EXPERIENCES IN THE FORMAL ANALYSIS OF THE GDOI PROTOCOL

Catherine Meadows
Code 5543
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil
http://chacs.nrl.navy.mil

MOTIVATION AND BACKGROUND

• Project started in 1999
• At that time, had long history of formal analysis of crypto protocols (about 20 years, starting with Dolev and Yao work)
• Applied to lots of different types of problems
• Has had some real success
  – Found previously undiscovered problems
  – But (as of 1999) – lack of impact on "real life" protocols
• Few examples to point to of formal analysis affecting fielded product
• WHY?
  – In this project, attempted to address this problem

OUR PLAN

• Work closely with standards developers as they draft standard
  – Give feedback as early in the standardization process as possible
• Discuss any problems we found as they arose
  – Allowed us to identify quickly which were real problems and which arose from misunderstanding of protocol
• Recommend fixes when appropriate

GROUP WE WORKED WITH

• Internet Engineering Task Force (IETF)
  – Mostly volunteer standards group responsible for internet protocol standards
  – Made up of different working groups concentrating on standards for different protocols
• Internet Research Task Force (IRTF)
  – Research group attached to IETF
  – Works on focussed research problems of interest to IETF
• Secure Multicast Working Group (SMuG) in IRTF
  – Devoted to protocols associated with secure multicast

WHAT I’LL TALK ABOUT TODAY

• How we worked with SMuG
• Protocol we worked on, GDOI
• A little background of formal methods for crypto protocol analysis
• Tool we used, NRL Protocol Analyzer
• Technical challenges we faced
• The outcome so far
• A coda

HOW WE WORKED WITH SMUG

• Attended SMuG meetings regularly
  – Helped to
    • Get to know SMuG members
    • Learn about background of SMuG protocols
    • Inform SMuG members of our own requirements
• Early on, picked Group Domain of Interpretation (GDOI) protocol as a good candidate
• Used GDOI drafts as basis for formal specifications as they came out
• When found problems or ambiguities, would discuss them with authors
  – Would often lead to new GDOI drafts
MULTICAST ARCHITECTURE USED BY GDOI

SA = “security association”
SA1 = pairwise key
SA2 = key encryption key (can be key hierarchy, used for access control)
SA3 = traffic encryption key

GDOI PROTOCOLS

Groupkey Pull Protocol
Initiator (Member) Responder (GCKS)
------------------ ----------------
| HDR*, HASH(*) | Hib | M-ID | Ni | ID | KE_I | CERT | POP_I |
| HDR*, HASH(*) | Hib | M-ID | Ni_b | Nr | SA | KE_R | CERT | POP_R |

Hashes are computed as follows:

HASH(1) = prf(SKEYID_a, M-ID | Ni | ID)
HASH(2) = prf(SKEYID_a, M-ID | Ni_b | Nr | SA)
HASH(3) = prf(SKEYID_a, M-ID | Ni_b | Nr_b | KE_I | CERT | POP_I)
HASH(4) = prf(SKEYID_a, M-ID | Ni_b | Nr_b | KE_R | SEQ | KD | CERT | POP_R)

Groupkey Push Message
Member GCKS or Delegate
------------------ ----------------
| HDR*, SEQ | SA | KD | CERT | SIG |

Key hierarchies can be used to prevent expelled member from learning new key-encryption keys

Initially, each user gets all keys in its path to K
- When u1 leaves, GCKS computes new k12', k14', K'
- U2 gets k2[k12'], k12'[k14'], k14'[K']
- U3 gets k34[k14'], k14'[K']

GDOI does not specify key hierarchies but is compatible with them

GDOI

- Protocol facilitating distribution of group keys by Group Key Distribution Center (GCKS)
  - Embodies SMuG framework and architecture
  - Based on ISAKMP and IKE
  - Standards developed for key exchange
- Protocol uses
  - IKE to distribute Category-1 SAs (pairwise keys)
  - Groupkey Pull Protocol initiated by member to distribute Category-2 SAs (KEKs)
    - May also distribute Category-3 SAs (TEKs)
  - Groupkey push Datagram to distribute Category-2 and Category-3 SAs

KEY HIERARCHIES FOR ACCESS CONTROL

- Key hierarchies can be used to prevent expelled member from learning new key-encryption keys

THE NRL PROTOCOL ANALYZER

- Formal methods tool for verifying security properties of crypto protocols and finding attacks
- User specifies protocol in terms of communicating state machines communicating by use of a medium controlled by a hostile intruder
- User verifies protocol by
  1. Proving a set of lemmas to limit size of search space
  2. Specifying an insecure state
  3. Using NPA to search backwards from that state to see if attack can be found

NRL Protocol Analyzer Model

- Honest Principals modeled as communicating state machines
- Dolev-Yao Adversary
- Dishonest principals part of the adversary
- Each run of a protocol local to a principal assigned a unique round number
  - Allows distinguishing of different runs local to a principal
NPA Events

- Each state transition in an NPA spec may be assigned an event, denoted by $\text{event}(P, Q, T, L, N)$
  - $P$: principal doing the transition
  - $Q$: set of other parties involved in transition
  - $T$: name of the transition rule
  - $L$: set of words relevant to transition
  - $N$: local round number
- Events are the building blocks of the NPATRL Language

NPATRL

- NRL-Protocol-Analyzer-Temporal-Requirements-Language
  - Pronounced 'N Patrol'
- Requirements characterized in terms of event statements
  - $\text{learn}$ events indicate acquisition of information by adversary
  - Syntax closely corresponds to NPA language, e.g.,
    \[ \text{receive}(A, B, \{\text{message}\}, N) \]
- Add usual logical connectives, e.g., $\neg\neg\neg\neg$, $\land\land\land\land$, $\Rightarrow\Rightarrow\Rightarrow\Rightarrow$
- One temporal operator meaning "happens before" $\diamond$

Example NPATRL Requirement

- If an honest $A$ accepts a key $\text{Key}$ for communicating with an honest $B$, then a server must have generated and sent the key for an honest $A$ and an honest $B$ to use.

\[
\text{accept}(\text{user}(A, \text{honest}), \text{user}(B, X), \{\text{Key}\}, N') \Rightarrow \\
\Diamond \text{send}(\text{server}, \text{user}(A, \text{honest}), \text{user}(B, \text{honest}), \{\text{Key}\}, N)
\]

THREE TYPES OF REQUIREMENTS

- Secrecy requirements
  - Intruder should not learn secrets, except under certain failure conditions
- Authentication requirements
  - If $A$ accepts a message as coming from $B$ intended for purpose $X$, then $B$ should have sent that message to $A$ and intended it for purpose $X$
- Freshness requirements
  - Conditions on recency and/or uniqueness of accepted messages
  - Some models bundle freshness and authentication together

Analysis Using NPA/NPATRL

- Map event statements to events in an NRL Protocol Analyzer specification
  - Interpret atomic formulae
- Take negation of each NPATRL requirement
  - Defines a state that should be unreachable iff requirement is satisfied
- Use NPA to prove goal is unreachable, or
  - Use NPA to reach goal, i.e., find attack

Existing NPATRL Requirements Suites

- Requirements have been given for
  - Two party key distribution protocols
  - Two party key agreement protocols
  - Credit card payment transactions
  - SET (Secure Electronic Transactions)
NPA SPEC OF GDOI
- Protocol starts with GCKS creating a group and a group key
- At any time after that, a group member may request to join the group by initiating a Groupkey Pull Exchange
  - GCKS responds by completing protocol
- At any time after that any of the below may occur
  - GCKS may expel member and refuse to send it new keys
  - Group member may initiate new Phase 2 exchange
  - GCKS may send keys to group member using Groupkey Push Datagram
- Initial spec took a little under a week to write

STRUCTURE OF SPECIFICATION

HOW SPECIFICATION LIMITED
- NPA can't currently handle unbounded data structures such as key hierarchies
  - Can specify them, but they will send NPA into infinite loop
  - Currently investigating appropriate abstractions
- So --
  - For the moment did not try to specify key hierarchies, assumed each KEK is a single key
  - Assumed that in Phase 2 Exchange, one SAK sent
  - Assumed three possibilities for Groupkey Push Datagram
    - One SAK or one SAT
- Also, did not include spec of IKE Phase 1

Challenges In Developing Requirements for Group Protocols
- In pairwise protocols, have notion of a session
  - Secrecy means keys not learned by parties not involved in the session
  - Freshness means key is unique to a session
- In group protocol session much more open ended
  - Many keys may be distributed in one session
  - Principals may join and leave the group during a session
    - How should their access to keys be limited?
    - How do different secrecy requirements interact with each other?

FRESHNESS ISSUES
- Like secrecy, freshness is more complicated for group protocols
  - Can no longer tie key to session
- For GDOI, identified two types of freshness
  - Recency Freshness
    - KEK generated most recently (or after a specific time) is the current one
  - Sequential Freshness
    - Principal should never accept KEK that is less recent than the one it has
- For Groupkey push datagram, can only ensure that key principal accepts is most recent known to it, not that it is current
**RECENCY FRESHNESS FOR PULL PROTOCOL**

\[
\text{member_acceptpullkey}(N,\text{GCKS},(G,K,PK),N) \implies \text{stealpairwisekey}(\text{env},(),\text{GCKS},M,PK),(N) \lor \\
\text{not}(\text{member_requestkey}(M,\text{GCKS},\text{Nonce},PK),N) \land \\
\text{gcks_expire}(\text{GCKS},(G,K),N) 
\]

If member accepts key K via a pull protocol, then either
1. his pairwise key was stolen, or
2. K should not have expired previously to the request can’t require that key be current at time of receipt, could have expired en route

**SEQUENTIAL FRESHNESS FOR PULL PROTOCOL**

\[
\text{member_acceptpullkey}(M,\text{GCKS},(G,K,PK),N) \implies \\
\text{stealpairwisekey}(\text{env},(),\text{GCKS},M,PK),(N) \lor \\
\text{not}(\text{member_requestkey}(M,\text{GCKS},\text{Nonce},PK),N) \land \\
\text{gcks_makecurrent}(\text{GCKS},(G,K),\text{(G.K},N)) 
\]

If member accepts a key K, then either
1. his pairwise key was stolen, or
2. he should not have previously accepted a key that became current later than K

**SECRECY REQUIREMENTS FOR GDOI**

- **Forward access control**
  - Principals should not learn keys distributed after they leave the group
- **Backward access control**
  - Principals should not learn keys that expired before they joined the group
- **Perfect forward secrecy**
  - If pairwise key stolen, only keys distributed with that key after the event should be compromised
- Other requirements may govern effects of stealing key encryption keys, etc.
- How do these interact with each other?

**SOLUTION: DEVELOP CALCULUS OF SECRECY REQUIREMENTS**

- Build collection of NPATRL statements of events that can lead to key compromise
  - Currently restricted to requirements for keks
  - Five non-recursive base cases describing
    - Stealing of pairwise and group keys
    - Group keys sent to dishonest members
  - Two recursively defined cases addressing generalizations of forward and backward access control
- Mix and match statements to get requirement of your choice

**AN UNEXPECTED DEVELOPMENT**

- All requirements could easily be expressed in terms of fault trees
  - Described sequences of events that should or should not lead up to event such as accepting a key, learning a key, etc.
  - Can reason about sequences that
    - Should both happen (AND)
    - One of which should happened (OR)
    - Should not happen (NOT)

Fig. 4 Forward Access Control Without PFS or Backward Access Control
SOME RESULTS OF SPECIFYING PROTOCOL

- Identified several omissions and ambiguities
- Found one major inconsistency
  - Sequence numbers were originally send in KD payload
  - Sequence numbers updated every time new KEK created
  - Didn’t account for fact that some push messages may not contain KEK’s
- Now sequence numbers updated every time new push message sent

SOME RESULTS OF SPECIFYING REQUIREMENTS

- Improvement to Proof-of-possession option
  - In old version, principals only signed own nonces
  - Didn’t work if pairwise keys compromised
  - Now, principals sign hash of both nonces
- Found detail that needed to be added to Groupkey Pull protocol
  - Did not satisfy sequential freshness unless require that member checks that SEQ number received in last message was greater than SEQ number it may currently hold

RESULTS OF ANALYSIS

- Two similar oracle attacks making use of type confusion
  - One found using NPA
  - Another (simpler) one found after NPA found first attack
    - Suggested by NPA result
- Will present simpler attack here
- Suppose dishonest group member wants to trick other group members into accepting a fake key as a genuine one
- Suppose that protocol uses Proof-of-Possession option
- Then ...

RESULTS

- Identified potential GDOI problems early on, resulting in a better protocol
- Formal analysis credited with speeding up acceptance of GDOI and of the new MSeC (multicast security) working group formed out of SMuG
- Starting to see interest from other parts of IETF in performing or applying formal analyses
- Some avenues for further research
  - Fault tree representation of requirements
  - Algorithms for detecting type confusion/oracle attacks

FIX TO PROTOCOL

- First, did quick analysis to see if attack was really possible
  - What kind of assumptions about lengths of data did it require?
- Whenever signature taken, prepend to signed data a tag saying what kind of signature it is
  - GCKS pop
  - Member pop
  - Groupkey push
A CODA

Most Important Need

- NRL Protocol Analyzer, and other formal crypto protocol analysis tools, don’t support incremental analysis well
- Even minor changes may require complete reverification
- As a result did complete formal analysis of system at only one stage
- What’s needed is a verification method that
  - is consistent with methods used by protocol designers
  - supports incremental verification

LOGIC FOR CRYPTO PROTOCOL ANALYSIS

- Work with Dusko Pavlovic, John Mitchell, Anupam Datta, Ante Derek
- Basic idea:
  - Axioms for deriving conclusions about protocol traces from messages received by principals
    - E.g: If A sends a challenge, to B, and gets an authenticated response from B, then A knows that B responded after A’s challenge
  - Logic provides means for composing proofs
- Applying it to GDOI with Dusko Pavlovic
  - Evaluating logic as we apply it
  - Using feedback from GDOI analysis to extend and improve it
  - Also doing this for Kerberos

GDOI AND POP AGAIN

- Recall that certificates “may” be used to distribute public key certificates in GDOI
- Proof of possession uses challenge-response to prove that you actually know the private key
  - Same nonces used for PoP as for challenge-response in core GDOI
- Language in current version of GDOI seems to indicate that certificates can be used to distribute new identities as well
  - There are two alternative means for authoring the GDOI/KEY-FULL messages. First, the Phase 1 identity of the certificate in the request is used. Second, a new identity can be passed in the GDOI/KEY-FULL request. The new identity could be specific to the group and be a certificate that is signed by the group owner to identify the holder as an authorized group member. The proof of possession payload validates that the holder possesses the secret key associated with the Phase 1 identity.
  - What can you prove from PoP in that case?

ATTEMPTED TO DERIVE PROOF

- Able to link request for key to Phase 1 identities
  - Showed that request for key came from possessor of phase 1 identity
- Able to link POP to identity in certificate
  - Showed that POP showed that principal named in certificate is in possession of key
  - What we couldn’t show:
    - That there is any link between phase 1 identity and principal in certificate!
    - Because there isn’t any!

AN ATTACK

Suppose that I is a GCKS that wants to join a group managed by another GCKS, B. Suppose that I doesn’t have the proper credentials to join B’s group. Then I can trick a member A who does into supplying them, as follows.

1. A → I: HDR*, HASH(1), Ni, ID  A requests to join I’s group, sending a nonce Ni
2. I → A: HDR*, HASH(2), Nr, SA  I responds to member A, but using B’s nonce Nr
3. B → I: HDR*, HASH(3), CERT(for A’s ID in group), POP = S_A(hash(Ni,Nr))  B sends keying information to I under impression the identity in A’s certificate belongs to I

\[ S_A(hash(Ni,Nr)) \]
CONCLUSION:
A VERIFIER’S WORK IS NEVER DONE