Run-time Monitoring Theory

Yannis Mallios
18-732
Spring 2015

Slides by Lujo Bauer
Logistics

- **Assignment 1 due tonight**
  - 4 late days max for this assignment
  - 10 overall

- **First exam on Tuesday**
  - Everything covered until today
  - One sheet of letter-sized paper with your own notes (i.e., handwritten, front and back, with your name on top)
Previously *(Lecture 1)*

Topics

0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement
Run-time Security Enforcement

- Enforce security using mechanisms that monitor systems as they *execute* (as opposed to *static* analysis of code)

- Some examples
  
  - *Stackguard*: Dynamic checks to prevent buffer overflows
  
  - *Taint Analysis*: Information flow control, malware detection
  
  - Stack inspection, firewalls, applet sandboxing, displaying security warnings, ...

• Lecture 3

• Lecture 6

• Lectures 4, 5
Previously (Lecture 1)

Learning Outcome

- Understanding of fundamentals of run-time enforcement of security properties
  - What class of security policies can be enforced using run-time monitoring?
  - How do we specify policies for run-time monitors to enforce?
Enforcement in Software

- **Static analyses**
  - Type checking, bytecode verification

- **Run-time enforcement**
  - Java stack inspection

- **Program rewriting**
Enforcement in Software

- **Static analyses**
  - Type checking, bytecode verification

- **Run-time enforcement**
  - Java stack inspection

- **Program rewriting**
Run-time Enforcement: Program Monitors

- A program monitor is a coroutine that runs in parallel with an untrusted application.
- Monitors process security-relevant actions:
  - decide to allow/disallow application actions
  - Disallow, e.g., by terminating application
Run-time Enforcement: Program Monitors

Application
fopen(f)
(Applet)

Monitor
fopen(f) is OK

Machine
fopen(f)
(JVM)
Common Monitor Examples

- File access control
- Firewalls
- Stack inspection
- Bounds checks on input values
- Security logging
- Displaying security warnings
- Operating systems and virtual machines
- ...

...
Program Monitors: Theory & Practice

- **Theory**
  - What is a program monitor?
  - What is a policy?
  - What policies can monitors enforce?

- **Practice**
  - How do we implement program monitors?
  - How do we specify policies?
An Abstract Model: Security Automata

- Formal models of program monitors
- Observe/monitor a sequence of untrusted actions
An Abstract Model: Security Automata

![Diagram showing abstractions and monitoring with fopen(f) is OK]
An Abstract Model: Security Automata

Application generates actions to be input into monitor

Automaton executes actions approved by monitor

Monitor

fopen(f) is OK

fopen(f)
An Abstract Model: Security Automata

Application generates actions to be input into monitor

Automaton executes actions approved by monitor

Abstraction

Monitor

fopen(f) is OK

fopen(f)
Security Automata

- $\sigma$: an execution (a sequence of actions)
- $\Psi$: universe of all possible sequences
  - Finite or infinite
- $\Sigma_S$: subset of $\Psi$ corresponding to the executions of target $S$
Security Automata

- **Deterministic finite- or infinite-state machine**
  \((Q, Q_0, \delta)\)
  - \(Q\) = state set
  - \(Q_0 \subseteq Q\) = set of start states
  - \(\delta : (Q \times I) \rightarrow Q\) = transition function
  - \(I\) = set of input symbols

- An action is “allowed” if a transition exists for it
Security Automata: Example

- “No send after file read” policy
- $I = \{ \text{FileRead, Send, FileWrite} \ldots \}$
- $Q = \{ \text{start, FR} \}$
- $Q_0 = \{ \text{start} \}$
Security Automata: Example

- $\sigma = \text{Open; FileWrite; FileRead; Send}$
Program Monitors: Theory & Practice

- **Theory**
  - What is a program monitor?
  - What is a policy?
  - What policies can monitors enforce?

- **Practice**
  - How do we implement program monitors?
  - How do we specify policies?
Policies

- “A security policy is specified by giving a predicate on sets of executions. A target $S$ satisfies security policy $P$ if and only if $P(\Sigma_S)$ equals true.” [Schneider]
Policies and Properties

- Policies:
  \[ P(\Sigma) \]

- Properties:
  \[ P(\Sigma) : (\forall \sigma \in \Sigma : \hat{P}(\sigma)) \]

- Policies hold on sets
- Policies are properties if they can be defined by predicates that hold on individual executions
Property or Not?

- Target does not send data after reading a file
  - “No send after file read”
- Target outputs a single random number
Property or Not?

- Target does not send data after reading a file
  - “No send after file read”
- Target outputs a single random number
Property or Not?

- Target does not send data after reading a file
  - “No send after file read”
- Target outputs a single random number

```c
int getRandonNumber()
{
    return 4; // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Policies and Properties

- **Policies**
  - Cryptographic uniformity
  - Nonce uniqueness

- **Properties**
  - Access control
Program Monitors: Theory & Practice

- **Theory**
  - What is a program monitor?
  - What is a policy?
  - What policies can monitors enforce?

- **Practice**
  - How do we implement program monitors?
  - How do we specify policies?
Enforceable Properties

- Monitors cannot predict the future

- Prefix closure
  - All prefixes of a “good” sequence are “good”
  - If a sequence is “bad”, all its possible extensions are also “bad”
Enforceable Properties

- Safety properties $\Gamma \subseteq \Psi$
  \[ \sigma \notin \Gamma \Rightarrow \exists i: (\forall \tau \in \Psi: \sigma[..i]\tau \notin \Gamma) \]
Enforceable Properties

- Safety properties $\Gamma \subseteq \Psi$
  
  $\sigma \notin \Gamma \Rightarrow \exists i: (\forall \tau \in \Psi: \sigma[..i]\tau \notin \Gamma)$

- Informally:
  - No “bad thing” will happen
  - “Bad things” cannot be undone

- Schneider’s security automata enforce exactly the set of safety properties
Enforceable Properties

- We formally defined monitors and policies, but what do we mean by “enforce”?

- Implicit in the operational semantics of security automata
Principles of Enforcement

- **Principle of Soundness**
  - All observable outputs obey the policy
  - The monitor always does the “right” thing

- **Principle of Transparency**
  - Semantics of executions that already obey policy must be preserved
  - If the target wants to do the “right” thing, the monitor will respect that
Program Monitors: Theory & Practice

- Theory
  - What is a program monitor?
  - What is a policy?
  - What policies can monitors enforce?

- Practice
  - How do we implement program monitors?
  - How do we specify policies?
Enforcing More Properties

- Give monitors more power (operational semantics)
  - Halt, OK, ...

- Give monitors more latitude (definition of enforcement)
  - Enforce precisely, conservatively, ...
Edit Automata

- Accept the action
- Halt the application
- Suppress (skip) the operation
- Insert some computation

- Monitors are sequence transformers
Edit Automata

- Single step (determined by $\delta$):
  
  $$(\sigma_{\text{in}}, q) \xrightarrow{\sigma_{\text{out}}} (\sigma_{\text{in'}}, q')$$

- Output sequence is observable
- Input sequences are not observable
Principles of Enforcement

- **Principle of Soundness**
  All observable outputs obey the policy

\[
\forall \text{ sequences } \sigma_{\text{in}} . \exists \text{ state } q'. \exists \text{ sequence } \sigma_{\text{out}}
\]

1. \((\sigma_{\text{in}}, q_0) \Rightarrow (\text{empty}, q')\)
2. \(P(\sigma_{\text{out}})\)

- **Principle of Transparency**
  Semantics of executions that already obey policy must be preserved

3. \(P(\sigma_{\text{in}}) \Rightarrow (\sigma_{\text{in}} \equiv \sigma_{\text{out}})\)
Semantic Equivalence ($\equiv$)

- **Necessary properties**
  - Reflexive, symmetric, transitive
  - $\sigma \equiv \sigma' \Rightarrow P(\sigma) \leftrightarrow P(\sigma')$

- **Examples**
  - $\text{fclose}(f);\text{fclose}(f) \equiv \text{fclose}(f)$
  - $\text{fopen}(f);\text{fopen}(g) \equiv \text{fopen}(g);\text{fopen}(f)$
Conservative Enforcement

- Automaton satisfies Soundness but not necessarily Transparency

- ∀ properties P .
  
  (∃ sequence σ . P(σ)) \Rightarrow
  
P can be conservatively enforced
Precise Enforcement

- Automaton satisfies Soundness and Transparency

- Valid sequences cannot be altered in any way, at any time
Effective Enforcement

- Automaton satisfies Soundness and Transparency

- Valid sequences can be altered (with certain conditions)
Enforcement Types

- What type of enforcement is practical?

Precise
Effective
Conservative

Range of enforceable policies
Edit Automata: The Banana Market

- **Set of application actions:**
  
  \[
  A = \{ \text{take(n)}, \quad \text{// take n bananas} \\
  \quad \text{pay(n)}, \quad \text{// pay for n bananas} \\
  \quad \text{browse,} \quad \text{// browse for bananas} \\
  \quad \text{receipt} \quad \text{// commit} \\
  \} 
  \]

- **Edit Automaton:**

![Diagram of Edit Automaton](image)
Edit Automata and Effective Enforcement

- A banana market policy
  - \( \text{browse}^*; ((\text{take}(n);\text{pay}(n) \mid \text{pay}(n);\text{take}(n)) \ ; \text{receipt})^* \)
  - \( \text{take}(n);\text{pay}(n) \equiv \text{pay}(n);\text{take}(n) \)
  - An edit automaton can \textit{effectively} enforce this policy:
    - Satisfies Soundness
    - Satisfies Transparency
    - Proofs are by induction over the possible inputs
  - Less powerful automata (truncation, suppression and insertion) cannot enforce the banana market property
    - Proof by contradiction shows either Soundness or Transparency will be violated
What Can Be Enforced?

- The enforceable properties depend upon
  - the definition of enforcement (conservative, effective, precise)
  - the class of automaton (truncation, suppression, insertion, edit)
  - the space of possible input programs
    - if the monitor can assume certain “bad” executions do not occur, it can enforce more properties
    - static program analysis (type systems; proof-carrying code) can constrain program execution in ways useful to run-time monitors
More Expressive Models?
More Expressive Models?

- Non-properties policies?
- Can the monitor insert EVERY action it wants?
  - Real-time response?
  - Secrets not known to the monitor
- Incomplete mediation?
- Knowledge about the target or inputs?
- Monitor circumvented?
Effective Enforceable Properties
Program Monitors: Theory & Practice

- **Theory**
  - What is a program monitor?
  - What is a policy?
  - What policies can monitors enforce?

- **Practice**
  - How do we implement program monitors?
  - How do we specify policies?
Security Automata SFI Implementation

- Monitor is specified as a security automaton
- Monitor is inlined into code
  - Checks whether operation is allowed
- Java & C implementations
- Operates on object code
  - Java bytecode, x86 assembly language

Rewritten Target
  Monitor
  Open(f,"w")

Program Monitor
  Open(f,"w")
is OK

Executing System
  Open(f,"w")
Inlining an Automaton

- “Push exactly one value onto a stack before returning” policy
Inlining an Automaton

- Insert security automata
- Evaluate transitions
- Simplify automata
- Compile automata

(Figure from Erlingsson & Schneider, SASI Enforcement of Security Policies, 1999.)
Specifying SASI Policies

- **Specification tied to object code**
  - Java is high level
    - `new java.net.URLConnection`
    - Application-level abstractions
  - C (x86 assembly) is low level
    - `mov ... , jne ...`
    - Must create own abstractions
Anti-circumvention Policies

- Monitor must enforce *complete mediation*
  - Must prevent *all* ways of reaching “bad” action

- **Must prevent target from:**
  - modifying security automaton state
  - circumventing automaton code
  - modifying its own code or jumping to code other than its own

- Can be enforced using SASI itself

- Observation: In Java much of this comes for free!
SASI Retrospective

- Automata make a poor specification language
  - E.g., implementing counters

- Acceptable performance
  - Run-time overhead is low
  - Not quite as good as special-purpose mechanisms, but close

- x86 assembly poor for specifying policies
  - Very expressive
  - Far from application-level abstractions
Wrap-up: Program Monitors

- Program monitors commonly used in enforcement

**Theory**
- What is a program monitor?
- What is a policy?
- What policies can monitors enforce?

**Practice**
- How do we implement program monitors?
- How do we specify policies?
Sources

Edit Automata and Effective Enforcement

**Theorem:** Any decidable predicate P on executions is a property that can be *effectively* enforced by some edit automaton

- Proof: construct a transactional edit automaton that suppresses and logs program actions when \( \neg P(S) \) and commits (outputs) when \( P(S) \), for every initial sequence of actions S in a program execution