A progress report on using Maude to verify protocol properties using the strand space model

Presented by

Robert P. Graham, MAJ, USAF/AFIT
Stephen W. Mancini, 1Lt, USAF/AFIT

Presentation date: 01 Oct 03
Objectives

• Briefly introduce MAUDE as a tool for exploring the Strand Space Model (SSM) for Security Protocol Analysis¹

• Demonstrate preliminary results of using MAUDE in the multiset rewriting and strand space approaches

• Solicit some ideas for potential directions of work
Overview

• Research goals
• Introduction to MAUDE
• Multiset rewriting in Maude
  – Needham-Schroeder
  – Penetrator
• Strand Space Modeling in MAUDE
  – Needham-Schroeder
  – Penetrator
• Summary
Research Goals

• Automate Guttman’s Authentication Tests
• Analyze Kerberos protocol (possibly others)
  – Verify it passes automated authentication tests
  – Check for vulnerabilities based on modeling of penetrator strands
  – Athena ran numerous protocols in a short time frame; my focus is only on a few protocols
• Search for alternative way to model penetrator activity
• Don’t reinvent the wheel!
What is Maude?

- Maude is a “reflective equational rewrite logic programming language”
  - Allows for concurrent execution of equations and rules
  - Programs consist of functional and System modules
    - Functional modules define data types and operations by means of equational theories
    - System modules specify rewrite theories
  - Reflection allows programmer to define operations and strategies at the meta level
- Maude’s design maximizes three dimensions:
  - Simplicity: Programs are easy to understand!
  - Expressiveness: Allows for easy expression of a large variety of applications
  - Performance: Fast execution for prototyping and real applications
    - Put together Needham-Schroeder model very quickly
- Our code uses core Maude
  - Maude also has something called Full Maude
Terms in MAUDE

fmod TERM is
  sorts Key Text Term .
  subsorts Key Text < Term .
  vars A G H : Term .
  var K : Key .
  var T : Text .

  op __ : Term Term -> Term [ctor prec 40] .
  op {{}_}_ : Term Key -> Term [ctor] .
  eq {{H}K}inv(K) = H .
  eq {{H}inv(K)}K = H .

  op new : Text -> Text .
  op inv : Key -> Key .
  eq inv(inv(K)) = K .

  op __<=_ : Term Term -> Bool .
  ...
endfm
Needham-Schroeder in MSR

- Needham-Schroeder model using the multiset rewriting approach:

  *** ROLE: Initiator
  rl [init_0] : pr(A) pr(B) pubkeyof(B, Kb) prvkeyof(A, Ka-1)
  => pr(A) pr(B) pubkeyof(B, Kb) prvkeyof(A, Ka-1) I0(A, B, Ka-1, Kb) .
  rl [init_1] : I0(A, B, Ka-1, Kb) => I1(A, B, Ka-1, Kb, new(A)) N({new(A) A}Kb) .
  crl [init_2] : I1(A, B, Ka-1, Kb, Na) N(msg) => I2(A, B, Ka-1, Kb, Na, Nb)
  if (Na Nb) := {msg}Ka-1 .
  rl [init_3] : I2(A, B, Ka-1, Kb, Na, Nb) => I3(A, B, Ka-1, Kb, Na, Nb) N({Nb}Kb) .

  *** ROLE: Responder
  rl [resp_0] : pr(A) pr(B) pubkeyof(A, Ka) prvkeyof(B, Kb-1)
  => pr(A) pr(B) pubkeyof(A, Ka) prvkeyof(B, Kb-1) R0(A, B, Ka, Kb-1) .
  crl [resp_1] : R0(A, B, Ka, Kb-1) N(msg) => R1(A, B, Ka, Kb-1, Na)
  if (Na A) := {msg}Kb-1 .
  rl [resp_2] : R1(A, B, Ka, Kb-1, Na)
  => R2(A, B, Ka, Kb-1, Na, new(B)) N({Na new(B)}Ka) .
  crl [resp_3] : R2(A, B, Ka, Kb-1, Na, Nb) N(msg) => R3(A, B, Ka, Kb-1, Na, Nb)
  if Nb := {msg}Kb-1 .
• A multiset-rewriting model of a Penetrator as developed in Maude
  – Uses the "standard" theory where none of the rules consume or destroy
    the Penetrator's knowledge

\[
\text{op I} : \text{Term} \rightarrow \text{State}.
\]

\[
\begin{align*}
\text{rl [rec]} & : \text{N(H)} \Rightarrow \text{I(H)}. \\
\text{rl [dcmp]} & : \text{I(G H)} \Rightarrow \text{I(G) I(H) I(G H)}. \\
\text{rl [snd]} & : \text{I(H)} \Rightarrow \text{N(H) I(H)}. \\
\text{rl [cmp]} & : \text{I(G) I(H)} \Rightarrow \text{I(G H) I(G) I(H)}. \\
\text{rl [encr]} & : \text{I(H) I(K)} \Rightarrow \text{I({H}K) I(H) I(K)}. \\
\text{rl [nnc]} & : \text{S} \Rightarrow \text{I(new(Pen))) S}.
\end{align*}
\]
Maude> search [20] init1 init2 =>* R2(Alice, Bob, Ka-1, Kb, Na, Nb) S.

Solution 1 (state 42281)
states: 42282 rewrites: 109014 in 5600ms cpu (5602ms real) (19466 rewrites/second)
S --> pr(Alice) pr(Bob) pr(Carol) N({new(Alice) new(Bob)}K1) l(inv(K3))
  pubkeyof(Alice, K1) pubkeyof(Bob, K2) pubkeyof(Carol, K3) prvkeyof(Alice, inv(K1))
  prvkeyof(Bob, inv(K2)) prvkeyof(Carol, inv(K3)) l1(Alice, Bob, inv(K1), K2, new(Alice))
Ka-1 --> K1
Kb --> inv(K2)
Na --> new(Alice)
Nb --> new(Bob)
Maude> search [20] init1 init2 =>* I(new(Bob)) R2(Alice, Bob, Ka-1, Kb, Na, Nb) S .

- In this search, never found an answer in 24 hours!
  - State space explosion!


- In this search, found numerous examples of when the penetrator learned a nonce in a very short time
MSR Lessons Learned

• Maude suffered from state space explosion
• Did well when searching for individual states
• Good learning tool!
Strand Spaces in MAUDE

• A single rewrite rule “executes” a set of strands:
  \[
  \text{var } S, T : \text{Strand} .
  \]
  \[
  \text{var } H : \text{Term} .
  \]
  \[
  \text{rl [label-p-reduction]} : + H S \mid - H T => S \mid T .
  \]
• Definition of the Needham-Schroeder protocol:

```plaintext
var A B : Text .  *** Think of A and B as names (certificates)
var Na Nb : Text .  *** Think of Na and Nb as nonces generated by A and B, resp.
var Ka Kb Kp K : Key .  *** Think of Ka and Kb as public keys owned by A and B, resp.
ops nsInitiator nsResponder : Text Text Text Key Key -> Strand .
op nsPenetrator : Text Text Text Key Key Key -> Strand .

eq nsInitiator(A, Na, Nb, Ka, Kb)
    = + {| Na A |} Kb - {| Na Nb |} Ka + {| Nb |} Kb .
eq nsResponder(A, Na, Nb, Ka, Kb)
    = - {| Na A |} Kb + {| Na Nb |} Ka - {| Nb |} Kb .
eq nsPenetrator(A, Na, Nb, Ka, Kb, Kp)
    = D(| Na A |) Kp, inv(Kp)) | E(Na A, Kb)
    | D(| Nb |) Kp, inv(Kp)) | E(Nb, Kb) .
```
SSM Traces

Maude> rew nsNormal(A, Na, Nb, Ka, Kb).
rewrite in NS-TEST : nsNormal(A, Na, Nb, Ka, Kb).
*********** rule
rl + H:Term S:Strand | - H:Term T:Strand => S:Strand | T:Strand [label
   label-p-reduction] .
H:Term --> { | Na A |}Kb
S:Strand --> - { | Na Nb |}Ka + { | Nb |}Kb
T:Strand --> + { | Na Nb |}Ka - { | Nb |}Kb
+ { | Na A |}Kb - { | Na Nb |}Ka + { | Nb |}Kb | - { | Na A |}Kb + { | Na Nb |}Ka -
   { | Nb |}Kb
--->
- { | Na Nb |}Ka + { | Nb |}Kb | + { | Na Nb |}Ka - { | Nb |}Kb
*********** rule
rl + H:Term S:Strand | - H:Term T:Strand => S:Strand | T:Strand [label
   label-p-reduction] .
H:Term --> { | Na Nb |}Ka
S:Strand --> - { | Nb |}Kb
T:Strand --> + { | Nb |}Kb
+ { | Na Nb |}Ka - { | Nb |}Kb | - { | Na Nb |}Ka + { | Nb |}Kb
--->
- { | Nb |}Kb | + { | Nb |}Kb
*********** rule
rl + H:Term S:Strand | - H:Term T:Strand => S:Strand | T:Strand [label
   label-p-reduction] .
H:Term --> { | Nb |}Kb
S:Strand --> empty
T:Strand --> empty
+ { | Nb |}Kb | - { | Nb |}Kb
--->
empty | empty
rewrites: 6
result Bundle: empty | empty
Maude>
Penetrator in SSM

- Standard penetrator strands representation
- Focus is on the key functions: Encrypt and Decrypt
- Maude makes penetrator operation much easier

\[\text{Encrypt} \quad \rightarrow m \quad \downarrow \quad \rightarrow k \quad \downarrow \quad \rightarrow \{m\}_k \quad \rightarrow \{m\}_k \rightarrow\]

\[\text{Decrypt} \quad \rightarrow \{m\}_k \quad \downarrow \quad \rightarrow k' \quad \downarrow \quad \rightarrow m \rightarrow\]
Penetrator in SSM, cont.

• This receives a key and message and sends out decrypted plaintext:
  \[
  \text{op } D : \text{Term Key } \rightarrow \text{Strand}.
  \]
  \[
  \text{eq } D(M, K) = - K - M + |M| K.
  \]

• This receives a key and message and sends out ciphertext:
  \[
  \text{op } E : \text{Term Key } \rightarrow \text{Strand}.
  \]
  \[
  \text{eq } E(M, K) = - K - M + |M| K.
  \]
Maude> rew nsSpoof(A, Na, Nb, Ka, Kb, Kp) .
rewrite in NS-TEST : nsSpoof(A, Na, Nb, Ka, Kb, Kp) .
************ rule
rl + H:Term S:Strand | - H:Term T:Strand => S:Strand | T:Strand [label
label-p-reduction] .
H:Term --> Nb
S:Strand --> empty
T:Strand --> empty
+ Nb | + Kb | + Kb | + inv(Kp) | + inv(Kp) | + Na A | + {| Nb |}Kb | + {| Na A
}]Kb | - Nb | - Kb | - Kb | - inv(Kp) | - inv(Kp) | - Na A | - {| Nb |}Kp |
- { | Na A }Kp | + { | Na A }Kp - { | Na Nb }Ka + { | Nb |}Kp | - { | Na A
}]Kb + { | Na Nb |}Ka - { | Nb |}Kb
--->
(+ Kb | + Kb | + inv(Kp) | + inv(Kp) | + Na A | + {| Nb |}Kb | + {| Na A |}Kb | - Kb | - Kb | - inv(Kp) | - inv(Kp) | - Na A | - {| Nb |}Kp | - { | Na A
}]Kp | + { | Na A |}Kp - { | Na Nb |}Ka + { | Nb |}Kp | - { | Na A |}Kb + { | Na
Nb |}Ka - { | Nb |}Kb) | empty | empty
********** rule
rl + H:Term S:Strand | - H:Term T:Strand => S:Strand | T:Strand [label
label-p-reduction] .
H:Term --> {|| Nb ||}Kp
S:Strand --> empty
T:Strand --> empty
empty | empty | empty | empty | empty | empty | empty | empty | empty | empty |
empty | empty | empty | empty | empty | empty | empty | empty |
| + {|| Nb ||}Kp | - {|| Nb ||}Kp
--->
(empty | empty | empty | empty | empty | empty | empty | empty | empty | empty |
| empty | empty | empty | empty | empty | empty | empty | empty | empty | empty)
| empty | empty | empty | empty | empty | empty | empty | empty |
rewrites: 25
result Bundle: empty | empty | empty | empty | empty | empty | empty | empty |
| empty | empty | empty | empty | empty | empty | empty | empty |
| empty | empty | empty | empty | empty | empty | empty | empty |
| empty | empty | empty | empty | empty | empty | empty | empty |
| empty
SSM Lessons Learned

• Maude accurately reduced states, verifying known weakness of Needham-Schroeder in very short time
• Allowed easy representation of strand space model
• Good learning tool!
• No search (which is good, and bad)
Course of Action

• Generate the code for the Authentication Tests
• Run automated tests against Needham-Schroeder
• Generate Maude code for Kerberos protocol
• Generate code for other protocols?
Summary

• Research goals
• Introduction to MAUDE
• Multiset rewriting in Maude
• Strand Space Modeling in MAUDE
Bibliography


