The Harrison-Rizzo-Ullman result

by
Michael Rothstein

Kent State University
given 11-30-2005

Overview

Access Control and the Access Control Matrix

Protection Systems

Turing Machines

Emulating a Turing Machine with a Protection System

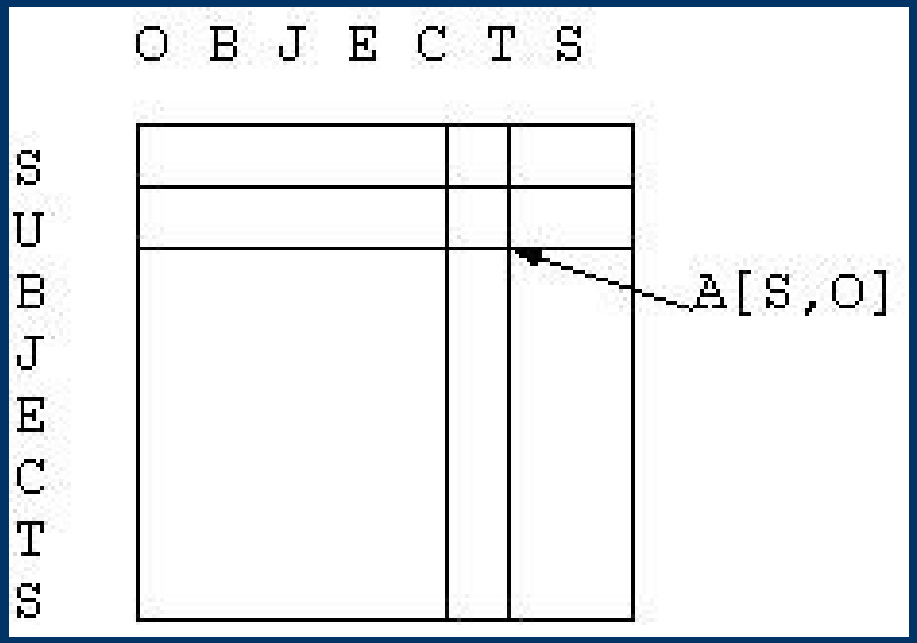
Undecidability Result

Policies and a decidable class of Protection Systems

Access Control

- One issue in computer security is controlling the set of allowed operations of different users and programs on the computer's resources.
- We often need a procedure to tell us whether a subject (user, program) has the right to perform an operation on an object.
- We use a predicate a(subject,object,right)

Representing the access predicate



- The access predicate, a[s,o] is usually represented as a 2-d matrix called the accessibility matrix.
- The entries in the matrix are subsets of the set of rights representing the allowed operations.

An example

	File1	File2	File3	File4
Joe	R W X O			R
Sally	R X	R W O	W	R W O
Alice		R	O	R W

- In this example, Joe owns, can read, write and execute File1 and read File4;
- Sally owns File2 and File4, can read all files except File3, ...
- Though not shown, subjects are also objects, so that we can also talk about the rights a subject has over another subject.

- The accessibility matrix does not remain static; most of the time:
  - New objects are created
  - New subjects are added or created.
  - Rights of subjects on objects are modified (e.g.
    - chmod, changing permissions, etc.
- Need a model for this situation.

Protection Systems

A protection system consists of:

- A set of rights.
- An initial set of subjects, an initial set of objects and an initial accessibility matrix, called an initial Protection State
- A finite set of commands which consist of a name which is invoked with a definite number of parameters which can denote subjects or objects.

Protection Systems (continued)

- These commands are interpreted as a sequence of elementary commands executed on the parameters, subject to the conjunction of the presence of certain rights in the accessibility matrix.
- The elementary commands are:
  1) Enter right r into matrix entry a[s,o]
  2) Delete right r from matrix entry a[s,o]
  3) Create subject s (no rights added)
  4) Create object o (no rights added)
  5) Destroy subject s
  6) Destroy object o

  where s denotes a subject, o denotes an object and r denotes a right.

Some Unix examples

Create a file(p,d,f):
(in directory d)
If w in a[p,d]
1) Create object f in d.
2) Enter own into a[p,f]
3) Enter r into a[p,f]
4) Enter w into a[p,f]

Create a process(p,q)
1) Create subject(q)
2) Enter own into a[p,q]
3) Enter r into a[p,q]
4) Enter w into a[p,q]
5) Enter r into a[q,p]
6) Enter w into a[q,p]

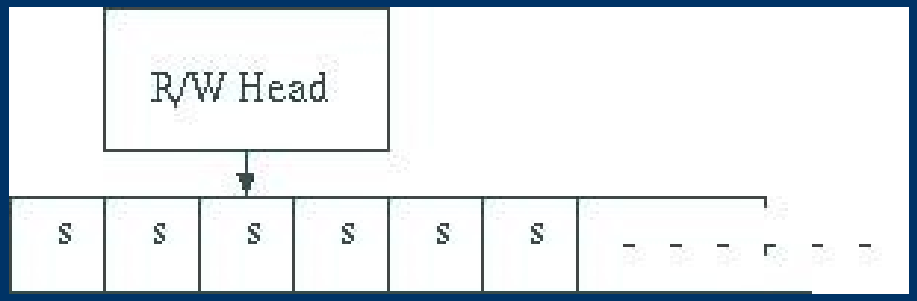
What about safety?

- Given a protection system, we would like to know whether “something unpleasant” may happen.
- One way to define “something unpleasant” is to say that a subject may not acquire a certain right over an object. In that case we will say that a “leak” has occurred.
- We shall see that we cannot determine whether a leak will occur. For that, we turn to Turing Machine theory.

A Turing Machine

- Is a model of computation, designed in the 1930's to explore the limits of computers.
- It was designed to be simple in order to make reasoning about it easier.
- It has approximately the same computational power of any computer.
- Surprisingly, not all problems can be solved with a Turing machine:
- The Halting problem, that is, designing a Turing machine which will determine whether an arbitrary Turing Machine, given an arbitrary input, will go into an infinite loop, is impossible.

A Turing Machine



A Turing machine is:

1) A tape of rectangles stretching to the right indefinitely. Each rectangle contains a symbol, most of which are “blanks” □.
2) A read/write head that is “looking” at one of the rectangles and can be in one of a finite number of “states”.
3) A transition function which determines the computation of the Turing Machine

Formal Definition of a Turing Machine

A Turing Machine is completely specified by a sextuple $(Q,\Sigma,\Gamma,\delta,q0,H)$ where

- Q is a set of states
- Σ is the input alphabet. It does not include the blank symbol $\square$.
- Γ is the working alphabet, which contains Σ and the blank symbol $\square$. It may contain other symbols.
- δ is the transition function (described on next slide)
- $q0$ is the starting state.
- H is the final, or halting, state.

The transition function δ

δ Is a function which maps a state and a symbol on the tape to a triple consisting of

- The new state of the Turing Machine
- The symbol written on the tape (it may be the same or a different symbol than the one before).
- Whether the head moves to the right or the left.

We will assume that the transition function has been written in such a way that the head will not move off the left side. The alternative is that, if the machine is on the first rectangle and the action requires moving to the left, the head will not move. Not really a restriction: we will show later how to change one into the other.

A Turing Machine Computation

1) The input is placed on the tape, left justified; formatting may be added.
2) The read/write head is placed on the leftmost rectangle in the special “start” or initial state.
3) The transition function is repeatedly applied to the current state of the read/write head and the symbol being examined on the tape.
4) The computation halts if the read/write head gets placed in the special “halt” state.

A Turing Machine Computation
An example: add 1 in Binary

- The following describes a machine that adds 1 in binary to the number on the input tape. The number should be placed, least significant bit first, on the tape.
- The symbols to be used are 0,1 (to represent the number) and the special blank symbol denoted by □.
- There will be 2 states, denoted by W and H; W will be the start state, H is the halt state.
- The transition function is given by:

$\delta(W,0) \rightarrow (H,1,R)$ (If the leftmost digit is 0, changing it to 1 is all)

$\delta(W,1) \rightarrow (W,0,R)$ (If we have a 1, 1+1 = 0, carry 1..)

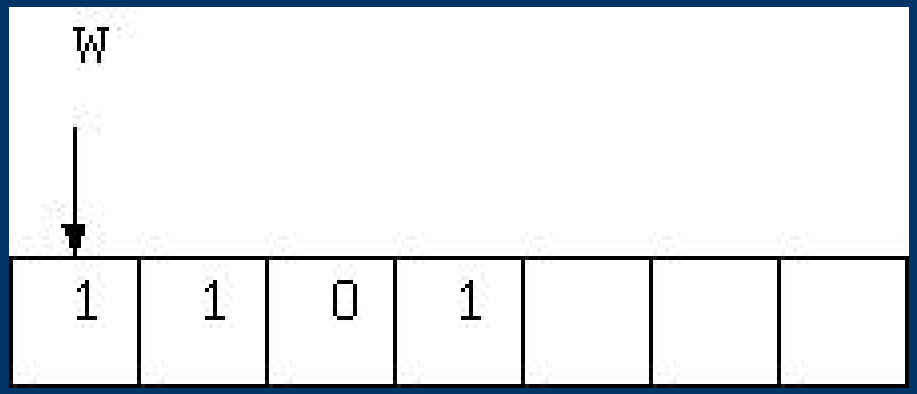
$\delta(W,□) \rightarrow (H,1,R)$ (If we carried to the end, write 1 and halt.)

Formally, this machine is given by:

$(\{W,H\},\{0,1\},\{0,1,□\},\delta,W,H)$

where δ is as defined above.

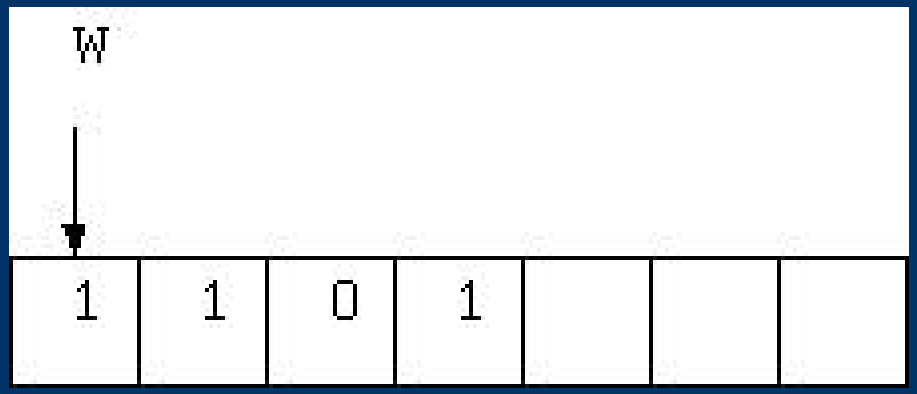
Adding 1 to 1011 with a Turing Machine



Start; place the input on the tape, least significant bit to the left.

Set the state of the machine to the start state W.

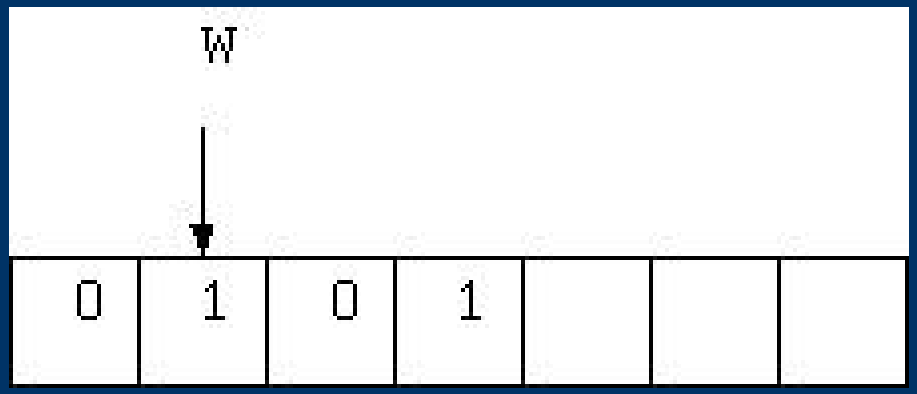
Adding 1 to 1011 with a Turing Machine



Step 1: we use the transition:

- $\delta(W,0) \rightarrow (H,1,R)$
- $\delta(W,1) \rightarrow (W,0,R)$ ◄
- $\delta(W,\square) \rightarrow (H,1,R)$

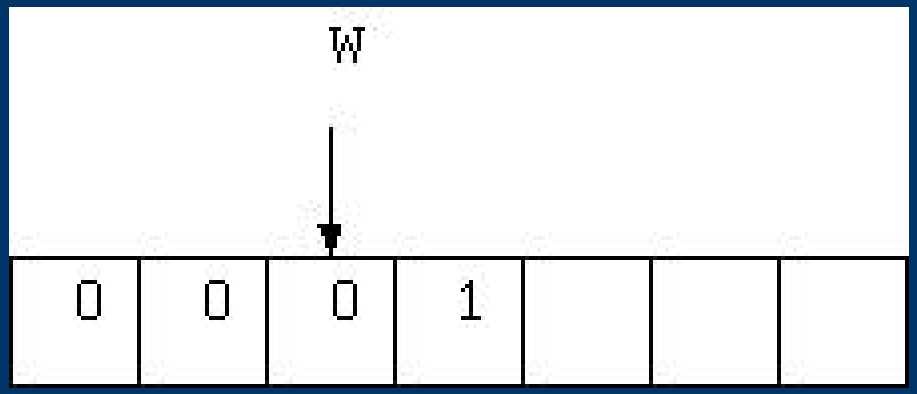
Adding 1 to 1011 with a Turing Machine



Step 2: we use the transition:

- $\delta(W,0) \rightarrow (H,1,R)$
- $\delta(W,1) \rightarrow (W,0,R)$ ◄
- $\delta(W,\square) \rightarrow (H,1,R)$

Adding 1 to 1011 with a Turing Machine

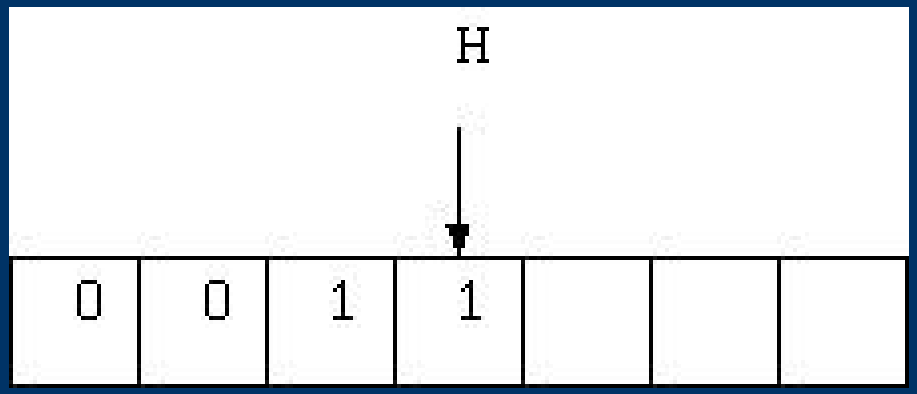


Step 3: we use the transition:

- $\delta(W,0) \rightarrow (H,1,R)$ ◀
- $\delta(W,1) \rightarrow (W,0,R)$
- $\delta(W,\square) \rightarrow (H,1,R)$

Adding 1 to 1011 with a Turing Machine

Final result:
1100



The Undecidability of the Safety Problem

The goal of this part of the talk is to show that determining safety of a general protection system is undecidable.

One way to prove undecidability is to show that, if there exists an algorithm for solving the problem at hand, we can solve the halting problem, which is known to be undecidable.

In order to do this, we must, given an arbitrary Turing Machine, show how we can emulate its computation with a protection system. But first...

The machine will never move off the left hand side.

- We mentioned above that we assume the transition function has been defined in such a way that the head never moves off the first recatngle to the left.

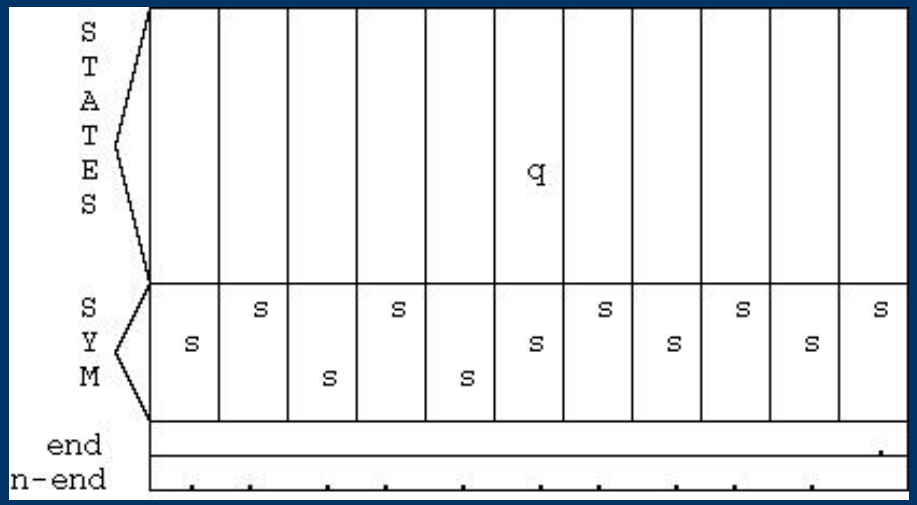
The machine will never move off the left hand side.

- In order to modify a machine where the head is not allowed to move off the left, to a machine whose transition function never moves off the left, we modify the original machine as follows:
  - Require that there be a special marker ⊩ as the leftmost symbol on the tape. The input is placed to its right.
  - Modify the transition function δ by defining
    $\delta(q, \Vdash) \rightarrow (q, \Vdash, R)$
    for all states q in Q. Thus, if the original transition function tries to move off to the left, this transition function moves it back again to the original first square.

Emulating a Turing Machine with a Protection System

- A row in the accessibility matrix (i. e. the set of objects a given subject can access) can be used to emulate a Turing Machine.
- For rights, we use one for each symbol in the Turing machine, plus one right each for each state in the Turing machine.
- We will add two special rights “at end” and “not at end” which will tell us whether the object in question is the rightmost.
- We will assume every command has an implicit parameter which is a subject S.
- We will number the objects from left to right and denote them by their number. The generic object will be denoted by n and represents the rectangle being worked on by the Turing Machine.

Emulating a Turing Machine with a Protection System (continued)



Subject S will have the following rights:

- Exactly one symbol right over each object;
- One state right over exactly one object (corresponding to the space being worked on by the Turing Machine).
- The state rights and the symbol rights are disjoint.
- The “at end” right over the rightmost object and
- the “not at end” right over all other objects.

Commands in the emulated Turing Machine

- Transitions of the form δ(q,s) → (q',s',L) get emulated by:

if q in a[S,n] and s in a[S,n] then
    delete q from a[S,n]
    delete s from a[S,n]
    enter s' into a[S,n]
    enter q' into a[S,n-1]

Commands in the emulated Turing Machine (continued)

- Transitions of the form $\delta(q,s) \rightarrow (q',s',R)$ need:

if q in a[S,n] and s in a[S,n] and not-at-end in a[S,n] then
  delete q from a[S,n]
  delete s from a[S,n]
  enter s' into a[S,n]
  enter q' into a[S,n+1]

Commands in the emulated Turing Machine (continued)

- Transitions of the form $\delta(q,s) \rightarrow (q',s',R)$ need:

if q in a[S,n] and s in a[S,n] and not-at-end in a[S,n] then
  delete q from a[S,n]
  delete s from a[S,n]
  enter s' into a[S,n]
  enter q' into a[S,n+1]

They also need:
if q in a[S,n] and s in a[S,n]
  and at-end in a[S,n] then
  delete q from a[S,n]
  delete s from a[S,n]
  delete at-end from a[S,n]
  enter not-at-end into a[S,n]
  enter s' into a[S,n]
  create object n+1
  enter at-end into a[S,n+1]
  enter q' into a[S,n+1]
  enter □ into a[S,n+1]

In summary

- We have shown how to take any Turing Machine and emulate its computation with the actions in a protection system.
- Since we cannot tell whether a Turing machine will halt, we cannot tell either whether a protection system will leak a right “H”.
- Therefore, the problem of telling whether a protection system “is safe” is undecidable.

In summary

- We have shown how to take any Turing Machine and emulate its computation with the actions in a protection system.
- Since we cannot tell whether a Turing machine will halt, we cannot tell either whether a protection system will leak a right “H”.
- Therefore, the problem of telling whether a protection system “is safe” is undecidable.
- But wait....

- We CAN tell when we are going to leak a given right.
- Why can't we just check?
- We can even generalize....

Policies

- Intuitively, a policy is a statement which states the operations that are and are not allowed.
- Formally,

  A policy is a recursive mapping P from the set of subjects, the set of objects and the set of rights into the boolean set {true, false}
- Given the above definition, we can define a protection system to be safe if:

  for all subjects s, all objects o and all rights r:

  r is in a[s,o] implies P(s,o,r)

Policies: An example

This definition is a generalization of the concept we used in the first part of this talk: we can define, for all objects o, P(S,o,H) is false, otherwise, for all subjects s≠S, all objects and all rights r, P(s,o,r) is true.

P-Protection Systems

Definition: A P-protection system consists of:

- A set of rights Ř.
- A set of potential objects Ŏ, a set of potential subjects Š and a policy P mapping
  Š X Ŏ X Ř → {true, false}
- Initial subsets O of Ŏ, S of Š and an initial accessibility matrix a[S,O]. As before, we will call configuration states to triples consisting of a subset of Š, a subset of Ŏ and an accessibility matrix. Thus the initial triple will be called an initial protection state.

Definition of P-Protection Systems (continued)

- A finite set of commands which consist of a name, invoked with a definite number of parameters which can be subjects or objects.
- An interpretation of these commands which are procedures built up from boolean predicates checking rights in either the accessibility matrix or the policy, and six elementary operations (in next slide)

Definition of P-Protection Systems (continued)

- The elementary operations are (where s denotes a subject, o denotes an object and r denotes a right):
  - If P(s,o,r) enter r into a[s,o]
  - Delete right r from a[s,o]
  - Create subject s (from Š) (no rights added to a)
  - Create object o (from Ŏ) (no rights added to a)
  - Destroy s
  - Destroy o

Definition:

Let (Ř,Š,Ŏ,P,S,O,a,C) be a P-protection system, (S',O',a') a protection state in the protection system. Then (S',O',a') will be called safe if, for all s in S', o in O' and right r in Ř, r in a'(s,o) implies P(s',o',r).

The main result of this talk

- Let (Ř,Š,Ŏ,P,S,O,a,C) be a P-protection system. Then, if (S,O,a) is safe, any configuration reached by any sequence of commands executed on the initial configuration will be safe.
- The proof follows by induction on the elementary commands executed:

Induction step of the proof:

- Only command adding a right.
  - If P(s,o,r) enter r into a[s,o]
    - Only adds right if allowed by the policy.
- The remaining commands do not add rights:
  - Delete right r from a[s,o]
  - Create subject s (from Š) (no rights added to a)
  - Create object o (from Ŏ) (no rights added to a)
  - Destroy s
  - Destroy o

Conclusions

- We have shown a stricter version of the Harrison-Ruzzo-Ullman theorem (1976)
- We have shown that the undecidability can be resolved by being careful: check against policy at every possible problematic step.

Directions for future work

- Main efforts for future work will be setting up criteria for policies which preserve the security triad of confidentiality, integrity and availability.

Directions for future work

- Main efforts for future work will be setting up criteria for policies which preserve the security triad of confidentiality, integrity and availability.
- Main emphasis, at present, is concentrating on information flow.

Information Flow

- Controlling Information Flow preserves confidentiality by controlling what subjects have access to information: information which should not be divulged is kept among subjects allowed to use it.
- Controlling information flow also preserves Integrity by guaranteeing that “tainted” information doesn't “contaminate” good information.

Questions?