Architectures for Practical Client – Side Security

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

NSF Trustworthy Computing Workshop
October 28, 2010

* based on joint work with
J. McCune, B. Parno, Adrian Perrig, A. Vasudevan and Z. Zhou
How would an NSF Panel react to this title?

…. Architectures for Security …

Report Comments 😊:

…
When was the last time you’ve seen such things in real life?
…
My grandfather did this in kindergarten …
NSF Panel:

Architectures for Practical Security?

Report Comments 😊:

… are you kidding?
NSF Panel:

Architectures for Practical Client –Side Security?

Report Comments (without apologizing to John McEnroe):

… you can’t be serious ! ! ! 😊

… followed by an e-mail from NSF:

We regret to inform you …. 
I regret to inform you…

0. “Axioms” of (in)Security and Human-Usable Security

1. Client-side (ir)Relevance of Past Security Architectures
   - virtualization/OS hypervisors, security kernels

2. Trustworthy Spaces; e.g., OS and Application Partitions
   - simple, understandable security properties
   - stable (as “immutable” as hardware?)

3. Focus: Trustworthy Communication
   - more than just vanilla “trusted path” and secure crypto channels
   - human-oriented
“Axioms” of (in)Security

1. There will always (15+ yrs.) be
   - “bugs” and “operator errors” that will lead to **security vulnerabilities**
   - **adversaries** (e.g., malware, insiders) willing and able to exploit
     security vulnerabilities

2. There will always be **rapid innovation** in information technology.
   => **use components of diverse provenance in the supply chain**
   => non-uniform security assurances (toxic as
   => more attack surfaces

   => **insufficiently stable TCB configurations**
   => perennially out-of-date (high?) assurances

3. There will always be
   - large, complex systems whose **security is not fully understood by**
     **most users**

   “only (nb. software) giants survive [Lampson 99]”
“Axioms” of Usable Security

1. Users Understand only Very Simple Security Mechanisms and Policies
   - the simpler and more uniform the better;
     e.g., coarse-grained; incremental refinement, if any

2. Designers’ Maximum Expectation for Users: Separation of Assets
   - significant assets: financial, health care, defense/intelligence embedded systems
     (e.g., multi-user games, physical systems)
   - everything else: freedom of choice in unrestricted use of Internet
     => accept whatever usable security the market provides for free

   - confine effects of security breaches
   - accountability and recovery (rational expectations?)
Irrelevance of Virtualization to Humans - Past

- VM/370 and apps separation broken in *ad-hoc* ways for communication needs

- VM/370 and apps separation – *not under exclusive user control* (e.g., unauthorized VM switches; Trusted Path for Human Users?)

- *Credible Assurance?*
  
  very large VMM => impractical for (formal) verification

- DEC’s VAX VMM [1990]  
  
  fixed much … but,  
Irrelevance of Virtualization to Humans - Now

+ VM and apps separation broken in more controlled ways
  - e.g., via new device virtualization

- VMs/apps not under exclusive user control

- Credible Assurances?
  very large TCB => still impractical for (formal) verification
  e.g., VMM ~ 160 - 650 K SLoC
  min Root Domain ~10 - 14 M SLoC
  Xen vs HyperV

- Application-level Security?
  Components? Insiders? Internet?
Irrelevance of Security Kernels/TCB

Ex: Scomp ‘85/XTS 400, Gemsos ‘87, TMach ’90, SE Linux ‘01 ...

- isolated (or tamperproof)
- non-circumventable (non-bypassable, or always invoked)
- small enough to be verifiable

- stable TCB Configuration?
- simple policies?
- separation of assets?
- Internet?
Physical Machine Partitioning

Red and Green
(á la Lampson, 2005; CACM 2009)

- off-line Red-Green switch:
  e.g., login - logout
- secure comm. channels:
  e.g., via IPSEC

But,
- ≥ 2 physical machines per user
  e.g., phones
- Green machines: not fully assured

Trustworthy communication?
  - accept input from Red?
  - output to Red?
  (need a “Checkpoint Charlie” – D. Clark)
Logical Machine Partitioning

Red and Green
- by resource partitioning, not virtualization
- very small code base

User-Verified Red-Green switch (via button, lights, buzzer, voice) e.g., “Lockdown” [Vasudevan et al 09]

- Trustworthy Red <-> Green communication?
Application Code Partitioning

Red and Green Separation

User-Verified Red - Green switch

Security-Sensitive Code Blocks
- A, B data secrecy & integrity, execution integrity
- Apps isolated from A, B

e.g., “TrustVisor”: small code base
[McCune et al ’10]

- Trustworthy Red <-> Green <-> Green communication?
User-controlled Code Execution

Red and Green Separation

User-Verified Red - Green switch

Security-Sensitive Code Blocks

Trusted Path for Selected Devs.
- still a small code base

- Trustworthy Red <-> Green <-> Green communication?
Focus: Trustworthy Communication

Trustworthy Communication
- past: secure crypto channels, trusted path
- now: trustworthy content?
  e.g., accept foreign input.
  control output dissemination?
  communication compliance?

Diagram:
- Machine & Code R-G Partitions
- Hyper ++ Hypervisor
- HW++
- external verifier (V)

Trustworthy Communications
Trustworthy Input?

Receiver Makes it Right – “all trust is local” [Lampson, CACM 09]

- verification of input acceptability (and sanitization) - application based; e.g.,
  - ascii/unicode? … pdf? … doc, ppt, xls? … Java and other scripts?

However, … can Input be always verified or sanitized?
No!

- Input = arbitrary code
  verification of code’s “output behavior” by Receiver (undecidable)

When Input can be verified, is verification always efficient?
No, not likely!

- Input = solutions to some co-NP complete problems (i.e.,
  efficient solutions at Sender & inefficient verification at Receiver)
Trustworthy Input?

When Input verification is efficient, is it \textit{always} practical?
No!

- Input = results/output of a computation outsourced to Sender
efficient result verification by Receiver [Parno 2010]
  => \textit{fully homomorphic encryption}
    [Gennaro, Gentry, Parno 2010]

When Input verification is efficient and practical,
is it \textit{always} scalable (e.g., in the Internet)?
No!

- Input = multi-level integrity, integrity-labeled object [Biba 77]
  => \textit{integrity-labeled closed input}

- Input = output of a \textit{trusted transaction} [Clark-Wilson 87]
  => \textit{application-closed input}
Handling Untrusted Input ...

Do not always click “Accept”!

Instead:

Input cannot be verified/sanitized

=>

optimistic trust (i.e., “Accept”)

=> accountability & recovery
An Optimistic-Trust Protocol [D. Ariely 08]

An Optimistic Trust

$10$

Dealer

$10$

or Accept Deal:
send $10$ to Player 2
$+$ $30$

receive $25$ from Player 2

$0$

home

$10$

home

$50$

$25$

$25$

$25$

cheats: keeps $50$

send $50/2$ to Player 1
(Rational) Accountability & Recovery

Ernst Fehr’s “Revenge Protocol” and PET-scan experiment

Revenge: most Players 1 paid the Dealer (at extra loss) to punish Player 2 with pleasure: Players 1’s brain striatum showed pleasure and reward

Accountability ($50 – $25) and Recovery (+ $25)
Handling Untrusted Input ...

Input cannot be verified/sanitized =>

**optimistic trust (i.e., “Accept”)**

=> accountability & recovery

Ex. 1: accountability of **certificates signers? self-signed** certificates?

- signers are *always accountable*; e.g., notarized certificates [GLP 92]
- signers are *sometimes accountable*;
  - vouchers; e.g., “friend of a friend” in PGP; Facebook?
  - network evidence of safe use; e.g., “Perspectives,” [EGB02]

Ex. 2: accountability with “second opinion;” multiple providers of **same input**

- conflicting signed evidence [RPGH08]

Human-Oriented **Accountability** Policies?
Input Compliance?

Input Compliance with Receiver Policies

=> global policies

- flooding control [G03], spam, super-spreading worms [PPZ09]

Can Input Compliance be *always* verified by Receiver?

No!

- Input-Provider can be arbitrary code
  code’s “output behavior” verification by Receiver (undecidability?)

A possible solution:
- user agreements: attestation of Input-Provider code
  e.g., install hypervisor + “virtual network card,”
  trusted third party to set/verify compliance
Trustworthy Output? Compliance?

Sender Makes it Right: “all trust is local”
- permissible output? (application semantics); e.g.,
  - guards, filters, censors
  - “object-reuse” control: memory scrubbers, clean up
- output permissibility cannot be checked => accountability of output receivers
  e.g., self-signed certificates?
- signers are always accountable; e.g., notary certificates [GLP 92];
  conflicting evidence [RPGH08]
- signers are sometimes accountable;
  - vouchers; e.g., “friend of a friend” in Facebook;
  - network evidence of safe use; e.g., “Perspectives,” [EGB02]

Receiver Compliance w/ Sender Policies => global policies
- originator control, digital rights management, traitor tracing
- memory-less subsystems, confinement
- do you know where your data is?
Conclusions

1. Need to Rethink Security Architectures for Clients
   - impact of truly usable security
   - impact of feasible recourse after security breaches

2. Trustworthy Spaces: Isolation, Encapsulation/Sandboxing
   - reflect asset separation goals of humans -- not just system designers’
   - area needs more engineering (e.g., stable partitions, assurances)
   - human-oriented policies, tools, aids

3. Focus: Trustworthy Communications for Humans
   - trusted path beyond login (for humans)
   - verification of input/output legitimacy
   - establishment of accountability & recovery
   - sender/receiver compliance
Virtualization for Security – Brief History

VMM-based security:
  - ’67 – 73 IBM develops CP-67 based and VM/370 operating systems
  - 1973 - S. Madnick and J. Donovan suggest a VMM-based security
  ~ 1973 - IBM and SDC begin security study of IBM’s VM/370 systems
  ~ 1976 - 1980 SDC produces KVM/370
  ~ 1986 - 1990 Digital Equipment Corp produces VAX VMM
  ~ 1990 - 2000 “Dark Ages” of VM-based security

More recently:
  - novel uses for VM architectures; e.g., server consolidation, load balancing, clouds
  ~ 2000 -> now; VM-based security has been revived

“Virtual Machines*
are the Way of the Future and Always Will Be”

* paraphrasing Jim H. Morris’ statement re: “capability-based systems”