Combining Monitors for Runtime System Verification

Joshua Levy, Hassen Saïdi, and Tomás E. Uribe
System Design Laboratory
SRI International
{levy,saidi,uribe}@sdl.sri.com

OASIS P.I. Meeting
Santa Rosa, California
August 2002
Overview

• Application:
  – Intrusion-tolerant Web server architecture

• Problem:
  – Runtime verification of entire systems
  – Emphasis on security properties

• Approach:
  – Use multiple runtime monitors for diverse features
  – Relate and combine monitors for system verification

• Presented at FLOC Workshop on Runtime Verification (July 2002)
Need for runtime verification

- Systems difficult to analyze completely off-line
  - Complexity and size
  - Descriptions not amenable to formal analysis (e.g. C code)

- Systems rely on third-party components
  - Limited specifications, uncertain behavior
  - Opaque internals can’t be verified before runtime
Need for runtime verification

- Systems difficult to analyze completely off-line
  - Complexity and size
  - Descriptions not amenable to formal analysis (e.g. C code)

- Systems rely on third-party components
  - Limited specifications, uncertain behavior
  - Opaque internals can’t be verified before runtime

- RV compensates for these weaknesses
  - Need only check individual runs, making verification easier
  - Can verify actual behavior of third-party components
System-wide verification

• Verification of an entire system requires a combination of off-line and on-line techniques

• Much static analysis focuses on a single model — but the model may not be accurate

  – Security: Many exploits circumvent the model that was considered at design time
System-wide verification

- Verification of an entire system requires a combination of off-line and on-line techniques

- Much static analysis focuses on a single model — but the model may not be accurate
  - **Security**: Many exploits circumvent the model that was considered at design time

- Need to focus on multiple aspects of system
  - Verify operation at both abstract and concrete levels
  - Use RV to “tie together” models and real system operation
  - RV can help verify that model assumptions are accurate
Approach

• Build multiple monitors for diverse features
  – At different levels of abstraction and granularity
  – High-level events, component communication, code execution, operating system functions, etc.
Approach

• Build multiple monitors for diverse features
  – At different levels of abstraction and granularity
  – High-level events, component communication, code execution, operating system functions, etc.

• Base monitors upon available mechanisms
  – Analysis of system input and output
  – Monitoring of component interfaces
  – Code annotation
  – OS-based monitoring
  – . . .
Approach

• **Build multiple** monitors for diverse features
  – At different levels of abstraction and granularity
  – High-level events, component communication, code execution, operating system functions, etc.

• **Base monitors upon available mechanisms**
  – Analysis of system input and output
  – Monitoring of component interfaces
  – Code annotation
  – OS-based monitoring
  – . . .

• **Combine** them to monitor desired properties
Monitors and properties

- A monitor $M$
  - Observes runs of the system $r : s_0, s_1, s_2, \ldots$
  - Raises alarm
  - Provides information
Monitors and properties

• A *monitor* $M$
  
  – Observes runs of the system $r : s_0, s_1, s_2, \ldots$
  – Raises alarm
  – Provides information

• A *property* $\varphi$ is a set of runs

• For a property $\varphi$, $M$ can be
  
  – *Sound* with respect to $\varphi$: raises no false alarms
  – *Complete* with respect to $\varphi$: raises alarm whenever specification violated
Combining monitors

- *Combination* of $M_1$ and $M_2$:

$$M_1 \oplus M_2$$

Raises alarm whenever $M_1$ or $M_2$ does.
Combining monitors

- **Combination** of $M_1$ and $M_2$:

  $$M_1 \oplus M_2$$

  Raises alarm whenever $M_1$ or $M_2$ does.

- Two questions about combining monitors:
  - How do we relate the properties they check?
  - How do we combine the information they collect?
Relating properties

- Use standard assume-guarantee formalism

- If $M$ is sound and complete with respect to $\varphi_2$ for all runs satisfying $\varphi_1$, write

$$[\varphi_1] M [\varphi_2]$$

- If $M$ is running and $\varphi_1$ holds, then $\varphi_2$ is properly monitored.
Relating properties

- Use standard assume-guarantee formalism

- If $M$ is sound and complete with respect to $\varphi_2$ for all runs satisfying $\varphi_1$, write

$$[\varphi_1] M [\varphi_2]$$

- If $M$ is running and $\varphi_1$ holds, then $\varphi_2$ is properly monitored.

- Assumed properties can be checked by other monitors.

- When combining monitors, formalism helps relate properties checked by each monitor
Simple proof system

Examples:

\[
\begin{align*}
[P_1] M_1 [Q_1], & \quad [P_2] M_2 [Q_2] \\
[P_1 \land P_2] (M_1 \oplus M_2) [Q_1 \land Q_2] & \quad [P] M_1 [\varphi], \quad [\varphi] M_2 [Q] \\
[P] (M_1 \oplus M_2) [\varphi \land Q] & \quad [P_1] M_1 [Q_1], \quad P \rightarrow P_1, \quad Q_1 \rightarrow Q \\
[P] M_1 [Q] & \quad [P_1] M_1 [Q]
\end{align*}
\]
Simple proof system

Examples:

\[
\begin{align*}
[P_1] M_1 [Q_1], & \quad [P_2] M_2 [Q_2] \\
[P_1 \land P_2] (M_1 \oplus M_2) [Q_1 \land Q_2] & \quad [P] M_1 [\varphi], \quad [\varphi] M_2 [Q] \\
& \quad [P] (M_1 \oplus M_2) [\varphi \land Q]
\end{align*}
\]

\[
\begin{align*}
[P_1] M_1 [Q_1], \quad P \rightarrow P_1, \quad Q_1 \rightarrow Q
\end{align*}
\]

\[
\begin{align*}
[P] M_1 [Q]
\end{align*}
\]

More generally,

\[
\begin{align*}
[A_1] M_1 [G_1], \ldots, [A_n] M_n [G_n], \quad (\bigwedge_i (A_i \rightarrow G_i)) \rightarrow (A \rightarrow G)
\end{align*}
\]

\[
\begin{align*}
[A] (M_1 \oplus \cdots \oplus M_n) [G]
\end{align*}
\]
Information flow between monitors

- **Complementary** combination (no communication)
  - Better coverage

- **Dependent** combination
  - Information reuse

- **Correlating** combination
  - Stronger properties

\[ M_1 \oplus M_2 \]
\[ M_1 \oplus M_2 \oplus \cdots \]
\[ M_1 \oplus M_2 \oplus M_3 \]
Complementary combination

- $M_1$ and $M_2$ independently check their properties
- Do not communicate
- But designed to work together
  - One can check the assumptions of the other, e.g.
    \[
    [P] M_1 [\varphi], \quad [\varphi] M_2 [Q]
    \]
    \[
    [P] (M_1 \oplus M_2) [Q]
    \]
  - One can verify that the other is running
Simple example 1: Code annotation and process monitoring

- Source code annotation ($M_A$) verifies correct execution ($P_1$)
  - If code is running in the first place
Simple example 1: Code annotation and process monitoring

- Source code annotation ($M_A$) verifies correct execution ($P_1$)
  - If code is running in the first place

- OS-based monitor ($M_E$) verifies process is running ($P_2$)
  - E.g. /proc kernel information in Unix-type OS
  - This monitor is more secure

- Together, the two monitors complement each other to provide better coverage

\[
\begin{align*}
[P_2] M_A [P_1], & \quad [true] M_E [P_2] \\
[true] (M_A \oplus M_E) [P_1 \land P_2]
\end{align*}
\]
Simple example 2: Resource use and performance

- In many systems, resource use should correlate with performance
  - Resources: memory, CPU use, number of processes, . . .
  - Performance: requests processed, clients connected, . . .

- Unexpected deviations can indicate a bug or compromise
  - Memory leak, password cracker running, . . .
Simple example 2: Resource use and performance

• In many systems, resource use should correlate with performance
  – Resources: memory, CPU use, number of processes, . . .
  – Performance: requests processed, clients connected, . . .

• Unexpected deviations can indicate a bug or compromise
  – Memory leak, password cracker running, . . .

• Combine monitors to catch violations
  – Build monitors for performance and resource use separately
  – Correlate using a third monitor
DIT proxy operation

Single thread of execution in proxy:

- Start: Accept request
- Valid request: Forward request to n servers
- Invalid request: Ask more servers, insufficient agreement
- Valid request: Identify suspicious servers
- Service unavailable: Ask more servers, limit reached
- Insufficient agreement: Insufficient agreement
- Sufficient agreement: Service unavailable
DIT system operation

- Intrusion detection system and challenge-response protocols trigger alerts

- The current *regime* specifies how many servers to query and how many of them must agree

- The regime changes in response to alerts and disagreements

- System includes multiple runtime monitors
  - IDS/network monitoring
  - Proxy software operation
  - Proxy OS operation
Example:
Request forwarding in DIT system

- Desired property:
  \( P \): Proxy always sends requests to correct set of application servers
Example:

Request forwarding in DIT system

• Desired property:
  \( P \): Proxy always sends requests to correct set of application servers

• Identify properties suitable for monitoring
  \( P_1 \): Proxy software is running
  \( P_2 \): In proxy software, each regime transition allowable
  \( P_3 \): Current regime in proxy software is followed
  \( P_4 \): Servers considered compromised are not queried
Example:
Request forwarding in DIT system

- Desired property:
  \( P \): Proxy always sends requests to correct set of application servers

- Identify properties suitable for monitoring
  \( P_1 \): Proxy software is running
  \( P_2 \): In proxy software, each regime transition allowable
  \( P_3 \): Current regime in proxy software is followed
  \( P_4 \): Servers considered compromised are not queried

- Monitors:
  \( M_E \): OS-based execution monitor
    (on proxy)
  \( M_I \): Internal proxy monitor
    (on proxy)
  \( M_N \): Network monitor
    (on IDS box)
Example:
Request forwarding in DIT system

\[ P_1 \land P_2 \land P_3 \land P_4 \rightarrow P \land \begin{array}{c}
[true] M_N [P_3 \land P_4] \\
[true] (M_E \oplus M_I \oplus M_N) [P_1 \land P_2 \land P_3 \land P_4]
\end{array} \]

- Note: soundness of the conclusion depends on the soundness of the premises.

- In practice, \( true \) must be replaced by further assumptions.
Conclusions

• Greater coverage (complementary combination)
  – More coverage at concrete levels
  – Often necessary for checking security properties

• Stronger properties (correlating combination)
  – Monitor properties involving more than one aspect or level of abstraction

• Work reuse and efficiency (dependent combination)

• Convenience
  – Modularity
  – Build monitors from available mechanisms
Related work

- Assume-guarantee [Abadi, Lamport '93]

- Many types of monitoring mechanisms
  - Execution monitoring
    - Code instrumentation [JP\textsubscript{AX}], [MaC], [Alamo], . . .
    - Type safety for C [StackGuard], [CCured], [Cyclone], . . .
  - Intrusion detection
    - Specification-based IDS [Ko, Ruschitzka, Levitt '97], . . .
    - Anomaly- and attack-based IDS [EMERALD], . . .
  - . . .

- Enforceable properties [Schneider '98]
Future Plans

• Develop formalism, explicitly address abstraction

• Relate to system design
  – Design assurance argument
  – Design system top-down, assemble monitors bottom-up
  – Synthesis of monitor assembly?

• From detection to enforcement
The End