivileged Kernel call

**EP**: `fingerd` [1988]

**DI**: `... ret. address`

**CH**: unprivileged Kernel call

**EP**: `statd` [1996]

**DI**: `...`

**CH**: unprivileged Kernel call

---

**All ? strategies**

---

**All ? strategies**
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

\[
\begin{align*}
\text{Attack}_1 & < \text{Goal}_1, \text{Capabilities}_1, \text{Strategies}_1 > \\
\text{Attack}_2 & < \text{Goal}_2, \text{Capabilities}_2, \text{Strategies}_2 > \\
\rightarrow & \\
\leftrightarrow & \\
\neq &
\end{align*}
\]

**Implication** (\(A_1 \Rightarrow A_2\)):
System security properties that counter attack \(A_2\) also counter attack \(A_1\)

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

**Strict Dominance:** \((A_1 > A_2) = (A_1 \Rightarrow A_2) \land (A_2 \nRightarrow A_1)\)

**Equivalence:** \((A_1 \Leftrightarrow A_2) = (A_1 \Rightarrow A_2) \land (A_2 \Rightarrow A_1)\)

**Incomparability:** \((A_1 \nLeftrightarrow A_2) = (A_1 \nRightarrow A_2) \land (A_2 \nRightarrow A_1)\)
Traditional Attack Representations

- attack tree

Goal_0

and/or

Goal_1 = Capability_1

Goal_2 = Capability_2

System Interfaces

System

... Entry Point
Data Input
Channel

Size of Attack Surface?

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

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

**Legend: “kill chain” = 1 attack surface (E,I,C) 1-3 + 1 execution strategy (4-6)**
Wimps and Giants in Human Protocols

Certificates
- ID, group, role …
Attributes . . .

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

Either Takes Specified Action or Rejects

Verifies Receiver’s Action; Provides Specified Service or Does Not
Human Protocols: **Wimps and Giants**

Sender's ID, Credentials; No Action => Service Ends
Human Protocols: **Wimps and Giants**

Value Proposition

- *Both parties are honest*  \(\Rightarrow\) *Both are better off* after session

- Future sessions: rational Receiver Takes Action again
Should I accept this *unknown* Sender’s Certificate?
Typical non-answers . . .

Uninformative

Good, Useless Advice

Security Warning

The server you are connecting to is untrusted due to the following problems with the certificates it is using:

- The issuer of the certificate is unknown.

You should examine the certificate and ensure you are willing to trust it for the purpose of identifying the server.

Accept the certificate only if you are sure there is no security risk. If you are unsure, click Reject and contact an Administrator.

View Certificate...

- Accept this certificate temporarily for this session
- Accept this certificate permanently
- Accept this certificate and the Certificate Authority chain permanently

Accept | Reject
Typical non-answers . . .

Speculative, adds a little fear

Good, Useless Advice

Unable to verify the identity of www.********* as a trusted site.

Possible reasons for this error:
- Your browser does not recognise the Certificate Authority that issued the site's certificate.
- The site's certificate is incomplete due to a server misconfiguration.
- You are connected to a site pretending to be www.*********, possibly to obtain your confidential information.

Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to to accept this certificate for the purpose of identifying the Web site www.*********? com?

Examine Certificate...

- Accept this certificate permanently
- Accept this certificate temporarily for this session
- Do not accept this certificate and do not connect to this Web site
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?

Isolation from Sender: verify input, output compliance

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

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

No Behavioral Trust => Isolation

Behavioral Trust
- Beliefs
  - trustworthiness
- Preferences/Aversions
  - Risk
  - Betrayal

=> Trustworthiness Evidence (future)
=> Recovery
=> 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

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