Distributed Proving in Access-control Systems

Lujo Bauer, Scott Garriss, Michael K. Reiter
CMU

Presented by Andrew K. Adams
Motivation

• Authors noticed that all prior work used, what they call “eager” strategy, i.e., the prover does all the work (occasionally requesting credentials from others) ...

• ... burden of work is on the prover ...

• ... additionally, in a system where user discretion is involved, a certificate provider may prefer to issue a less sensitive (or less powerful) cert!
Question Hints

• **Hint 1**: Detection & Recourse
• **Hint 2**: Complete Mediation
• **Hint 3**: Akamai
Outline

• Quickly talk about logic and rules.
  • ... just a quick recap (for those that need it)!

• Proving strategies.

• Proving algorithm.

• Experiment:
  • what they did
  • what we learned
Access-control Logic

• Logic inhabited by terms and formulas
  
  • $t ::= s|p$
  • $p ::= \text{key}(s) \mid p.s$
    • $p.s$ referred to as *local namespace*

• Formulas describe principals’ beliefs
  
  • $A \text{ says } F$
  • pubkey $\text{signed } F$
  • $\text{action}(\text{resource, nonce})$
  • $A \text{ speaksfor } B$
  • $\text{delegate}(A, B, \text{resource})$

• Formula syntax:
  
  \[
  \phi ::= s \text{ signed } \phi \mid p \text{ says } \phi \\
  \phi ::= \text{action } (s, s) \mid p \text{ speaksfor } p \mid \text{delegate}(p, p, s)
  \]
Inference Rules

• Used as tactics in the Tactical Theorem Prover ...
Proving Strategies

• Eager Strategy
  
  • prover attempts to decompose proof into *only* subgoal formulas of the type $A \text{ signed } F$
  
  • these represent credentials to be downloaded
  
  • prover needs to basically guess what certificates remote parties will contribute to proof
    
    • which, of course, the remote party may not want to do
Proving Strategies, Cont.

- Lazy Strategy
  - A **signed** F implies A **says** F
  - so go ask Bob to *prove* Bob **says** F
  - Alice gets the subproof or failure in a single request
  - Bob gets to choose what certificates to use in proof, i.e., what amount of authority he will give to Alice
Proving Strategies, Cont.

- Assuming a well-written network protocol to exchange proof messages (and of course, a usable PKI), can we stop or even detect Bob from misbehaving? That is, in the paper’s example, if Bob the floor manager *could* satisfy a proof, but lies via a returned “failure” ... what’s my recourse?
Proving Strategies, Cont.

- Assuming a well-written network protocol to exchange proof messages (and of course, a usable PKI), can we stop or even detect Bob from misbehaving? That is, in the paper’s example, if Bob the floor manager *could* satisfy a proof, but lies via a returned “failure” ... what’s my recourse?

- We have a record, report to DH? (Would have to get suspicious over time?)

- Use previous ATGs in some way (although I haven’t talked about ATGs yet :-((( }
Proving Algorithm

0  global set KB
1  substitution bc-ask(                          /* returns a substitution */
    list goals,                                    /* list of conjuncts forming a query */
    substitution θ,                                /* current substitution, initially empty */
    set failures)                                  /* set of substitutions that are known */
                                           /* not to produce a complete solution */
2  local substitution answer                      /* a substitution that solves all goals */
3  local set failures'                             /* local copy of failures */
4  local formula q'                                /* result of applying θ to first goal */
5  if(goals = [] ∧ θ ∈ failures) then return ⊥    /* θ known not to produce global solution */
6  if(goals = []) then return θ                   /* base case, solution has been found */
7  q' ← subst(θ, first(goals))                   /* prove first goal locally or remotely? */
8  l ← determine-location(q')                     /* prove first goal locally or remotely? */
9  failures' ← failures                          /* prove first goal locally or remotely? */
10 if (l ≠ localmachine)                            /* make remote request */
   while ((α ← rpc(bc-ask(first(goals), θ, failures'))) ≠ ⊥)
   failures' ← α ∪ failures'                      /* prevent α from being returned again */
11   answer ← bc-ask(rest(goals), α, failures)    /* prove remainder of goals */
12   if (answer ≠ ⊥) then return answer           /* if answer found, return it */
13 else foreach (P, q) ∈ KB                       /* investigate each tactic */
14   if ((θ' ← unify(q, q')) ≠ ⊥)                 /* determine if tactic matches first goal */
15     while ((β ← bc-ask(P, compose(θ', θ), failures)) ≠ ⊥)
16       failures ← β ∪ failures                  /* prevent β from being returned again */
17     answer ← bc-ask(rest(goals), β, failures)  /* prove remainder of goals */
18     if (answer = ⊥) then return answer         /* if answer found, return it */
19 else return ⊥                                /* if no proof found, return failure */
Implementation

• Model physical world door access on a floor of a building.

• Certificate hierarchy composed of department heads, floor managers and users.
  • CMU says action(room15)

• If UserC wants access to resource...
  • ... must prove: key(K_CMU) says action(resource, nonce)
Proof for UserC

• Credentials needed by UserC to satisfy proof:

P1 = $K_{CMU}$ signed (key($K_{CMUS}$) speaksfor key($K_{CMU}$))
P2 = $K_{CMU}$ signed (key($K_{CMUCA}$) speaksfor key($K_{CMU}.CA$))
P3 = $K_{CMUCA}$ signed (key($K_{UserA}$) speaksfor key($K_{CMU}.CA.UserA$))
P4 = $K_{CMUCA}$ signed (key($K_{UserB}$) speaksfor key($K_{CMU}.CA.UserB$))
P5 = $K_{CMUCA}$ signed (key($K_{UserC}$) speaksfor key($K_{CMU}.CA.UserC$))
P6 = $K_{CMUS}$ signed (delegate(key($K_{CMU}$), key($K_{CMU}$).$DH1$, resource))
P7 = $K_{CMUS}$ signed (key($K_{CMU}$).CA.UserA speaksfor key($K_{CMU}$.$DH1$))
P8 = $K_{UserA}$ signed (delegate(key($K_{CMU}$).$DH1$, key($K_{CMU}$).$DH1$.FM1, resource))
P9 = $K_{UserA}$ signed (key($K_{CMU}$).CA.UserB speaksfor key($K_{CMU}$.$DH1$.FM1))
P10 = $K_{UserB}$ signed (delegate(key($K_{CMU}$).$DH1$.FM1, key($K_{CMU}$).CA.UserC, resource))
P11 = $K_{UserC}$ signed (action(resource, nonce))
Tactics for Proof

0 key(KCMU) says (key(KCMUS) speaksfor key(KCMU)) SAYS-I(P1)
1 key(KCMU) says (key(KCMUCA) speaksfor key(KCMU).CA) SAYS-I(P2)

2 key(KCMUCA) says (key(KUserA) speaksfor key(KCMU).CA.UserA) SAYS-I(P3)
3 key(KCMUCA) says (key(KUserB) speaksfor key(KCMU).CA.UserB) SAYS-I(P4)
4 key(KCMUCA) says (key(KUserC) speaksfor key(KCMU).CA.UserC) SAYS-I(P5)

5 key(KCMU).CA says (key(KUserA) speaksfor key(KCMU).CA.UserA) SPEAKSFOR-E2(1, 2)
6 key(KCMU).CA says (key(KUserB) speaksfor key(KCMU).CA.UserB) SPEAKSFOR-E2(1, 3)
7 key(KCMU).CA says (key(KUserC) speaksfor key(KCMU).CA.UserC) SPEAKSFOR-E2(1, 4)

8 key(KCMUS) says (key(KCMU).CA.UserA speaksfor key(KCMU).DH1) SAYS-I(P7)
9 key(KCMU) says (key(KCMU).CA.UserA speaksfor key(KCMU).DH1) SPEAKSFOR-E(0, 8)

10 key(KUserA) says (key(KCMU).CA.UserB speaksfor key(KCMU).DH1.FM1) SAYS-I(P9)
11 key(KCMU).CA.UserA says (key(KCMU).CA.UserB speaksfor key(KCMU).DH1.FM1) SPEAKSFOR-E2(5, 10)
12 key(KCMU).DH1 says (key(KCMU).CA.UserB speaksfor key(KCMU).DH1.FM1) SPEAKSFOR-E2(9, 11)

13 key(KCMUS) says delegate(key(KCMU), key(KCMU).DH1, resource) SAYS-I(P6)
14 key(KCMU) says delegate(key(KCMU), key(KCMU).DH1, resource) SPEAKSFOR-E(0, 13)

15 key(KUserA) says delegate(key(KCMU).DH1, key(KCMU).DH1.FM1, resource) SAYS-I(P8)
16 key(KCMU).CA.UserA says delegate(key(KCMU).DH1, key(KCMU).DH1.FM1, resource) SPEAKSFOR-E2(5, 15)
17 key(KCMU).DH1 says delegate(key(KCMU).DH1, key(KCMU).DH1.FM1, resource) SPEAKSFOR-E2(9, 16)

18 key(KUserB) says delegate(key(KCMU).DH1.FM1, key(KCMU).CA.UserC, resource) SAYS-I(P10)
19 key(KCMU).CA.UserB says delegate(key(KCMU).DH1.FM1, key(KCMU).CA.UserC, resource) SPEAKSFOR-E2(6, 18)
20 key(KCMU).DH1.FM1 says delegate(key(KCMU).DH1.FM1, key(KCMU).CA.UserC, resource) SPEAKSFOR-E2(12, 19)

21 key(KUserC) says action(resource, nonce) SAYS-I(P11)
22 key(KCMU).CA.UserC says action(resource, nonce) SPEAKSFOR-E2(7, 21)
23 key(KCMU).DH1.FM1 says action(resource, nonce) DELEGATE-E(20, 22)
24 key(KCMU).DH1 says action(resource, nonce) DELEGATE-E(17, 23)
25 key(KCMU) says action(resource, nonce) DELEGATE-E(14, 24)
Caching

• Initial implementation had 1000s of credential requests per access ... not good!
  • so authors went with caching

• First attempt, cache positive results ...
  • ... but no visible difference in performance!

• Second attempt, cache *failed results* (what they call negative caching) ...
  • ... finally, an improvement in lazy strategy!

• Automatic Tactic Generation
  • use unbounded vars in creds, wow, lazy 3x better
Caching, Cont.

- Uh, but doesn’t caching break the security principal *Complete Mediation*?
Caching, Cont.

- Uh, but doesn’t caching break the security principal *Complete Mediation*?
- Security *is* a trade-off ... if the system doesn’t get used, then who cares if it’s secure.
Caching, Cont.

• Uh, but doesn’t caching break the security principal *Complete Mediation*?

• Security *is* a trade-off ... if the system doesn’t get used, then who cares if it’s secure.

• Okay, if we accept that caching is an acceptable trade-off, will systems like Akamai (to cache ATGs or other certs) help with distributed proving?
Cons

• Authors way too eager to use “eager”.

• Although their *distributed* system model used a j,k,l tree, in reality there was only *one* parent CA ... that’s not what I’d call distributed.

• It would have been nice to see the requests generated by each subgoal in the example proof!
Pros

• Key points:
  • Eager vs. Lazy (distributed proving)
  • Caching strategies critical

• Maybe this was blatantly obvious, but there’s still hope for interactive authorization, providing ...
  • ... the reference monitor is on a cell phone/PDA ...
  • ... and the domain doesn’t require > 10/ day
Questions?