MiniBox: A Two-Way Sandbox for x86 Native Code

Yanlin Li, Jonathan McCune, Jim Newsome,
Adrian Perrig, Brandon Baker, and Will Drewry

Carnegie Mellon University
Google, Inc.

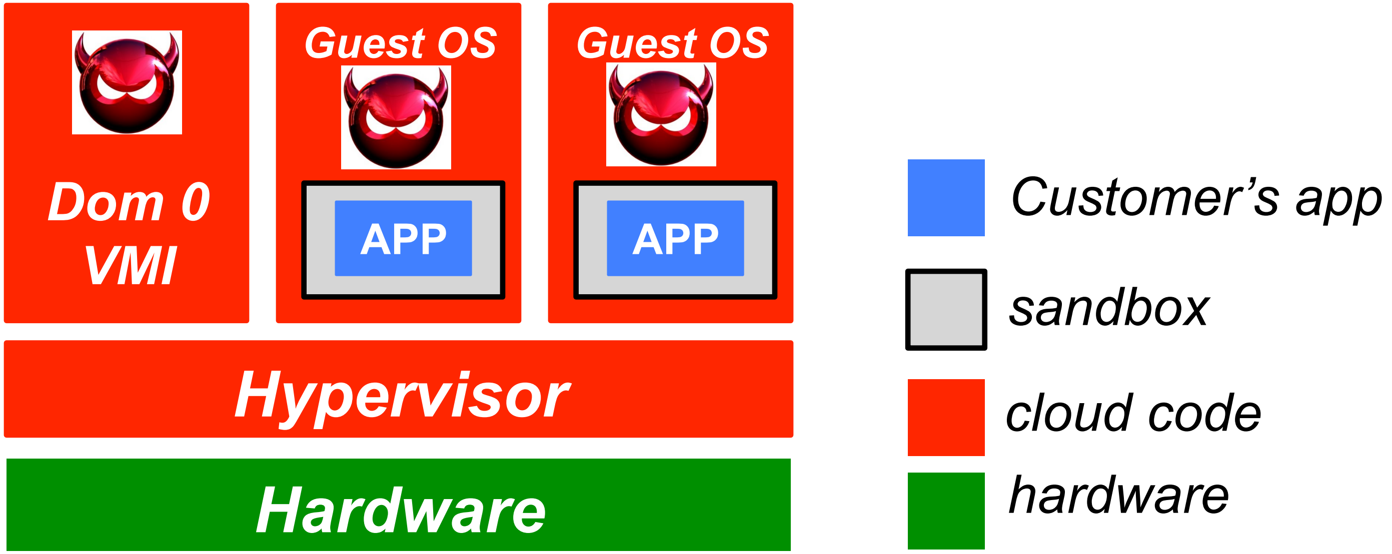
Platform as a Service

- One of the most commercialized forms of cloud computing
  - One million active applications were running on Google App Engine in 2012[1]
- It is critical to protect the OS from the large number of applications in PaaS
  - Sandbox is deployed to protect the guest OS

[1] http://gigaom.com/2012/06/28/google-app-engine-by-the-numbers/

Current Sandbox

- Only one-way protection
  - Protect OS from malicious Apps
- App is exposed to malicious code in guest OS
  - **Not desired by customers**



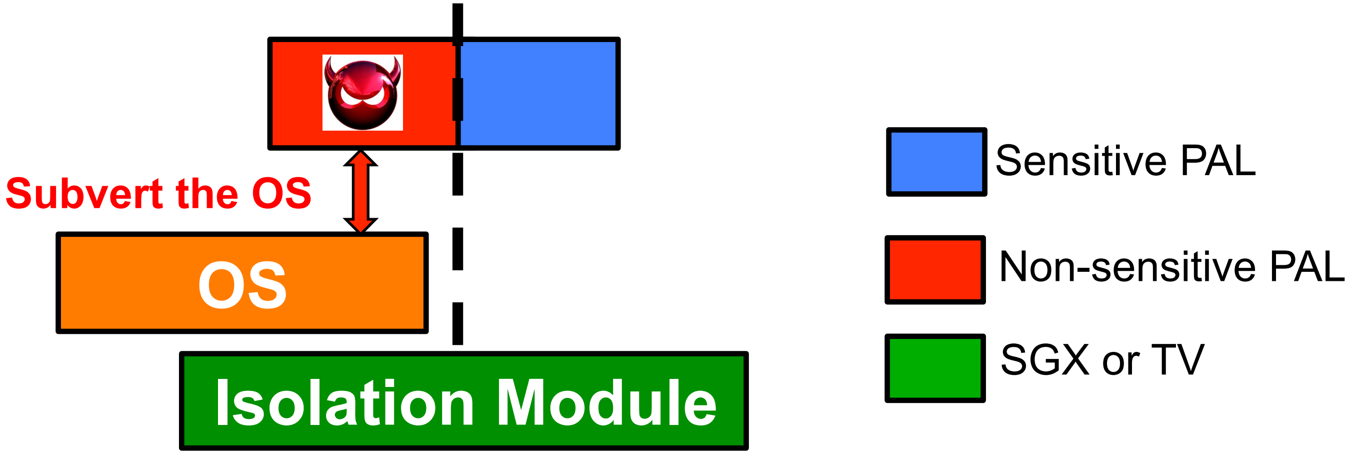
Hypervisor-based architecture

Goal: Two-Way Sandbox

- Two-way protection for x86 native code
  - OS Protection: protect a benign OS from a misbehaving application
  - Application Protection: protect an application from a malicious OS

Wait.. It has been solved!?

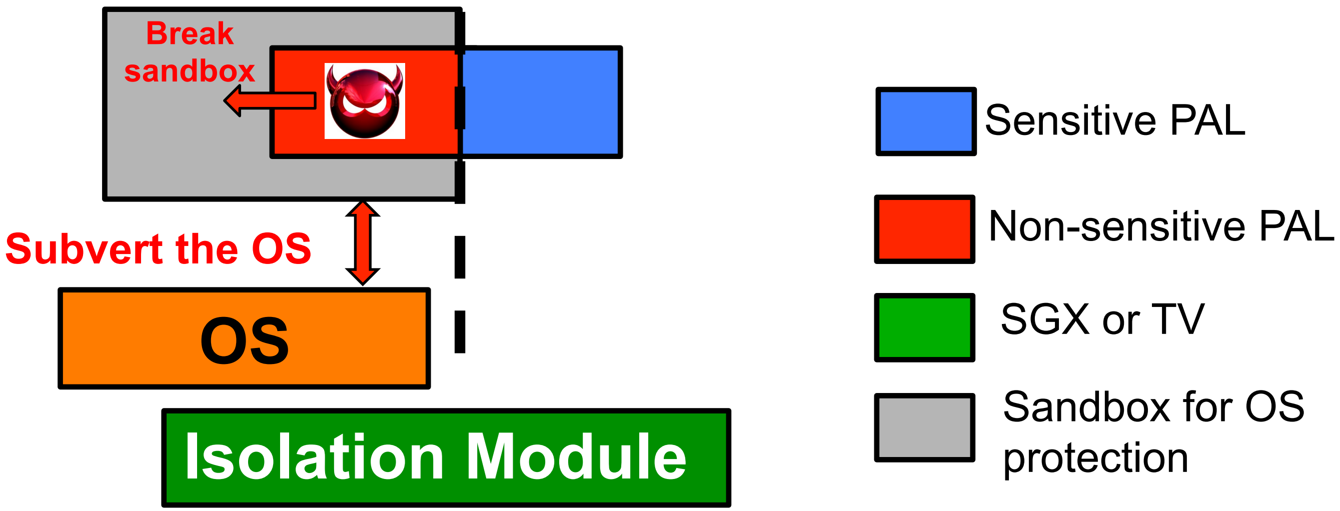
- Intel Software Guard Extensions (SGX) [1]
  - Hardware-based two-way memory isolation
- TrustVisor (TV) [2]
  - Hypervisor based two-way memory isolation
- Only isolate a Piece of Application Logic (PAL) from the OS



[1] Innovative technology for CPU based attestation and sealing. *HASSP* (2013)
[2] TrustVisor: Efficient TCB reduction and attestation. IEEE *S&P* (2010)

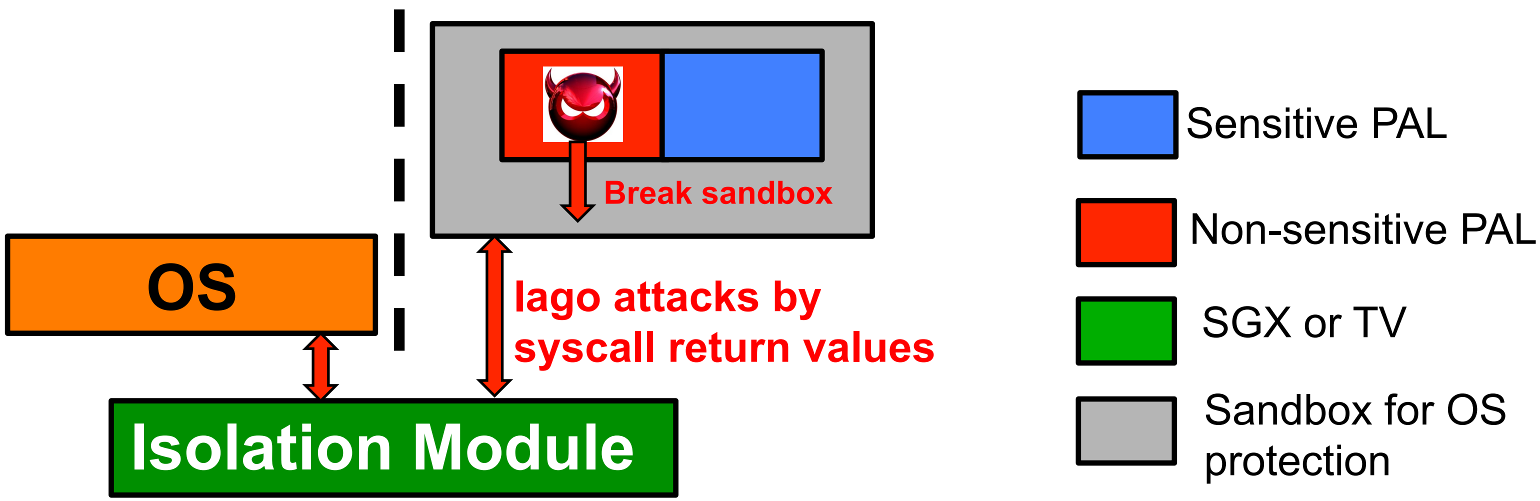
Combine Sandbox and Isolation

- Sandbox to confine the non-isolated PAL
  - Sandbox exposes large interface to the application
  - Developers need split the application
    - require substantial porting effort



Combine Sandbox and Isolation

- Deploy sandbox in an isolated environment
  - Avoid porting effort
  - Sandbox exposes large interface to the application
- Iago attacks [1]:
  - A malicious OS subverts a protected process by returning a carefully chosen sequence of return values to sensitive system calls



[1] Iago attacks: Why the system call API is a bad untrusted RPC interface. ASPLOS 2013

Challenges

- It is promising to combine a one-way sandbox and a two-way memory isolation mechanism to establish two-way protection
- Challenges
  1. System design of combining a one-way sandbox and a memory isolation mechanism to establish two-way protection
  2. Minimize and secure the interface between software modules for OS protection and the application
  3. Protect the application against Iago attacks

Contributions

1. Design, implement, and evaluate MiniBox, the first attempt toward a practical two-way sandbox for x86 native applications.
2. Demonstrate it is possible to provide a minimized and secure communication interface between software modules for OS protection and the application to protect against each other.
3. Demonstrate it is possible to protect against Iago attacks, and provide an efficient execution environment for the application.

Outline

- Motivations
- Goal and Challenges
- Assumptions & Adversary Model
- MiniBox Design
- Implementation & Evaluation
- Related Work
- Conclusion

Assumptions

- For both protections
  - No physical attacks (e.g., CPU is trusted)
  - Cryptographic primitives are secure
- For application protection
  - Applications do not have memory safety bugs (e.g., buffer overflows) or insecure design
- For OS protection
  - The small system call interface that OS exposes to the application on MiniBox is free of vulnerabilities
  - OS does not have concurrency vulnerabilities in system call wrappers[1]

[1] WATSON, R. N. M. Exploiting concurrency vulnerabilities in system call wrappers. In *Proceedings of USENIX Workshop on Offensive Technologies* (2007).

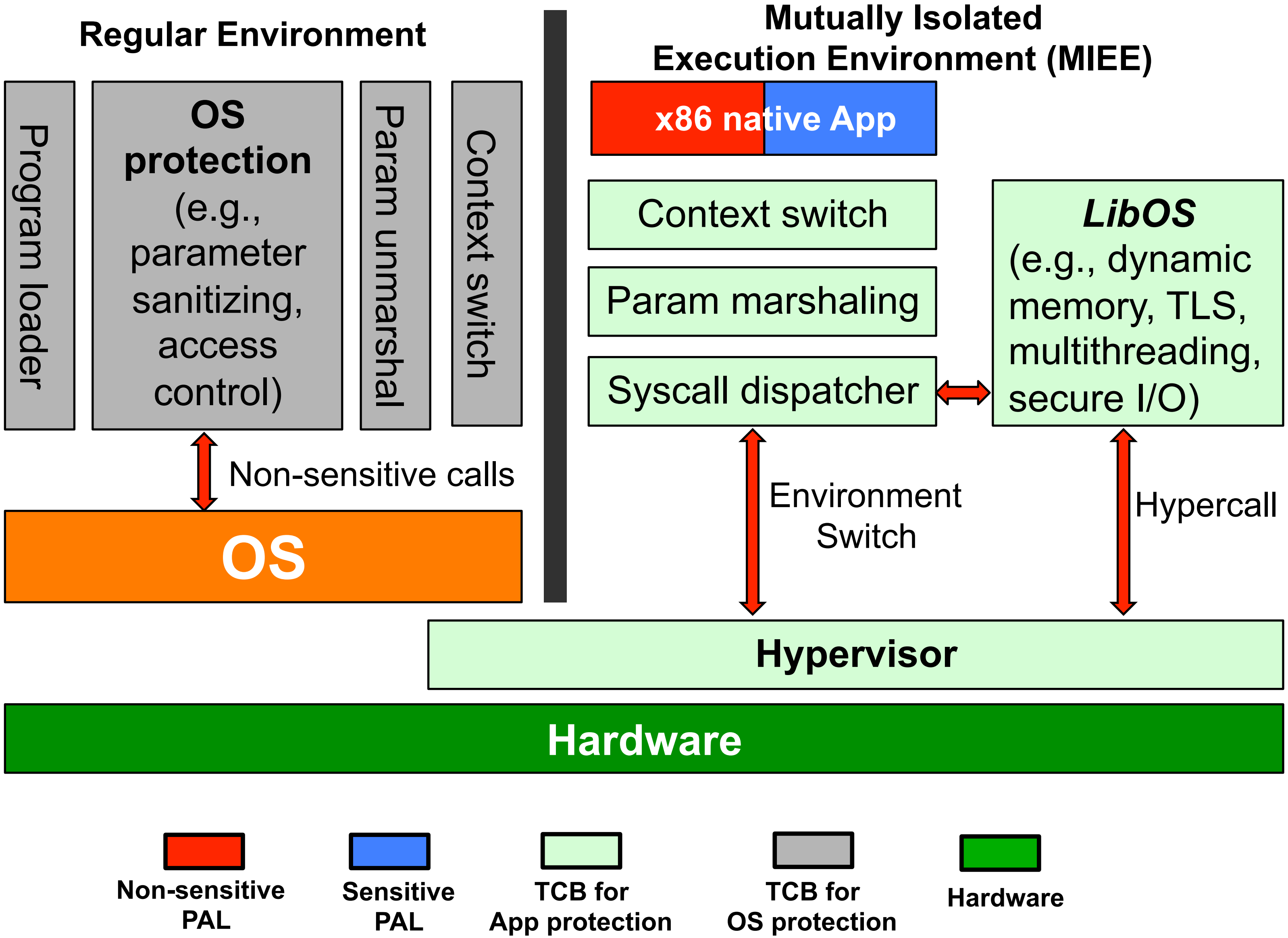
Adversary Model

- Adversary model for App protection
  - OS is controlled by adversaries
  - Attempt to access the app’s memory
  - Attempt to perform Iago attacks
- Adversary model for OS protection
  - App is malicious and contains privileged instructions
  - Attempt to subvert and control the OS
- Do not prevent
  - DoS attacks or side channel attacks

MiniBox Overview

1. Combine one-way sandbox for x86 native code and hypervisor-based two-way memory isolation
2. Split sandbox components into service runtime modules and OS protection modules
   - Include the service runtime in the isolated memory space with the App to support App execution
3. Expose a subset of system call interface to the App, and Split system calls into sensitive calls and non-sensitive calls
   - Handle sensitive calls in the isolated environment
4. Minimize and secure the communication interface between OS protection modules and the application

MiniBox Architecture



Minimized and Secure Communication Interface

- Minimized communication interface between two environments
  - In load time: program loader
  - In run time: only system call interface
- Secure communication between two environments
  - Application specifies system call information
  - Hypervisor passes system call parameters and return values between two environments
  - OS protection modules check the system call parameters

Exceptions/Interrupts and Debugging

- Exceptions and interrupts
  - Hypervisor handles exceptions and non-maskable interrupts
  - Maskable interrupts are disabled
- MiniBox Debugging mode
  - The hypervisor-based memory isolation is disabled
  - One app-layer module copies system call parameters between two environments
  - Developers can use *GDB* for applicaiton debugging

Implementation

- MiniBox prototype
  - Public implementation of TrustVisor (Version 0.2.1) [1]
  - Native Client open source project [2]
  - Support for multi-core and both Intel and AMD processors
  - Ubuntu 10.04 as the guest OS

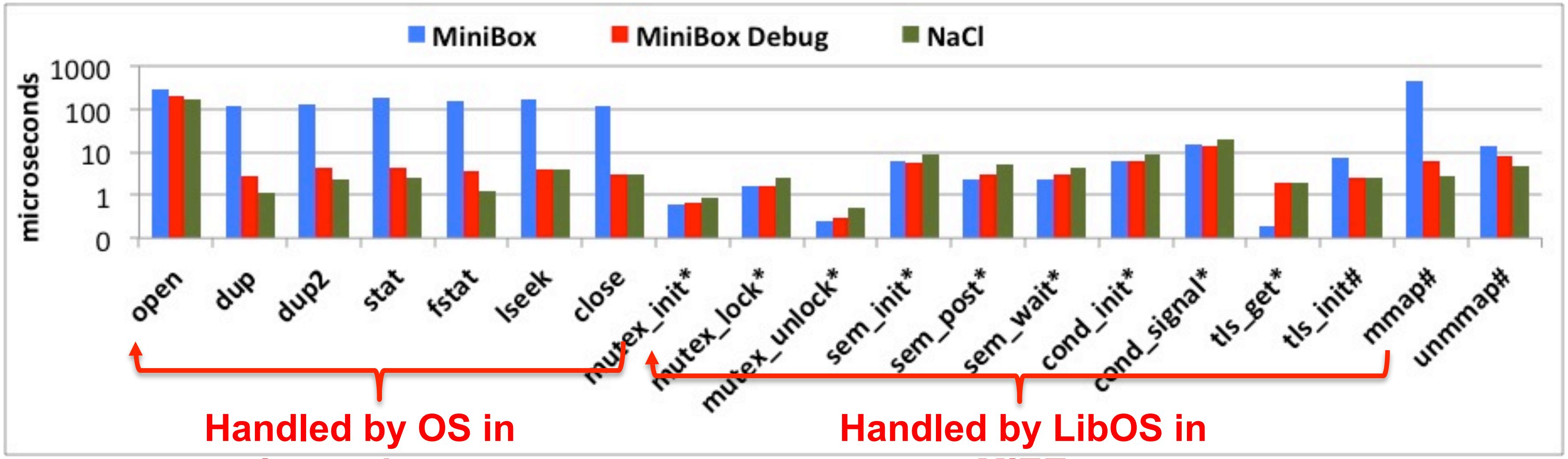
Modules	SLoC
Hypervisor	14414 (TrustVisor), add 691
NaCl ELF file Loader	add 299
Service runtime in MIEE (including the LibOS)	3550

[1] Design, implementation and verification of an extensible and modular hypervisor framework. *IEEE S&P* (2013)
[2] Native Client: A sandbox for portable, un- trusted x86 native code. *IEEE S&P* (2009)

Evaluation

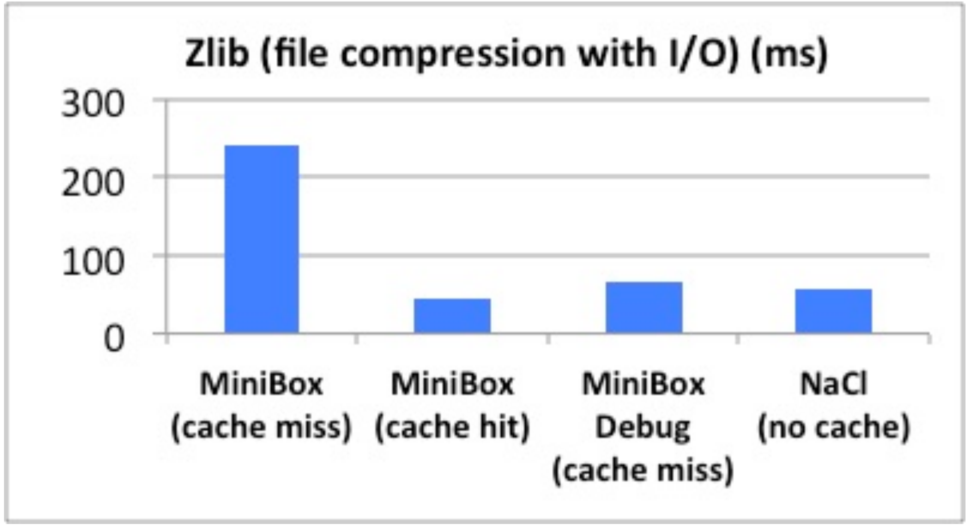
- Microbechmarks
  - System call overhead
- Application benchmarks
  - I/O-bound applications
  - CPU-bound applications

System Call Overhead



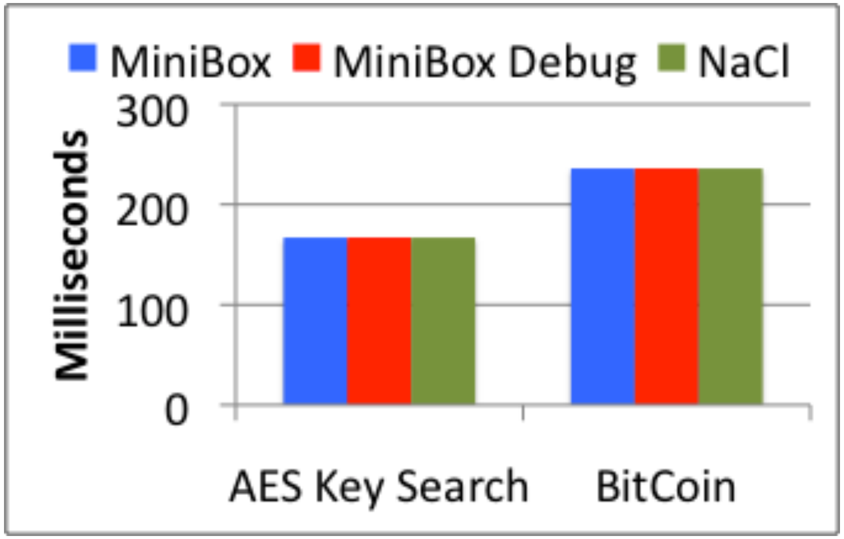
- System calls handled by the OS have high overhead on MiniBox
  - Each call causes environment switches
  - ***Hypervisor-based Environment switches on MiniBox cause high overhead for non-sensitive system calls***
- System calls handled inside the Mutually Isolated Execution Environment have similar performance to those on vanilla NaCl

I/O-Bound Application (Zlib)



- Zlib application
  - Read 1 MB of file data from file system
  - Compress the read data
- File I/O is expensive on MiniBox
- *We expect that **cache buffer** will improve the application performance in practice*

CPU-Bound Applications



- AES key search
  - Encrypt 128-Byte plain text for 200, 000 times
- BitCoin
  - Perform 200, 000 SHA-256 computation
- *MiniBox does not add any noticeable overhead to CPU-bound applications over NaCl*

Related Work

- Protecting applications
  - HOFMANN, O., DUNN, A., KIM, S., LEE, M., AND WITCHEL, E. InkTag: Secure applications on an untrusted operating system. ASPLOS, 2013.
  - BAUMANN, A., PEINADO, M., HUNT, G., ZMUDZINSKI, K., ROZAS, C. V., AND HOEKSTRA, M. Secure execution of un-modified applications on an untrusted host. http://research. microsoft.com/apps/pubs/default.aspx?id=204758, 2013.
  - TA-MIN, R., LITTY, L., AND LIE, D. Splitting interfaces: Making trust between applications and operating systems configurable. SOSP, 2006.
  - MCCUNE, J. M., LI, Y., QU, N., ZHOU, Z., DATTA, A., GLIGOR, V., AND PERRIG, A. TrustVisor: Efficient TCB reduction and attestation. *IEEE S&P,* 2010.
  - SINGARAVELU, L., PU, C., HA ̇RTIG, H., AND HELMUTH, C. Reducing TCB complexity for security-sensitive applications. *EuroSys,* 2006.
- Sandbox for OS protection
  - PORTER, D. E., BOYD-WICKIZER, S., HOWELL, J., OLINSKY, R., AND HUNT, G. C. Rethinking the library OS from the top down. *SIGPLAN,* 2011.
  - YEE, B., SEHR, D., DARDYK, G., CHEN, J. B., MUTH, R., ORMANDY T., OKASAKA, S., NARULA, N., FULLAGAR, N., AND GOOGLE INC. Native Client: A sandbox for portable, un- trusted x86 native code. *IEEE S&P,* 2009.
  - JANA, S., PORTER, D. E., AND SHMATIKOV, V. TxBox: Building secure, efficient sandboxes with system transactions. In *IEEE S&P,* 2011.
  - KIM, T., AND ZELDOVICH, N. Practical and effective sand- boxing for non-root users. In *Proceedings of USENIX ATC,* 2013.

Conclusion

- We made the first attempt toward a practical two-way sandbox for x86 native code.
- We proposed a generic architecture for establishing two-way protection for x86 native code on commodity computer systems.
- We anticipate that MiniBox will be widely adopted on systems where two-way protection is desired (e.g., the PaaS cloud computing platforms).

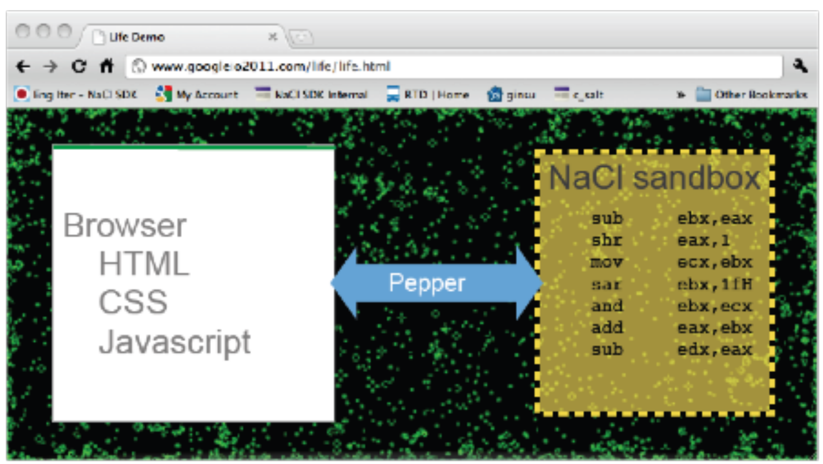
Thank you for your attention!

Q & A

Email: yanlli@cmu.edu

Native Client[1]

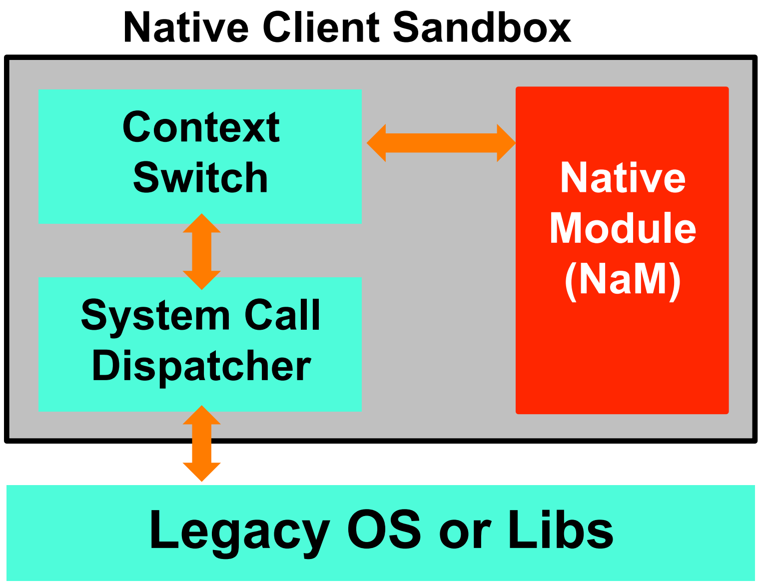
- NaCl: a sandbox technology for running Native Module (NaM) on the Web
  - Software Fault Isolation (SFI)
    - NaM runs in its own segmentations
  - Disassembler & Validator
    - Guarantee that there are no privileged instructions that can break the SFI in the NaM



[1] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Orm, S. Okasaka, N. Narula, N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. Oakland, *2009*

Native Client[1]

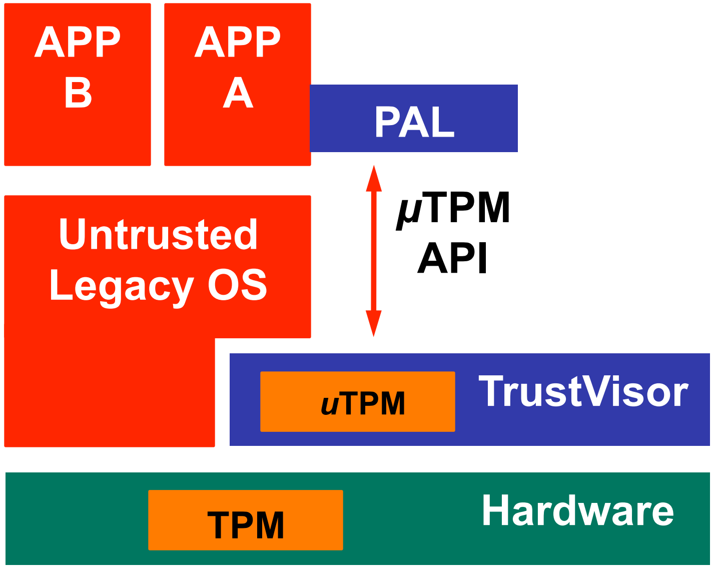
- Service Runtime in NaCl
  - System call interfaces for NaM
- Special toolchain to build NaM
  - Support service call APIs



[1] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Orm, S. Okasaka, N. Narula, N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. Oakland, *2009*

TrustVisor[1]

- A small hypervisor that:
  - Isolates a Piece of Application Logic (**PAL**) from the legacy OS by nested pages
  - Provides *μ*TPM APIs to the PAL
  - Measures integrity of PAL for attestation
- Integrity Measurement
  - Hardware TPM → TrustVisor
  - TrustVisor → PAL
- Shortcomings
  - No system call from PAL
  - Porting Effort



[1] J M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig. TrustVisor: Efficient TCB reduction and attestation. Oakland, *2010*

Exceptions/Interrupts and Debugging

- Exceptions and interrupts
  - Hypervisor handles exceptions and non-maskable interrupts
  - Maskable interrupts are disabled
- Debugging mode
  - The hypervisor-based memory isolation is disabled
  - One app-layer module copies system call parameters between two environments

Against Iago Attacks

- Handle sensitive calls in LibOS inside the isolated execution environment
- LibOS supports
  - Dynamic memory management
  - Thread local storage management
  - Multi-thread management
  - Secure file I/O

MiniBox Architecture