Principles-Driven Forensic Analysis

Sean Peisert, Matt Bishop,
Sidney Karin, and Keith Marzullo
UC San Diego and UC Davis

New Security Paradigms Workshop (NSPW) 2005
Lake Arrowhead, CA
September 20–23, 2005

Re-Defining *Forensic Analysis*

forensic analysis: the process of answering the question:

"What happened?"

Tools Used For Forensics Today

- Two Parts of Forensics: *Logging & Auditing* (Gathering, Processing, Examining, Analyzing)
- Logging
  - *syslog, TCPWrappers,* IDS logs, firewall logs, process accounting, keystroke logging, *BSM, ReVirt*
- Auditing
  - *grep/strings*
- Both
  - *BackTracker, Plato, (Tripwire)*

What can't we currently (generally) detect forensically?

1. *User functions* and *variables* used
2. Changes in the *user environment*
3. *Race conditions in memory*
4. *Buffer* and *numeric overflows*
5. *Code injected* into the program instruction stream
6. *Covert channels* through memory reads/writes
7. *Covert channels* through raw disk device at points with unallocated inodes
8. Interception of *user input*
9. *Programmer backdoors exploited*
10. *Code written to the heap* and executed dynamically at runtime

Principles

1. Consider the entire system
2. Assumptions should not control what is logged.
3. Consider the effects of events, not just the actions that caused them.
4. Context assists in interpreting the meaning of an event.
5. Actions and results must be presented in a way that can be analyzed and understood by a human forensic analyst.

What about feasibility?

- Significant performance considerations are obvious.
- We have no desire to fundamentally change the system.
- Importance of logging is how well the data it captures enables auditing.
- Currently concentrating on completeness and efficacy, rather than efficiency and performance. Basis for this is two-fold:
  - We advocate starting from a desired “end state.”
  - Limited, special-purpose systems may tolerate inefficiency.
- Obvious approaches to future solutions include information compression, co-processor-assisted logging, and dedicated hardware.

Implementation Goals

- *Goal:* Collect all data that a human analyst might need. Do this irregardless of: intent of attacker, whether attacker is an insider or outsider, & nature of activities.

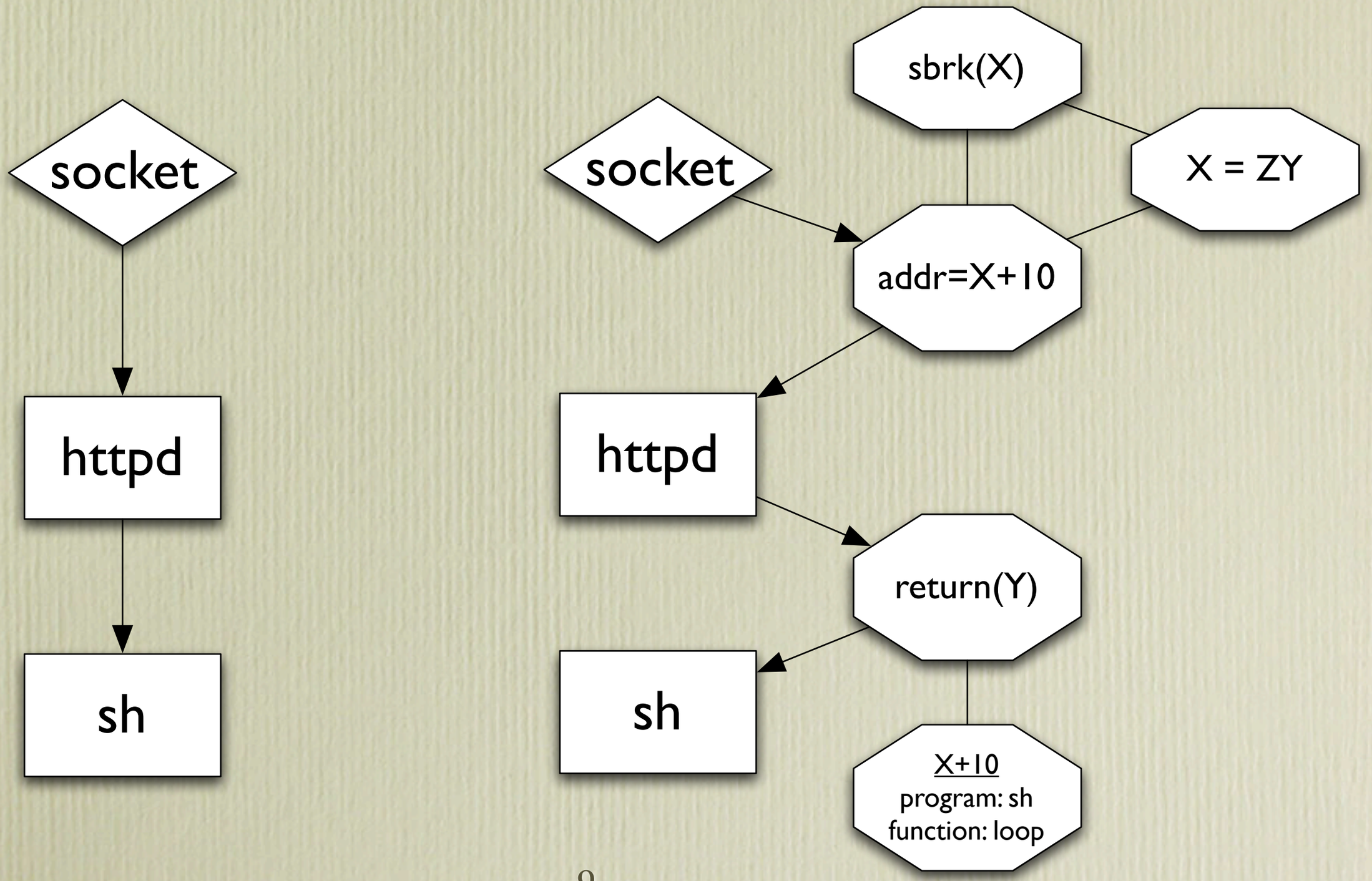
- *Goal:* Automate processing and presentation of that data in a way that makes it easier for a human to understand and direct further processing of later.

- *Goal:* Automate search for certain activities to draw an analyst's attention, with limited "false flags."

Principles-Based Auditing: The Data Model

- *multi-resolution:* "a correlated, layered perspective (state table), encompassing all levels of a system's software state, objects, and events, including memory, network, kernel, disk, applications."
  - Example: Program is composed of functions, variables, etc....
- translate abstraction shortcuts
  - Example: Memory location is 0x2231291
- actions *and* effects
  - Example: Keystroke is 'k', the effect is ??
- context
  - Example: When a command is executed, what is the path searched?

Auditing
Example of Desired Output



Auditing:
Validating Suspicions

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.
- Example #2

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.
- Example #2
  - Event Question: Is a location in memory being written to by a process or thread which does not "own" that location?

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.
- Example #2
  - Event Question: Is a location in memory being written to by a process or thread which does not "own" that location?
  - Labelling Question: Is that a *race condition*? Run models.

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.
- Example #2
  - Event Question: Is a location in memory being written to by a process or thread which does not "own" that location?
  - Labelling Question: Is that a *race condition*? Run models.
- Example #3:

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.
- Example #2
  - Event Question: Is a location in memory being written to by a process or thread which does not "own" that location?
  - Labelling Question: Is that a *race condition*? Run models.
- Example #3:
  - Event Question: Is a process jumping to unexpected places in memory, such as non-word boundaries?

Auditing: Validating Suspicions

- Not real-time intrusion detection, so we can do this repeatedly.
- Example #1: Buffer or Integer Overflow
  - Need to know the sizes of all memory allocations, both static and dynamic, and all memory writes. Compare allocations to writes.
- Example #2
  - Event Question: Is a location in memory being written to by a process or thread which does not "own" that location?
  - Labelling Question: Is that a *race condition*? Run models.
- Example #3:
  - Event Question: Is a process jumping to unexpected places in memory, such as non-word boundaries?
  - Labelling Question: Are those *covert channels*? programming errors? Run models.

Principles-Based Logging: Kernel and System Apps

- Start with existing tools: Instrument kernel to capture:
  - traps & interrupts
  - events involving the filesystem and network stack
  - reads/writes directly to raw devices (disk, network)
  - context

- Instrument shells and other applications (vi, emacs, X Windows)
  - Application execution paths, library paths, user limits, current working directory, keystroke mappings, command aliases

Principles-Based Logging: User Space

- The mechanism is less relevant
- Can use several methods to gather memory traces:
  - Virtual machine introspection (e.g. *ReVirt/BackTracker*)
  - Binary rewriting (e.g. *Eraser*)
  - Compiler instrumentation (e.g. *LLVM "passes")*
    - Symbol table, data types
    - Arguably easier

Summary

- Principles may lead to answers will be more easily proved correct, including for the *insider problem*.
- Techniques based on the principles can enable improvements in forensics: analysts can exhaustively and intelligently view data and validate their suspicions, instead of inferring conclusions from insufficient data.
- Nothing wrong with *inferring* errors, except when it inhibits collecting data.
- Efficiency is a concern, though results can be valuable, even if not generally or widely applicable.
- Proof of efficacy may lead to OS or hardware changes that could do the same thing by using more predictable/computable data, and therefore less recorded data.
- Open research areas remain about best presentation methods and ways of automatically classifying actions

How can we force the use of a particular compiler on everything?

- NetBSD has *verified exec* feature.
- Future operating system could offer this features as well, or may use hardware to enforce it.