ReVirt : Enabling Intrusion Analysis through Virtual Machine Logging and Replay

Or

“We Can Remember It for You Wholesale”
(with apologies to Philip K. Dick)

George Dunlap, Sam King, Sukru Cinar, Murtaza Basrai and Peter Chen
University of Michigan

The Problem

- Security of computer systems today needs to be improved

- Well maintained systems still suffer from break-ins

- CERT has shown a steady increase over the last four years in incidents handled, vulnerabilities reported and advisories posted

- How does one make systems more secure?
    - Prevent vulnerabilities from occurring
    - Analyze attacks after they occur

Preventing Vulnerabilities

• Complexity and rapid rate of change prevent the thorough auditing of code

(Illustration taken from xkcd)

• Infeasible to prevent computer compromises

Post-Attack Analysis

- Help understand an attack and fix the problem

- Audit logs are maintained by the OS, which log all kinds of user activity on the system

- Two failings of current logging system
    - Integrity
    - Completeness

Lack of Integrity

- “Quis custodiet ipsos custodes” (Who will watch the watchman?)

- Assume that the kernel is trustworthy, while the kernel may have been compromised

- Attackers who subvert the kernel can modify and delete log files to hide their presence

- Custom kernel or boot sector can also be inserted

Lack of Completeness

- Do not log sufficient information to recreate the entire attack

- Difficult to determine the cause and extent of the break-in

- Logging of input does not involve non-deterministic effects so difficult to replay conditions like race conditions

Goals of ReVirt

- Encapsulate the target system inside a virtual system
- Logging software is placed beneath this virtual machine
- Completeness of logger is improved by using techniques such as checkpointing, logging and roll-forward recovery
- Replays the complete, instruction by instruction execution of the virtual machine – before, during and after an attack

Virtual Machine Monitors

• Can be run on another operating system (Type II hypervisors) or directly on the bare hardware (Type I hypervisors)

• In the former, the operating system it is run on is called the host OS and the operating system running inside the virtual machine is called the guest operating system

• VMMs make a better trusted computing base because they are smaller and easier to verify than entire operating systems

OS-on-OS vs. Direct-on-Host

- OS-on-OS – Guest OS runs inside a type II hypervisor virtual machine

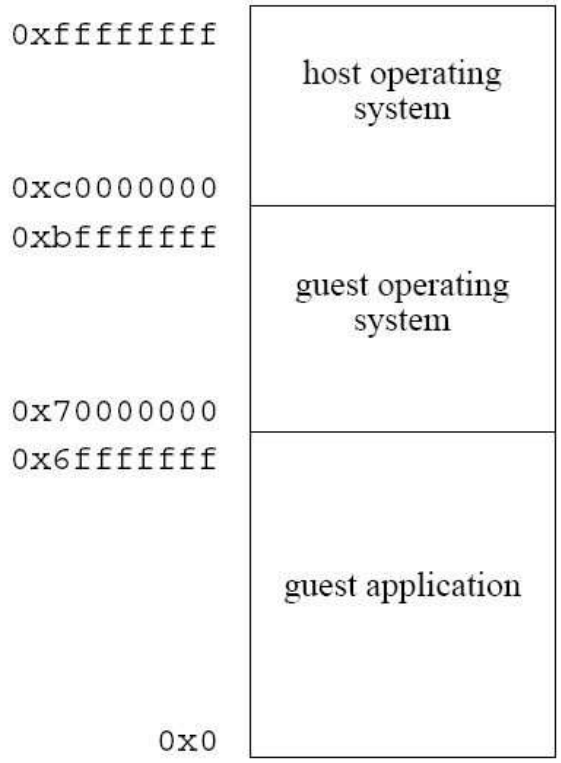
- Direct-on-Host – Target applications run directly on the operating system

- ReVirt uses an OS-on-OS architecture, with UMLinux being the virtual machine that is hosted on a Linux kernel

Operation of UMLinux

- Virtual machine instructions are executed directly on the host CPU

- Memory access is translated using the host's MMU



Privilege Levels of the Guest OS

- VMM maintains whether the current privilege is set to *kernel* (calls by the guest kernel) or *user* (calls by the guest application)
- VMM checks the privilege level at every system call
- When kernel, then the guest kernel made the call
- Passed through to the host kernel
- When user, then the guest application made a system call
- Redirected to guest kernel using SIGUSR1 signal

Trusted Computing Base

- For ReVirt, it consists of the VMM of UMLinux, and the parts of the host kernel that are directly used by the guest OS

- Security of OS-on-OS structure is greater than that of direct-on-host structure

- Smaller attack surface for malicious packets

- In direct-on-host, a vulnerability can lead to a user having access to the entire host OS, while in OS-on-OS an attack is limited in scope to the guest OS

Logging in UMLinux

- All non-deterministic events need to be logged
- Time and external input
- Time is the exact place in the instruction stream where the event took place
- External input is the input that accompanies the event – it could be a key press or a network packet

Process of Logging

- Checkpoint of current state is made by cloning the VM
- Log records are added to a circular buffer in kernel memory and written to disk by a user-level daemon
- ReVirt logs the PC and number of branches since last interrupt to allow it to replay correctly
- branch_retired gives the number of branches since last interrupt
- Hardware interrupts, traps and faults are subtracted to get the number of user level branches

Replay

- Two pass method to find the correct instruction at to which to deliver the interrupt

- In the first phase, branch_retired counter generates an interrupt after most branches

- In the second phase, breakpoints are inserted after every branch, where the current number of branches since last interrupt is compared to the required number

- When it matches, the interrupt is delivered

Cooperative Logging

- Logged network messages generate a lot of data

- Messages do not need to be logged on both the sending and the receiving end

- If the sender is being logged, than on replay it will resend the same message

- Receiver does not need to log the received data

Alternative Architectures for Logging

- Direct-on-host structure is another alternative – less secure and more difficult to implement

- Involves logging and replaying multiple processes on the host system

- Larger number of non-deterministic events

- Scheduling order is difficult to replay with interrupts in the kernel

- Solutions such as defining points in the kernel where interrupts are allowed require significant changes to the kernel

Analysis using ReVirt

- Execution sequence can be replayed – partially or completely

- Internal tools can be used to probe the system or change the current files or debug processes

- Tools such as debuggers and analyzers can be run off-line from the host machine

- May detect discrepancies with the view of the system from inside the virtual machine

Performance and Logging Overhead

- Performance of UMLinux is compared to a native Linux (Table 2)

- Overhead ranges from 1% in CPU intensive tasks to around 58% in tasks that issue a large number of guest kernel calls

- Logging adds a further overhead of not more than 8% (Table 3)

- Log data files range from 0.04 GB/day for CPU intensive workloads to approximately 1.4 GB/day for interrupt heavy workloads

Conclusion

- Encapsulates the system inside a virtual machine and logs the entire execution stream

- Replay execution before, during and after the attack

- Analysis of the logs and replay allows administrators to track and fix vulnerabilities

- Non-deterministic events like race conditions can be simulated

- Reasonable space and time overhead is added

Discussion Time

- Security and Privacy
- Performance and Practicality
- Checkpointing
- Automation
- Multicore performance

Security and Privacy

• Why bother to attack the virtual system in UMLinux? Why not just attack the host system?

• If the guest OS is compromised, and it can communicate with the host OS, isn't the host OS automatically at risk?

• Is OS-on-OS really more secure than the Direct-on-host approach?

• With the system logging all non-deterministic inputs, confidential data and user credentials will also be logged. Isn't this a privacy issue, especially if ReVirt itself is compromised?

Performance and Practicality

- Isn't 58% overhead just way too high a price to pay? What would be an acceptable figure?

- Would moving ReVirt to Dom0 of Xen speed it up?

- Is this practical for a home user? What is the 'normal' workload tested with?

- Is it possible to make it efficient and useful enough for a home environment, where it alerts the user when an intrusion is detected and transfers the logs off-site?

- Is the logging overhead too much? Is some filtering of things logged necessary?

Checkpointing

• Downtime of a few minutes every few days is unacceptable for highly critical services. Is it practical to checkpoint a running system or would the overhead be too high?

• Is it necessary to checkpoint the entire disk of the system, or could it be done more efficiently by just storing a delta from the last checkpoint?

• Could the checkpoints be used for a snapshot kind of functionality?

Automation and Multi-core Logging

- Is there any automated way to discover that an attack has taken place?

- Can we detect attacks on the fly?

- Sifting through the mounds of collected data is like searching for a needle in the haystack. Can finding the intrusion during replay be automated?

- Can it be used to log and replay over multiple cores?