# Obligations that Require and Affect Authorizations

William Winsborough

Department of Computer Science

University of Texas at San Antonio

Joint work with:

Keith Irwin

Ting Yu

Computer Science Department

North Carolina State University

# Obligations and Security

- Obligations are an essential part of security practice
  - Integrity, reliability, privacy, etc.
- Examples
  - System admin must remove accounts and physical access within 24 hours when an employee leaves the company
  - Private data must be deleted after one year
  - Help desk must respond to tickets within 4 hours
  - Dr. Alice must finalize her analysis of the lab reports within one week after receiving them
  - If you check code out, you have to check it back in
  - If you submit a purchase order, your manager has to approve or deny it

# Increasing Importance

- Policy-based automated systems are managing more interactions and workflows
  - Many involve not just access control, but also obligations
- New laws increase obligation requirements
  - HIPAA
  - Sarbanes-Oxley
- Security languages increasingly model obligations
  - Ponder, Rei

# What is an Obligation?

- User action to be performed in future
  - As opposed to system action (XACML, KAoS)
  - System cannot ensure obligation is fulfilled
  - Fulfillment can be monitored
- Time limit
  - Deadlines make the world go round
  - (*user*, *action*, [$t_{start}$, $t_{end}$])

# Some of the Prior Work

- System obligations: XACML, KAoS
- User obligations: Ponder, SPL, Rei
- Obligations incurred via access control: Bettini
- Monitoring: Heimdall
  - Deadlines: xSPL

# Obligations in Security Policies

- Obligations arise from user actions
  - To take action A, you must agree to perform action B later
    - File a travel report
  - When you take action A, someone else has to do action B later
    - Help desk ticket, bug report
- Could also be triggered by other events

# Distinguishing our Work

- Interaction between access control and obligations
  - Obligatory actions are subject to access control
  - Obligatory actions affect authorization state
  - This creates dependencies between obligations
- New access-control requirements
  - Prevent assigning obligations that
    - Perform unauthorized actions
    - Change authorization state in a way that interferes with other obligations
  - Prevent performing any action that interferes with existing obligations

# Policy Model

- Policy statements associate actions with access control conditions and the obligations the actions produce

  – action(subject, objects) ← condition : {obligations}

# Example: Research Organization Library

- Items must be returned
  - CheckOut(u,b) ← Employee(u) ∧ Book(b) : {(u, CheckIn(b), [today, today + 1 month])}
  - CheckOut(u,d) ← Employee(u) ∧ DVD(d) : {(u, CheckIn(d), [today, today + 1 week])}
- Video camera borrowers must log footage
  - ReturnCamera(u, x, y) ← TakenWith(x, y) : {(u, LogFootage(y), [today, today + 1 week])}

# Example: Research Organization Library

- Magazine requests must be reviewed promptly
  - Request(u,x) ← Magazine(x) : {(Librarian, ApproveOrDeny(x), [today, end of next week])}

# Goals for Obligations

- System cannot guarantee that all obligations are fulfilled
- Want to ensure that all obligations are authorized
- Obligations are like a contract
  - User agrees to take an action
  - System agrees to ensure action is authorized, provided other users are diligent

# Talk Outline

- Accountability: ensuring diligent users have the rights they need to perform their obligations
  - – [Irwin, Yu, Winsborough, ACM CCS 2006]
  - – Full article submitted for publication
- Assigning responsibility when obligations are unfulfilled
  - – [Irwin, Yu, Winsborough, IFIPTM 2008]

# Reference Monitor

- If a user requests an action that assigns an obligation, will the obligatory action be authorized?
- Two easy answers
  - Definitely!
    - Grant the request
  - Absolutely Not!
    - Deny the request
- But what about the rest of the time?

# Option 1: Be Pessimistic

- If there is any possibility that the obligatory action will be unauthorized, deny the request
- But …
  - What if the possibility is really remote?
  - What if there’s a super user who can take away privileges?

# Option 2: Be Optimistic

- If there is any possibility that the obligatory action will be authorized, grant the request
- But …
  - What if the possibility is really remote?
  - What if there’s a super user who can grant permissions?

# A More Satisfactory Approach

- Assume that the other obligations in the system will be fulfilled
  - – If they are not, it is not the system’s fault
- Assume that only obligatory actions will occur
  - – When other actions are requested, analyze their impact at that time

# Accountability

- A property of states
  - State = authorization state + set of pending obligations
  - Property intuition: if all users are diligent, then all obligations will be fulfilled
- When accountability is maintained, obligatory actions are always authorized at the appropriate times

# Maintaining Accountability

- When deciding whether to grant an action request, determine whether
  - Obligations it introduces will be authorized
  - The action itself interferes with existing obligations
  - The obligations in introduces interfere with existing obligations

# Metamodel (Fully Abstract)

- Subjects, Objects, Actions
- Policy: set of policy rules
  - action ← condition : obligation set
- System State:
  - Current subject and object sets
  - Time
  - Set of Obligations
  - Abstract component of state
    - Models authorization state and effects of actions

# Example: Research Organization Library

- Subjects: employees, librarians...
- Objects: books, DVDs, cameras...
- Actions: checkout, reserve...
- Policy: rules from previous slides
- State:
  - Eligible borrowers, collection contents
  - Time, pending obligations
  - Application-specific state: What is checked out, who has it, patron authorizations, etc.

# Strong Accountability vs. Weak Accountability

- Strong accountability
  - At any point within an obligation’s time interval, the obligatory action is authorized
- Weak accountability
  - By the end of an obligation’s time interval, the obligatory action is authorized
- Useful in different situations

# The Accountability Problem

- Given a policy and a system state, is the state strongly (resp., weakly) accountable
  - Will all obligations be authorized if all users are diligent
- Undecidable in the fully abstract case outlined above
  - Fully abstract state can represent Turing machine configuration
  - Obligatory action performs one step and generates an obligation to perform the next
- Polynomial time for some important specific cases

# An Abstract Subproblem That is Polynomial

- Monotonicity
  - Once action is authorized, it remains authorized
- No cascading obligations
  - Actions are partitioned:
    - Those that can impose obligations
    - Those that can be obligations
- Commutative, time-independent actions
  - If two actions are authorized, the result of performing both of them is independent of the order in which they are performed
  - The authorization of an action is independent of the time at which it is requested

# This Abstract Subproblem is Polynomial

- Results:
  - O(nm) algorithm for Weak Accountability
  - O($n^2$m) algorithm for Strong Accountability
    - n is the number of pending obligations
    - m is the number of rules in the policy
  - If only two of the three assumptions holds, the problem is co-NP Hard

# A Concrete Model That is Poly-time: Access Matrix

- Model authorization state as access matrix
  - A set of triples of (*subject, object, permission*)
- Action conditions are a logical combination of positive and negative tests for permissions
- Action effects are the granting or revoking of permissions
  - Deleting a subject or an object is treated like revoking all their permissions at once

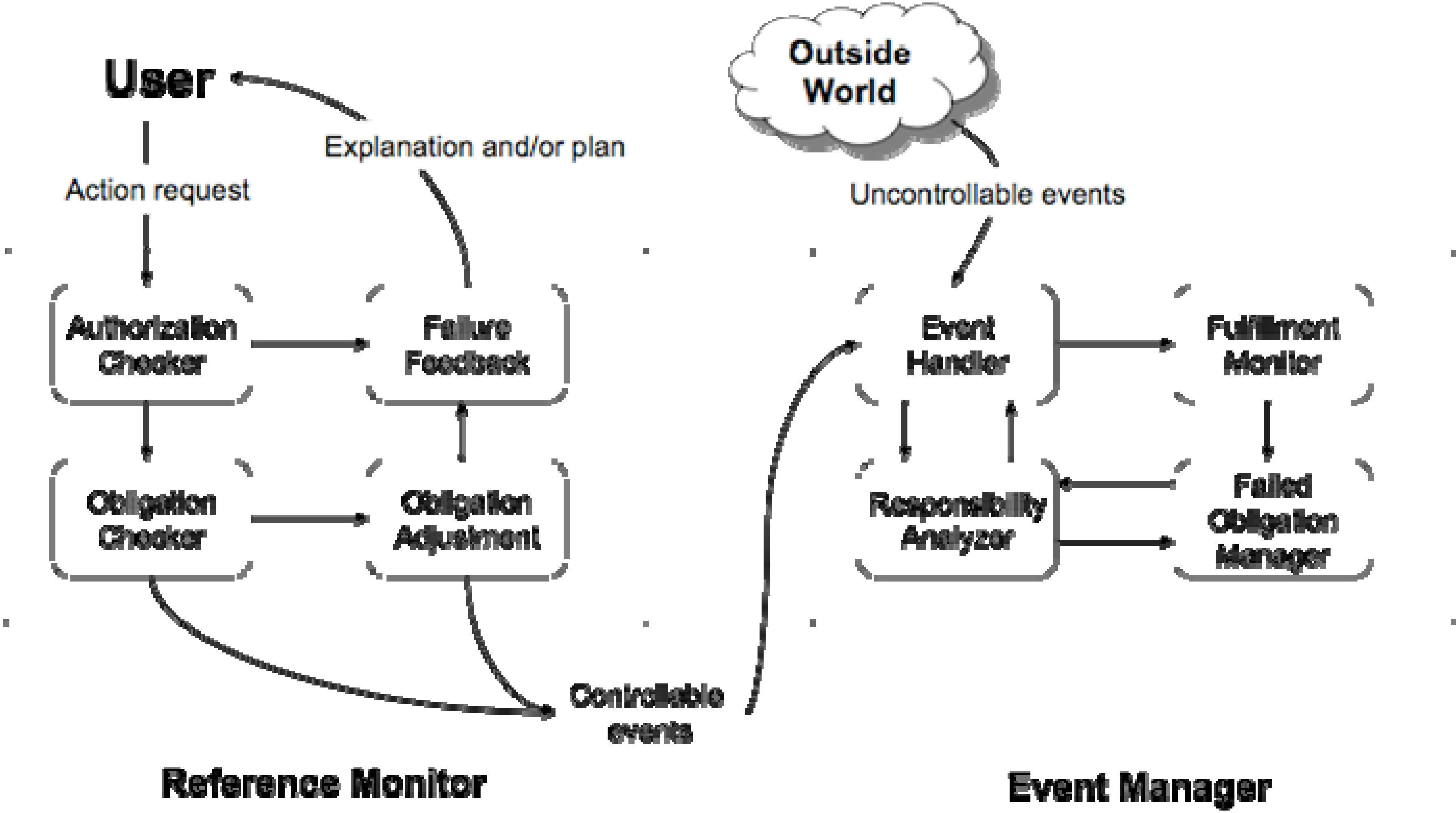
# Poly-time Concrete Model

- No cascading obligations
- Conditions are expressed in CNF
- Results:
  - $O(n^4m^2)$ algorithm for Strong Accountability
    - n is the number of pending obligations
    - m is the size of the policy
  - Weak Accountability is co-NP Hard

# Corrections to CCS paper

- Access-matrix Strong Accountability algorithm in the paper is not correct
  - Sound, but not complete
  - See tech report for correct algorithm
- Access-matrix Weak Accountability algorithm promised to be in the tech report does not exist
  - We had a sound algorithm, which we did not realize was not complete
  - Completeness co-NP Hard
  - See tech report for a reduction

# User Obligation Management System



# Talk Outline

- Accountability: ensuring diligent users have the rights they need to perform their obligations
  - – [Irwin, Yu, Winsborough, ACM CCS 2006]
  - – Full article submitted for publication
- Assigning responsibility when obligations are unfulfilled
  - – [Irwin, Yu, Winsborough, IFIPTM 2008]

# Why Fault Assessment is Needed

- Obligation systems should strive to maintain accountability
  - Depends on users fulfilling their obligations if they can
- When they do not or cannot not, it is important to know who dropped the ball
  - Contractual obligations, reputation, trust
- In an accountable system, any single failure is the fault of the user who did not fulfill their obligation
- When multiple failures have occurred, the assignment of fault is more complex

# Why Fault Assessment is Tricky

- User obligations can have complex interdependencies
- One possible approach is analyzing dependencies after the fact
  - Assigning fault on this basis is difficult to do appropriately
- Appropriate assignment of fault relies on a notion of responsibility
  - This often depends on exogenous factors, such as agreements, job roles, and the order in which obligations were incurred

# Example

- Carol, a manager, needs to submit a situation report to her boss
- Alice and Bob both work for Carol
  - Both are assigned an obligation to prepare a situation report
  - Neither does so, so Carol cannot submit
- Who is at fault?

# Example

- While Alice and Bob might be equally at fault, this is not a satisfactory conclusion in general
  - Dubious managerial practice
  - Does not scale when there are cascading failures
- Perhaps Alice’s primary responsibility is to prepare situation reports
  - Bob was supposed to do one only as a training exercise
- Perhaps Alice had a family emergency
  - Bob assured her that he would prepare the report
- None of this can be reasoned about simply by examining dependencies among obligations

# Appropriate Fault Assessment

- Ideally, when an obligation b is violated, we would like to identify a set of prior obligations that were authorized but not fulfilled and that, had they been fulfilled, would have enabled b to be fulfilled
- We would like the set to be minimal
- When either of two or more obligations could be included in the set, we seek a basis for selecting which to include

# Our Approach

- We propose on-line “responsibility” tracking
  - Subset of the dependence relation
- Use a policy to determine which obligations are considered responsible for enabling which others as the system state evolves
  - Example policy: the first obligation that was introduced and ensures a given obligation is authorized is responsible for it
- This allows for accurate fault assessment when an obligation is violated
  - Note: responsibility does not entail fault
- Helps users understand the consequences of their actions and inactions

# Bounds of Responsibility

- Users whose obligations are strictly need for a given obligation to be fulfilled should be held responsible for it
- Users whose obligations cannot affect a given obligation should not be held responsible for it
- There are many different ways to assign responsibility within those bounds

# Concrete Failure Assessment Algorithm

- In the paper, we show how to construct the responsibility graph incrementally as the system state evolves
- This is done for an access-matrix-based system

# Ongoing Work

- Currently looking at planning techniques to be used for two purposes:
  - Restoring accountability after an obligation is unfulfilled
  - When a desired action would make the system unaccountable, what compensating actions would allow the action
- Techniques under investigation:
  - AI planners
  - Model checking

# Future Work

- Extending our techniques to
  - Support events that cannot be prevented by the system
  - Ensure obligatory actions have needed resources as well as authorizations

# Other Recent Work

- Alternate model in which authorizations are granted based on assigned tasks (cf. obligations)
  - [Irwin, Yu, Winsborough, SACMAT 08]
  - We prevent insecure combinations (and sequences) of privileges and actions in this model
    - Static analysis
    - Dynamic control of privileges
    - Dynamic control of actions

# Conclusions

- Accountability: ensuring diligent users have the rights they need to perform their obligations
  - – [Irwin, Yu, Winsborough, ACM CCS 2006]
  - – Full article submitted for publication
- Assigning responsibility when obligations are unfulfilled
  - – [Irwin, Yu, Winsborough, IFIPTM 2008]

# Thank You

- Questions?