Improving Forensic Analysis Through Transaction-Based Security

Sean Peisert
Sid Karin

UCSD/SDSC

Forensic Analysis

- *forensics:* "The use of science and technology to investigate and establish facts in criminal or civil courts of law." (American Heritage Dictionary)
- Forensic analysis helps to recreate past events. As an example, it may be used to determine what an intruder has done to a computer system, and used to try to help recover from the intrusion.
- Problems we seek to address:
  - Forensic analysis may have legal considerations
  - Forensic analysis may be hard.
  - Forensic analysis may require a huge amount of data.

Existing Approaches

- "Coroner's Toolkit" (Farmer & Venema)
  - Gathers existing data and attempts to analyze the state of a system, primarily including "mactimes" and unallocated disk space disk (deleted files).
- *Takedown*
  - Dr. Andrew Gross (fmr. UCSD ECE Ph.D. student) automated and formalized forensic methods that he developed and used with Tsutomu Shimomura to capture Kevin Mitnick.

Forensics & Debugging

- "Forensic analysis" and "debugging" have a lot in common.
- Both attempt to use available evidence to recreate an event, be it an intrusion or a bug.
- Both are aided by a combined approach of instrumenting a system to give the right data, then analyzing the data.
- This could be debugging output, log output, system call traces, etc...

Forensics and Fault-Tolerance

- Fault-tolerance techniques do not involve analysis of *faults*, like debugging, but detecting failures and recovering a system back to a correct state.
- *Checkpointing* stores information which can be used to restart a system. It usually involves saving frequent snapshots of states of the system.
- *Message logging*, a form of checkpointing, involves not only saving checkpoints, but the decisions that were made at non-deterministic points in the code.
- A system can theoretically be implemented using message logging to store less data than standard logs. This may improve forensic

Legal Considerations

- In a legal case, rules for handling evidence demand that a chain of custody be guaranteed. On a computer, sufficient information must be logged to do this.
- Most logging mechanisms can be spoofed. As a technique for defending against spoofing, more information than usual must be recorded to obtain legally-valid data.

Transactions

- *Transaction*: A result-oriented unit of communications processing (Cisco Systems Internetworking Terms and Acronyms)
- *"Transaction"* is a commonly used term among database programmers to describe an interaction with the database server.
- Any SQL query, for example, can be considered a transaction at a certain level, regardless of what the query asks.
- Transactions can frequently be recorded to track changes in case something catastrophic happens to the system and it needs to be restored/reconstructed.
- We consider a transaction to be atomic unit of interaction, from

Transaction-Based Security

- *Transaction-based systems*, more generically, are systems for which some primary aspect of operation is broken into (complete) atomic units.
- *Transaction-based security* is a transaction-based system which uses one or more levels of uniform transaction units for security purposes.
- We consider transactions for forensic analysis.

Real-World Transaction Systems

- Databases and web servers are examples of real-world transaction-based systems.
- In principle, one can recreate events in a database by determining everything that a user has looked at or modified by entering SQL statements.
- Database systems and web servers already support journaling, i.e. "total" software logging.

Journalling vs. Message Logging

- Journaling relates to message logging in that both save data about what happened.
- Message logging saves primarily information from the non-deterministic points in a system
- Journalling can save information about every activity.
- Journalling may be more complete, though message-logging may be more efficient.

WISE

- The *WISE* system considers transactions for access to resources.
- The WISE system does not necessarily consider simply one “level” of action to make use of on a computer system, like a database server does with SQL queries.
- The WISE concept can be applied so that the system could be implemented in as “low” a level as the hardware or as “high” a level as simple human

WISE and Forensics

- Does a WISE-enabled system provide better forensics?
- What data does a WISE interaction create which could be useful for forensics?
- What does the concept of "protected resources" add to forensics?

Basic Questions

- In principle, transaction-based systems in general, like database systems, can record anything. How close can we approach this on an entire computer system?
- How much benefit for forensic purposes do we get by recording more information through WISE?
- Most computer systems are handicapped by the lack of sufficient pertinent information recorded.

Tradeoffs

- Computer security always involves tradeoffs with other elements of a computer system, such as usability and performance.
- We can perform near-perfect forensic analysis if we capture all data. It is impractical to capture all data, though.

So, what do we care about?

- What matters in security?
  - Data Integrity
  - Data Confidentiality
  - System Availability
- What can we do forensically to address the three primary general security issues? What is a threat? What can we analyze? Ultimately, two things:
  - Disk accesses (reads & writes)
  - Network accesses (send/receive/lookup)

What to ask? Some questions that can

- Which files were viewed or modified? How?
- Were programs run? Was a compiler run? Were user-written functions written? What did the programs do?
- Who is involved?
- Was there an interactive session?
- Was there a network access? A DNS lookup?

What information do we have access to?

- Most forensic analysis uses system logs. In principle, we can do more:
  - System calls
  - Library calls (dynamically linked and static)
  - Function calls (if we have the source)
  - File access tables
  - Network traffic

System Calls
“syscalls”

- Intrusion detection has long seen system calls as useful for anomaly detection (Hofmeyer, Forrest and Somayaji)
- Can we use their technique of limiting data just to privileged processes, very specific syscalls, or some other limit, to determine the amount of data necessary?
- Can we utilize their technique of statistical analysis of sequences of system calls?

Syscall Considerations

- Darwin, a FreeBSD derivative, has 331 system calls which programs utilize to access system functions like "open," "fork," "mount," "read," and "exit."
- If we log syscalls, we won't "miss" anything, because they would encompass both the operating system and all applications.
- Which syscalls are most important to forensics?
- What about "covert-channels" that don't use syscalls?

Experiment #1: System Calls

- Set up a BSD system with kernel-loadable modules which records all syscalls and their arguments.
- Run a short, known, simple series of events.
- Attempt to recreate the events using only syscalls and automate the system. How well does it work?
- Follow-up: What can we learn from analyzing for tty sessions?
- Follow-up: Can we determine if just a few specific system calls are necessary (i.e. open, close, and mmap), or all of

Experiment #2: Dynamic Library Calls

- Record all dynamically-made library calls by modifying lib.c.
- Attempt to recreate events. How well does it work?

Experiment #3: Library Call Comparison

- Instrument /dev and /proc to run “truss” on binaries or modify each system call individually using “ld preload”.
- Determine whether library calls are made to dynamic shared libraries or is statically linked into a program.
- Static library calls are a warning flag!

Experiment #4: Function Calls

- User-defined function calls are extremely difficult to capture. We can't easily know the function names and arguments without modifying source code. Modifying source code is dangerous because of memory manipulation.
- Soulution: Java compiler as a proof-of-concept that does not suffer from memory manipulation.
- Another solution: Instrument logging by going through a profiler. It's already built in!
- Attempt to recreate events. How well does it work?

Experiment #5: Binaries Executed

- If non-system binaries are executed, determine whether they are actually just scripts calling system binaries or are user-written.
- Do this by capturing series of "typical" system calls to determine "signatures" of known applications, as Hofmeyer & Forrest did.
- Does this work? Is it effective?

Experiment #6: Network

- Assuming we can obtain all of the information we need about the filesystem from system and library calls, we can look at networks.
- Can we learn enough by logging DNS names queried, ports used, packet types, etc...?
- Can we track these vulnerabilities, among others:
  - Port opened (vulnerability created)
  - DNS queried
  - Packets sent (information leaked)

Experiment #7: Users

- Log the "table of accesses" in realtime to determine which user is doing what.
- Does it help? Is it accurate? Are compromised accounts being used? Does it tell us about compilation?

Experiment #8: Message Logging

- "Message logging" is a popular form of checkpointing in fault-tolerant systems.
- Can we use message logging in non-deterministic conditions to replay an intrusion for forensic purposes with less data than typical logging?
- Can we use the fault-tolerance technique of not displaying system results until they have been properly logged?

Summary

- Forensics can use transaction-based systems to capture the right data.
- Forensics is closely related to both debugging and fault-tolerance and can rely on the previous research towards both.
- Experiments will demonstrate precisely which data needs to be captured and analyzed.
- Analysis of the experiments and related disciplines may show that recording only small amounts of data is practical and viable.