Deep Introspection

Michael E. Locasto
University of Calgary

Joint work with Sergey Bratus and Sean W. Smith
(Dartmouth College)
What is my program doing? Can I trust it?

DEEP INTROSPECTION
“Introspection”

“(from Latin introspicere, "to look within"), the process of observing the operations of one's own mind with a view to discovering the laws that govern the mind.”

- Encyclopedia Britannica (via dictionary.com)
Analogy: Deep Packet Inspection

The security and networking folks among you will note the similarity in name and concept to DPI; this is not accidental.

Both are attempts to peer through layers of abstraction.

Building a recognizer for embedded or tunneled languages.

162 pages of HTTP RFC 2616
Security Policy -> Enforcement

Inspection of code execution to detect “malice”
Such inspection requires a reference monitor
Reference monitor implementations necessarily employ “real” instrumentation primitives for interception

This constrains scale and speed (and thus power and practicality of the reference monitor) [VMSec2008]
DI: Central Observation

There is a cost to breaking through abstraction; a cost to marshalling semantics across an API or component boundary.

Extraction
Aggregation
Recognition
The Model Translation Gap

Abstraction enables system design
Abstract models elide detail

Models assume and entail operations like:
  – Data/state extraction
  – Event aggregation
  – Attribution / identification / context gathering
  – Recognition of patterns or computational constructs

These operations are expensive...
...and sometimes undecidable.
Wanderarounds

Doesn’t matter: algorithm runtime is $O(...)$
Use different hardware (or hardware tricks)
Just use virtual machines [NSPW2010]
Security should be slow / it is a tax / live with it
Find an efficient implementation
Example: Privilege “Rings”
The Cake is a Lie [VMSec2009]
Example: Clark-Wilson

Transformation procedures, integrity verification procedures, CDI, UDI, etc.

Modeled as functions (may not terminate)

Certification required (call out to human)
Example: GDB Watchpoints

Program Supervision

Watch for unexpected behavior
Watch for specific misbehavior or misuse
Trustworthy behavior is expected behavior

Debugging

“Debugging is precisely the activity which establishes the link between expected behavior and observed behavior.” (paraphrasing [VMSec2008])
Counting to 10,000

- with 4 sw checkpoints
- with 4 hw checkpoints
- no supervision
Limited Sight (examine one thread)
Not Enough Hardware (too few resources)
Hardware Too Small (mismatch type req.)
Watches Uninteresting Events (granularity)
Slow (must use simple interrupt mech.)
QUOTES FROM GDB DOCS
“Depending on your system, watchpoints may be implemented in software or hardware. GDB does software watchpointing by single-stepping your program and testing the variable's value each time, which is hundreds of times slower than normal execution. (But this may still be worth it, to catch errors where you have no clue what part of your program is the culprit.)”
Watches Uninteresting Events

“Currently, the awatch and rwatch commands can only set hardware watchpoints, because accesses to data that don't change the value of the watched expression cannot be detected without examining every instruction as it is being executed...”
Hardware Too Small

“Sometimes, GDB cannot set a hardware watchpoint because the data type of the watched expression is wider than what a hardware watchpoint on the target machine can handle.”
Not Enough Hardware

“If you set too many hardware watchpoints, GDB might be unable to insert all of them when you resume the execution of your program. Since the precise number of active watchpoints is unknown until such time as the program is about to be resumed, GDB might not be able to warn you about this when you set the watchpoints, and the warning will be printed only when the program is resumed…”
Limited Sight

“With the current watchpoint implementation, GDB can only watch the value of an expression *in a single thread...*”
Bridging the Gap

Measure inefficiencies
Case by case

Methodology + Facility
A model translation discipline to help researchers decide where to place mechanisms & data + system primitives to more efficiently extract and aggregate data

Develop understanding of computational limit
What Might This Kind of Thing Enable?

Automated Debugging

Fixing production applications on the fly; zero downtime; self-healing

Data Patching

Live testing under real workload
Challenge Problem: Extreme Constraint Enforcement

• Vary:
  – Number of watchpoints
  – Program type
  – Complexity of watchpoints
  – Problem size

• Evaluate 1000’s of constraints / conditions:
  – @ / near machine speed
  – In parallel
  – Without issuing interrupt
Parting Thoughts

Adopt this bias: there is no such thing as inefficient security

Main challenge: a principled way to transport trust-relevant data and events across layers of abstraction, resulting in a tighter coupling between security policy design and physical systems

Maybe I’m just building a better microscope
Questions?

THANKS FOR YOUR ATTENTION
Abstract

Commodity computing systems are often designed to execute code as quickly as possible; they are not designed to *inspect* code very quickly at all. Yet, such inspection provides the basis for most security reference monitors, and implementing security policies based on such measurement still results in poor performance. The key problem seems to be a significant gap between abstract or formal security policy models and the capabilities, peculiarities, and limitations of commodity computing systems. This gap leads to a mismatch between security policy requirements and a system's ability to efficiently (1) trap and aggregate events of interest and (2) extract state relevant to the security decision at hand.

This talk will consider system support for security policy expression, translation, and enforcement collectively called "Deep Introspection": the ability to transport security-relevant information and trust relationships across layers of abstraction in a principled fashion.
Approach :: Event Trapping Model

Relate each model object to real data construct

Relate each procedure to a computational construct (e.g., parser)

Identify code-data ownership relations

[TRUST2010]
Problem Statement

Observations at a particular data granularity require supporting operations and primitives above and beneath (a cross-layer communications pathway)
Problem Statement: “Model Kinematics”

Observations at a particular data granularity requires supporting operations and primitives above and beneath

The study of how to translate a model to a real system --- and understand the resulting behavior
Figure 5-1. Descriptor Fields Used for Protection