Enforcing Security Policies

- Rahul Gera

Brief overview

- Security policies and **E**xecution **M**onitoring.
- Policies that can be enforced using EM.
- An automata based formalism for specifying those security policies.
- Mechanisms for enforcing policies defined by the automata.

Security Policy

- The paper uses the definition of a security policy as “A *security policy* defines execution that, for one reason or another, has been deemed unacceptable”.
  - *access control :-* restrict what operations principals can perform on objects.
  - *information flow :-* restrict what principals can infer about objects from observing system behavior.
  - *availability:-* restrict principals from denying others the use of a resource.

More application Specific Security policies

- The paper also tries to address more specific applications like eCommerce in which a policy could say that if a customer pays for a service then in no execution does the seller not provide the service.

Execution **M**onitoring.

- The paper talks about an enforcement mechanisms that work by monitoring execution steps of some system, called the ***target***, and terminating the **target's** execution if it is about to violate the security policy being enforced.

What are bounds on EM

- **Targets** may be objects, modules, processes, subsystems, or entire systems.
- The execution steps monitored may range from fine-grained actions (such as memory accesses) to higher-level operations (such as method calls) to operations that change the security-configuration and thus restrict subsequent execution.

Non EM mechanisms

- Mechanisms that use **more information** than would be available only from observing the steps of a target's execution are, by definition, **excluded** from EM.
- Information provided to an EM mechanism is thus insufficient to predict future steps the target might take, alternative possible executions, or all possible target executions.
- Compilers and theorem-provers, which analyze a static representation of a target to deduce information about all of its possible executions, are **not EM mechanisms.**
- Mechanisms that modify a target before executing it. Although, these can be studied further **IF** The modified target is equivalent to the original, except for aborting executions that violate the security policy of interest. A definition for equivalent is thus required to analyze this class of mechanisms.

CHARACTERIZING EM ENFORCEMENT MECHANISMS

- ψ denotes a universe of all possible finite and infinite execution sequences.
- A target S defines a subset Σ_S of ψ corresponding to the executions of S.

Definition is terms of ψ

- **Definition of Security Policy**: A *security policy* is specified by giving a predicate on sets of executions. A target *S satisfies* security policy P if and only if $P(\sum_S)$ equals *true*.

- But given a security policy P and two sets of executions A and B.
  - if P(A) is true and B is a subset of A does not imply P(B) is true. E.g. information flow.

Safety Property

- A set of executions is called a *property* if set membership is determined by each element alone and not by other members of the set.
- As a result we cant categories information flow as a it cant be defined in the isolation of a single execution.
- Also, its not possible to enforce a policy in which
  - P^ (A) is true
  - But P^(A') is false
  - A' is the prefix to A which is some finite or infinite execution.

Safety Property

$$\mathcal{P}(\Pi)\colon (\forall \sigma \in \Pi\colon \hat{\mathcal{P}}(\sigma)) \tag{1}$$

$$(\forall \tau' \in \Psi^{-}\colon \neg\hat{\mathcal{P}}(\tau') \Rightarrow (\forall \sigma \in \Psi\colon \neg\hat{\mathcal{P}}(\tau'\sigma))) \tag{2}$$

$$(\forall \sigma \in \Psi\colon \neg\hat{\mathcal{P}}(\sigma) \Rightarrow (\exists i\colon \neg\hat{\mathcal{P}}(\sigma[..i]))) \tag{3}$$

- **Non EM-Enforceable Security Policies:** If the set of executions for a security policy P is not a safety property, then an enforcement mechanism from EM does not exist for P.
- This enables us to say that if EM enforces P' and P' -> P then EM enforces P.
- And an aggregate of policies can be taken as a conjunction of individual policies.

Policies we started with

- **Access control** defines safety properties. The set of proscribed partial executions contains those partial executions ending with an unacceptable operation being attempted.
- **Information flow** does not define sets that are properties, so it does not define sets that are safety properties. Not being safety properties, there are no EM enforcement mechanisms for exactly this policy.
- **Availability**:- if taken to mean that no principal is forever denied use of some given resource, is not a safety property as any partial execution can be extended in a way that allows a principal to access the resource, so the defining set of proscribed partial executions that every safety property must have is absent. If availability is defined to rule out all denials in excess of *MWT* seconds (for some predefined Maximum Waiting Time parameter *MWT*). This is a safety property; the defining set of partial executions contains prefixes ending in intervals that exceed *MWT* seconds during which a principal is denied use of a resource.

Security Automata

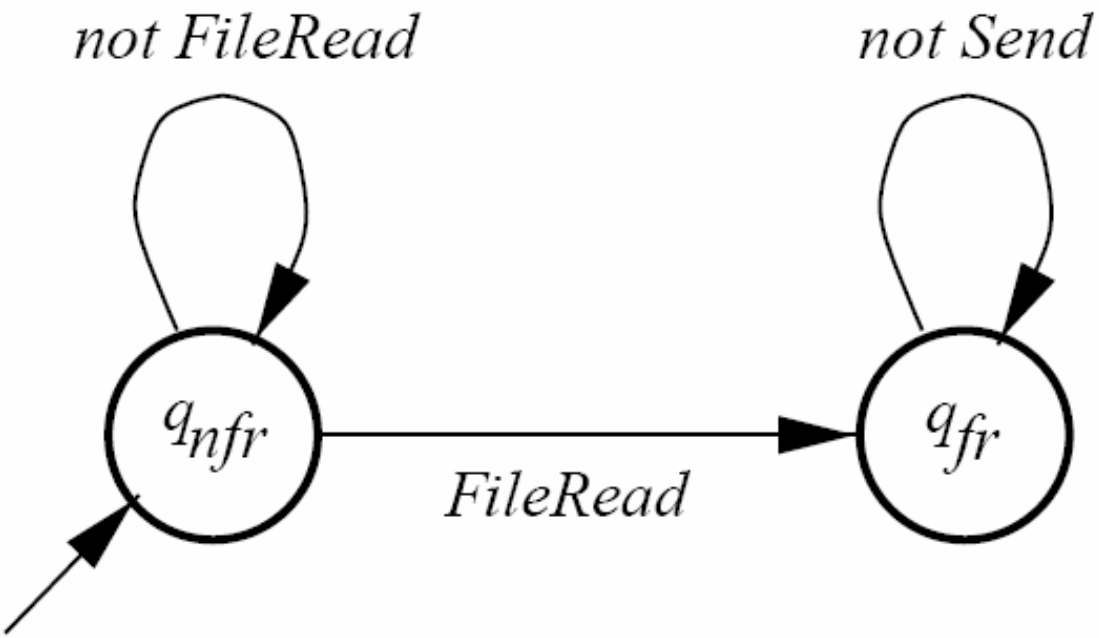

Fig. 1. No *Send* after *FileRead*.

The guard and the Command.

state vars $state : \{0, 1\}$ **initial** 0

transitions **not** $FileRead \wedge state = 0 \longrightarrow$ **skip**

$FileRead \wedge state = 0 \longrightarrow state := 1$

not $Send \wedge state = 1 \longrightarrow$ **skip**

Fig. 2. Alternative specification for policy: No *Send* after *FileRead*.

Example 2 :- Access control

- Principals p
- Rights r
- Objects o
- Oper(p,o,r)
- AddRight(p,p’,o’,r’)
- RmvRight(p,p’,o’,r’)
- AddP(p,p’)
- RmvP(p,p’)
- AddO(p,o’)
- RmvO(p,o’)

state vars P : **set of** $PRINS$ **initial** $\emptyset$
O : **set of** $OBJS$ **initial** $\emptyset$
A : **set of** $\langle s : PRINS;\ o : OBJS;\ r : RIGHTS\rangle$ **initial** $\emptyset$

transitions $Oper(p, o, r) \ \wedge \ \langle p, o, r\rangle \in A \longrightarrow$ **skip**

$$AddRight(p, p', r', o') \ \wedge \ \langle p, o', \text{cntrl}\rangle \in A \ \longrightarrow \ A := A \cup \{\langle p', o', r'\rangle\}$$

$$RmvRight(p, p', r', o') \ \wedge \ \langle p, o', \text{cntrl}\rangle \in A \ \longrightarrow \ A := A - \{\langle p', o', r'\rangle\}$$

$$AddP(p, p') \longrightarrow P := P \cup \{p'\}$$
$$O := O \cup \{p'\}$$
$$A := A \cup \{\langle p, p', \text{cntrl}\rangle\}$$

$$RmvP(p, p') \ \wedge \ \langle p, p', \text{cntrl}\rangle \in A \longrightarrow P := P - \{p'\}$$
$$O := O - \{p'\}$$
$$A := A - \{\langle p', \hat{o}, \hat{r}\rangle \mid \hat{o} \in O \wedge \hat{r} \in RIGHTS\}$$

$$AddO(p, o') \longrightarrow O := O \cup \{o'\}$$
$$A := A \cup \{\langle p, o', \text{cntrl}\rangle\}$$

$$RmvO(p, o') \ \wedge \ \langle p, o', \text{cntrl}\rangle \in A \longrightarrow O := O - \{o'\}$$
$$A := A - \{\langle \hat{p}, o', \hat{r}\rangle \mid \hat{p} \in P \wedge \hat{r} \in RIGHTS\}$$

Fig. 3. Access control.

Example 3:- Fair commerce transaction

state vars $state : \{0, 1\}$ **initial** 0

transitions **not** $Pay(C) \wedge state = 0 \longrightarrow$ **skip**

$Pay(C) \wedge state = 0 \longrightarrow state := 1$

$Serve(C) \wedge state = 1 \longrightarrow state := 0$

Fig. 4. Security automaton for fair transaction.

- This however does not have the power to specify guaranteed service after payment.

Mechanisms for enforcing policies defined by the automata.

- The target is executed in tandem with a simulation of the security automaton.
  - The initialization of the target causes an initialized instance of the security automaton simulation to be created.
  - Each step that the target is about to take generates an input symbol, which is sent to that simulation:

1. If the automaton can make a transition on that input symbol, then the target is allowed to perform that step and the automaton state is changed according to its transition function.
2. If the automaton cannot make a transition on that input symbol, then the target is terminated (for having attempted to violate the security policy).

Implicit Assumptions

- Bounded memory.
- Target Control. This is not always possible like in the case of Real Time Availability if MWT is in seconds.
- Enforcement Mechanism Integrity (isolation).

Automaton Simulation Pramatics.

- **Automaton Input Read:** A mechanism to determine that an input symbol has been produced by the target and then to forward that symbol to the security automaton simulation.
  - If the program counter is taken to be an input symbol a symbol would be created each time a machine languare instruction is executed and this will be quite costly.
  - Hardware traps can be very useful.
- **Automaton Transition:** A mechanism to determine whether the security automaton can make a transition on a given input and then to perform that transition.

- Optimization on Input read :-Input symbols are not forwarded to the security automaton if the state of the automaton just after the transition would be the same as it was before the transition.
- This mechanism can also be enforced in a virtual machine setting . The virtual machine instruction-processing cycle is augmented so that it produces input symbols and makes automaton transitions according to either an internal or an externally specified security automaton.

Beyond EM

- *Response to Violations*. Instead of Terminating a target that is about to violate a security policy simply notify the target that an erroneous execution step has been attempted. The target could then substitute another step and its execution might then continue.
- *Program Modification*. The overhead of enforcement can be reduced by merging the enforcement mechanism into the target. One such scheme is *software-based fault isolation* (SFI), also known as "sandboxing" [Wahbe et al. 1993; Small 1997]. In the case of memory protection, a program is edited before it is executed, and only such edited programs are executed by the target. The edits insert instructions to check and/or modify the values of operands, so that illegal memory references are never attempted. SFI is not in EM because SFI involves modifying the target, and such modifications are not permitted of enforcement mechanisms in EM.

- *Program Analysis*. P*roof carrying code* (PCC) [Necula 1997]. With PCC, a proof is supplied along with a program, and this proof comes in a form that can be checked mechanically before running that program. The security policy will not be violated if, before the program is executed, the accompanying proof is checked and found to be correct. To extend PCC for security policies that are specified by arbitrary security automata, a method is needed to extract proof obligations for establishing that a program satisfies the property given by such an automaton. Such a method does exist—it is described in Alpern and Schneider [1989].

QUESTIONS???