

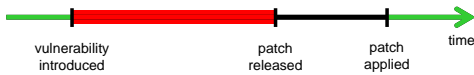
Detecting Past and Present Intrusions through Vulnerability-Specific Predicates

A. Joshi, S. King, G. Dunlap, P. Chen
SOSP 05

Motivation

- How do I know if my systems were affected by a 0-day exploit before a patch was released?
- I need time to test the patch, but I don't want to leave my systems vulnerable!
- Rebooting/restarting the application is not an option right now, but I don't want to leave my systems vulnerable!

Past...



...and Present Intrusions.



Example

1. `char *str = some_string;`
2. `int len = strlen(str);`
3. `char buf[BUFSIZE];`
4. `strcpy(buf, str);`

Vulnerability-specific predicates

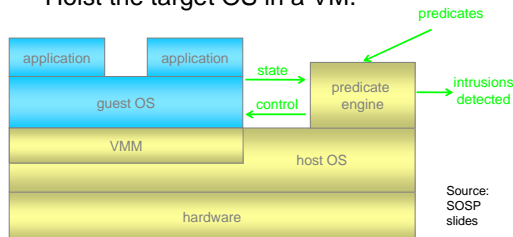
- Targets a specific vulnerability (hence the name) - not one exploit.
- Is written by the person who writes the patch.

Goals

- Predicates don't perturb the target.
- Works for both OS and applications.
- No need to restart application/reboot OS.
- Predicates are easy to write.
- Low overhead.

Solution: Virtual Machine Introspection

- Hoist the target OS in a VM.



How it works

- Insert traps for specific events.
- When trap is triggered, run predicate.
 - Predicates can use symbolic program names (via debug information).
 - Predicates can examine guest state.
 - Predicates can call guest functions (need to understand the calling convention)
- Checkpoint and rollback.

Dealing with applications

- Applications are not always in memory and their code is demand-paged.
- Code can get evicted from the buffer cache.
 - **Add predicates to *fork*, *exec*, *mmap* and the kernel scheduling function.**
- Virtual pages may be swapped out.
 - **Inject code to get the kernel to swap required pages in.**

Detecting Past Intrusions

- Log all outside input to be able to replay the past.
 - 1 GB/day.
- Replay the past, check if any predicate is triggered.
 - Replay 1 to 2 orders of magnitude faster.

Evaluation

- Install a predicate in 0.5 ms.
- Predicates written for 20 bugs:
 - Predicates are short, albeit slightly more complex than patches.
 - Predicates are easy to write.
- Overhead depends on how frequently the code path is executed.

Goals

- ✓ Predicates don't perturb the target.
- ✓ Works for both OS and applications.
- ✓ No need to restart application/reboot OS.
- ✓ Predicates are easy to write.
- ✓ Low overhead.

Pros

- Clear explanation of problem + solution.
- Solves real problems.
- Modular design.
- Solution feels practical (at least in some settings).
- Clever way to detect past intrusions.

Cons

- Changes Timing
 - May expose race conditions present in the target software
- Software breakpoint
 - On x86 platform, s/w breakpoint is implemented by temporarily overwriting instructions which could change in instruction of the code if it reads its own code segment
- Only works on native code
 - Predicates gain control as interpreted code about to run and evaluate the input.

Cons(2)

- Only works on Uniprocessors
 - Relies on ReVirt in the past time interval for system replay
 - Predicate refresh assumes that processes are interleaved by schedule function
- Predicates must be written by hand
- Heavily rely on the existence of a “patch” for a vulnerability

Cons (3)

- Scalability
 - How many predicates can run in parallel?
 - Log volume.
- Solves the problem:
 - A patch for a vulnerability is available and user do not want to apply it for some reason
 - When a patch is available, with the help of a replay system, try to see if vulnerability is already exploited by attackers

Terra: A Virtual Machine-Based Platform for Trusted Computing

T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh
SOSP 03

Motivation

- Commodity OS are too complex hence offer low assurance
- Commodity OS poorly isolate applications from one another
 - Compromise of any apps often compromises the entire platform
- Weak mechanisms for peers authentication
 - Make building secure distributed apps difficult
- No trusted path between users and apps

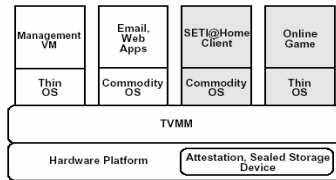
Introduction

- Trusted Virtual Machine Monitor (TVMM)
 - Provides "open" or "closed" box VM
 - All VMs run independently and concurrently on the same platform
- Allow applications to cryptographically authenticate to remote parties through *Attestation*

Virtual Machine

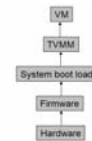
- Provides each VM a raw hardware interface
 - Application(s) + OS to run in each VM
 - Customizability
- Closed-box VM is isolated from the rest of the platform by encapsulating its content (cannot be inspected/changed by platform owner)

Architecture



VM Identity

- Attestation enables application inside a VM to authenticate itself to remote parties
- Certificate Chain



Attestation – Certificate Chain

- A component wanting to be certified:
 - Generates public/private key
 - Passes public key to lower-level
 - Lower-level generates and signs the certificate containing:
 - SHA-1 hash of attestable parts of higher component
 - Higher component's public key and application data

Implementing Attestation

- Granularity of Attestation
 - Single hash for entire entity not desired when subsections of a hashed entity to be verified
 - Divides attestable entities into fixed size blocks, and each block is hashed separately
 - VM descriptor contains a hash over these hashes
 - Hashing 4GB entity into 20-byte SHA-1 hashes with 4kB block sizes yields 20MB of hashes

Implementation Attestation (2)

- Modes of Attestation
 - Ahead-of-Time Attestation
 - Attest entire entity at boot time
 - Single bit corruption prevents booting for VM
 - Optimistic Attestation
 - Does not attest at startup, only lazily checked as blocks are read from disk at runtime
 - Halts VM immediately when block fails to verify at the time when it's read from disk

Platform Security

- Specified by TVMM and Management VM
- TVMM
 - Policies that is required for Attestation
 - E.g. VM isolation, protect closed-box VM contents
- Management VM
 - Platform Access Control + Resource Management
 - Access to peripherals, divides storage, CPU, and memory between VMs
 - Starts, stops, and suspends VMs

Trusted Quake

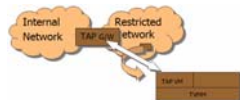
- Runs in minimal Linux kernel
- Closed-box VM boots Quake directly
- Preload library interposes Quake's network communication to perform attestation and key exchange
- Attests to each other hosts (clients and servers) running same version

Trusted Quake (2)

- Performance
 - Boot time:
 - No attestation: 26.6 seconds
 - Ahead-of-Time: 57.1 seconds
 - Optimistic: 27.3 seconds
 - Optimistic + encryption: 29.1 seconds
 - Interactive time:
 - No subjective differences (Untrusted vs. trusted Quake)

Trusted Access Points (TAP)

- A filter for network traffic that runs on each clients that wishes to connect to the network
- VM contains VPN client and firewall software
- Attests itself to the TAP gateway



Conclusion

- A VMM that
 - Supports “open-box” and “closed-box” VM
 - Require Tamper Resistant Hardware
 - Supports Attestation for Application Identification
 - Optimistic attestation

Pros

- A Simple and Sound Idea
- Flexible
 - Compatibility: Different VMs run different OSes
 - Extensibility: Increases number of VMs
 - Security: OSes can be tailored to reduce chances to vulnerability
- Performance seems not affected much by attestation (optimistic)

Pros(2)

- Presentation is clear
- Sounds like it works and the solution can be apply generally to existing systems
- Provides complete single VM implementations running on Terra
 - Trusted Quake and TAPs

Cons

- Paper lacks focus:
 - Discussion of Privacy, Interoperability... could have been tightened.
 - Discussion of several aspects of secure platforms where the authors have nothing new to propose: Secure User Interface, Device Driver Security, Hardware support.
- Contributions are unclear.
- Very superficial evaluation.

Cons (2)

- Very coarse-grained approach:
 - Attest to whole disk partitions.
 - The whole appliance has to come from the same vendor.
- Splitting a closed box between “Attested” and “Unattested” partitions may be hard.
- Licensing issues: What if my box requires Windows?

Labels and Event Processes in the Asbestos Operating System

P. Efsthopoulos, M. Krohn, S. VanDeBogart,
C. Frey, D. Ziegler, E. Kohler, D. Mazieres,
F. Kaashoek, R. Morris
SOSP 05

Goal

“Asbestos should support efficient, unprivileged, and large-scale server applications whose application-defined users are isolated from one another by the operating system, according to the application policy”

Goal

“Asbestos should support efficient, unprivileged, and large-scale **server applications** whose application-defined users are isolated from one another by the operating system, according to the application policy”

Goal

“Asbestos should support **efficient**, unprivileged, and **large-scale** server applications whose application-defined users are isolated from one another by the operating system, according to the application policy”

Goal

“Asbestos should support efficient, **unprivileged**, and large-scale server applications whose application-defined users are isolated from one another by the operating system, according to the application policy”

Goal

“Asbestos should support efficient, unprivileged, and large-scale server applications whose **application-defined** users are isolated from one another by the operating system, **according to the application policy**”

Goal

“Asbestos should support efficient, unprivileged, and large-scale server applications whose application-defined users are isolated from one another **by the operating system**, according to the application policy”

Why a new OS?

- Current OSs isolate applications poorly.
- Traditional Mandatory Access Control (MAC) is inadequate.
- The proposed changes impact crucial aspects of the system: IPC, permissions, programming model.

New OS Features

- Labels: Track information flows and prevent leakages.
- Event processes: Lightweight processes that optimized for event-driven servers.

Asbestos Labels

- Label: Function from handles to levels.
- Handles: correspond to a compartment.
- Levels: [\star , 0 , 1 , 2 , 3] (ordered set).
- Two labels per process P : P_S and P_R .
- Example:
 - $P_S = \{u1_T 3, 1\}$
 - $P_R = \{u1_T 3, 2\}$

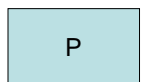
Label lattice

- $L_1 = \{h1 3, h2 1, 2\}$
- $L_2 = \{h1 2, h2 2, 2\}$
- $L_1 \sqcap L_2 = \{h1 2, h2 1, 2\}$ (min)
- $L_1 \sqcup L_2 = \{h1 3, h2 2, 2\}$ (max)
- $L_1 \subseteq L_2$ iff $L_1(h) \leq L_2(h)$ for all h .

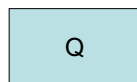
IPC

- P can only send to Q if Q can and is willing to accept P 's current contamination level (P_S) as indicated by Q_R .
- If Q accepts the message, Q 's contamination level (Q_S) changes.

Example

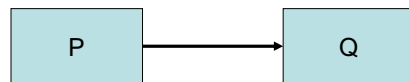


$P_S = \{u1_T 3, 1\}$
 $P_R = \{u1_T 3, 2\}$



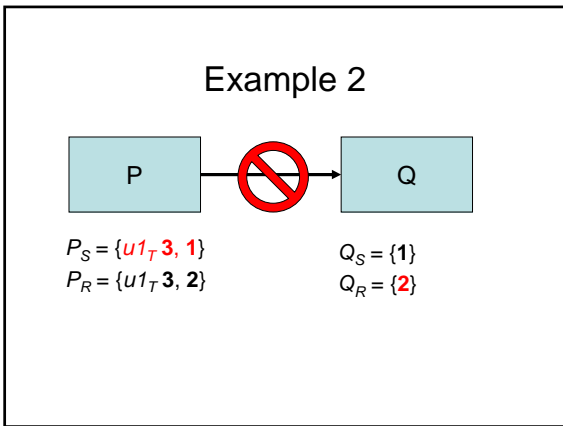
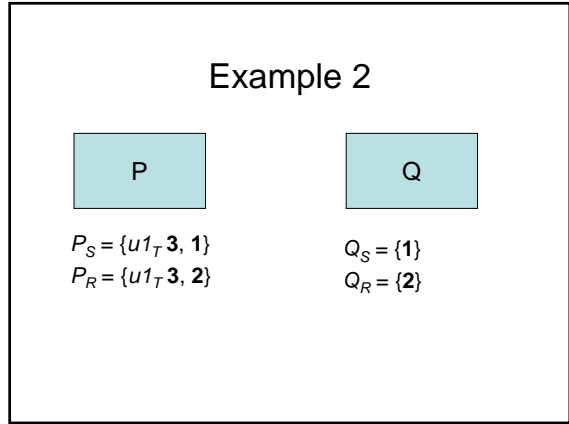
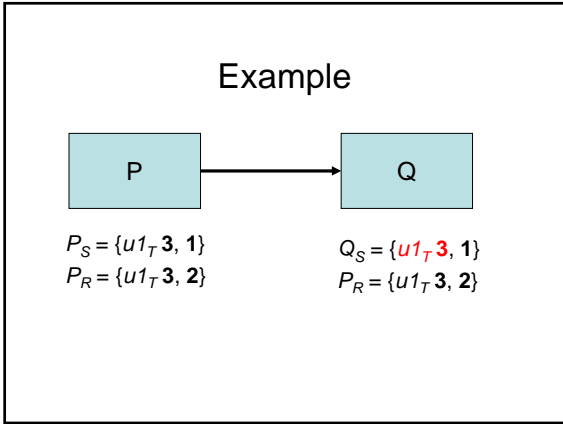
$Q_S = \{1\}$
 $Q_R = \{u1_T 3, 2\}$

Example



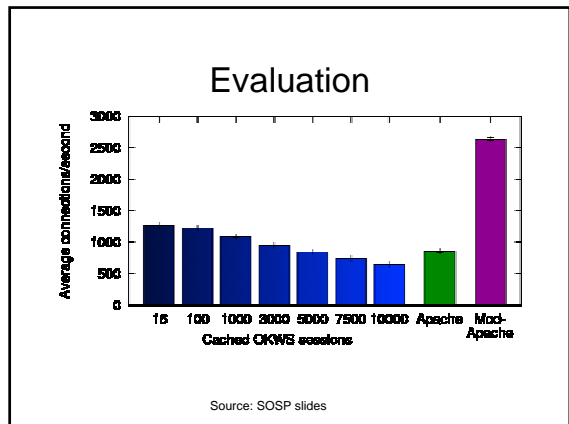
$P_S = \{u1_T 3, 1\}$
 $P_R = \{u1_T 3, 2\}$

$Q_S = \{1\}$
 $Q_R = \{u1_T 3, 2\}$



- ### Refinements
- Discretionary contamination: Sender can contaminate the Receiver's send label using discretionary *contamination label*.
 - Decontamination: Use special \star privilege and discretionary *decontamination labels*.
 - Integrity: Use level 0 and a special, discretionary *verification label*.
 - Contamination prevention: use *port label*.

- ### Event Processes
- Forked from a base process.
 - Leverage event-driven dispatch loop:
 1. **ep_checkpoint**(&msg)
 2. if (!state.initialized){
 3. initialize_state(state);
 4. state.reply = new_port();
 5. }
 6. process_msg(msg, state);
 7. **ep_yield**();



Conclusions

Asbestos:

- Lets applications use Mandatory Access Control for information flows.
- Can do so efficiently thanks to event processes.

Pros

- Very relevant and timely problem.
- Work triggered by shortcomings of existing OS (OKWS).
- Novel approach with a new angle.
- Empower the application.
- Proof-of-Concept implementation.

Cons

- Label cost seems to be high as number of sessions increases
- Lack of Implementation Details
 - No details on the actual OS implementation
- Labeling is not a new idea
- No similar work comparison on event processes
- More applications
- Difficult for existing applications to adapt

Credits

- IntroVirt SOSPP slides.
- Asbestos SOSPP slides.