The NRL Protocol Analyzer

CSE914
Instructor: Dr. Cheng
Computer Science and Engineering – MSU

Presenters: Ali Ebnenasir & AliReza Namvar

Outline
- Motivations
- The NRL Protocol Analyzer
- The Security Model of the NRL analyzer
- The Architecture of the NRL Protocol Analyzer
- Specification Language
- How does the NRL Analyzer work?
- Analysis
- Examples
- Conclusions

Motivations
- Motivations
  - Design:
    - Security protocols must work under adverse conditions
  - Maintenance:
    - Finding flaws of existing protocols
- How to achieve these goals?
  - A formal system for the specification and reasoning about security properties
  - Automation of the analysis
    - Use logic to model the security requirement (e.g., BAN)
    - Develop formal models based on the algebraic term-rewriting systems (e.g., The NRL analyzer)
  - What is a term-rewriting system?
Term-Rewriting Systems

- Characteristics:
  - Obtaining new words from given ones
  - Rewrite rules

- Formal Definition
  - A pair (\(\Sigma, P\)), where
    - \(\Sigma\): an alphabet
    - \(P\): a finite set of ordered pairs \((u, w)\) of words \(u\) and \(w\) over \(\Sigma\)
      (i.e., production/rewrite rule \(u \rightarrow w\))

- Chomsky defined his classification of grammars by imposing restrictions on \(P\)

The NRL Protocol Analyzer

- A software tool (model checker)
  - Verifying security protocols
  - Finding flaws

- Challenges
  - Cryptographic words should obey certain reduction/rewrite rules
  - The search space is unbounded

Security Model in the NRL analyzer

- Worst-case model of Dolev & Yao

- The intruder can:
  - Read messages
  - Destroy messages
  - Alter messages
  - Create new messages
  - Know some secret keys

- Thus:
  - Any received message is assumed to be sent by the intruder
  - Any sent message is assumed to be received by the intruder
  - Each protocol is an algebraic system manipulated by the intruder
Security Model Components

- **Words**
  - Generated by rewriting rules
  - e.g., \( d(X, e(X, Y)) = Y \)

- **Protocols: State Transition Machine**
  - **State:**
    - Set of words (known by the intruder)
    - Set of local variables and their values
  - **Transition:**
    - Input state: received messages and the values of local state variables
    - Output state: sent messages and the new values of local state variables

Security Model Assumptions

- The number of keys and words are assumed infinite
- Unbounded number of executions
- A principal may be engaged in more than one execution

- **Result:** infinite state space for the model
- **Model:** Infinite State Automata

The Architecture of the NRL Analyzer

- **Language Generator:** Filters unreachable states
- **State Unifier:** Identifies states that are reachable
- **UI Parser:** Description of an insecure state
- **State Generator:** Preceding states
- **Filter:** A set of rules
  - Description of a state
  - Reachability path
  - Query
  - Specified by the user
The Specification Language

- Development Language: The Prolog

The specification language:
- close to natural language
- translated to Prolog

We present the syntax and semantics of
- The NRL analyzer specification language
- The Prolog and its mechanism of unification
  - Equational Unification is the mechanism by which the model is checked

The Specification Language of the NRL analyzer

- Protocol specification sections
  - Words
  - Transition Rules

1. Words
- The type specification
  type: Key isa Word: L.

- The atom specification (designated by alphanumeric strings)
  - A word that is not constructed out of other words
  - Format: `atom(1): honest --> H:
  atom(2): dishonest --> H:.

The Specification Language of the NRL analyzer – cntd.

- The operations specification
  - Defining words produced by the operation of system participants (e.g., users, servers, the intruder)
  - Format: `Op(N): f(X1,..., Xn) \rightarrow W \ (or nopen)
  - E.g., `Op(1): e(X, Y) \rightarrow W; nopen.

- The function symbol specification
  - Differentiates between atoms
  - Identify different parties who play the same role
  - Format: `fs(N): f(X1,..., Xn) \rightarrow Y ::.
  - E.g., `fs(8): server(A) \rightarrow U::.
  `fs(9): server(A, H) \rightarrow U::.

- The intruder is not able to compute a function
The Specification Language of the NRL analyzer – cntd.

- The specification of words initially known by the intruder
  - E.g., User name, keys
  - Format: known: user(A, X), key(user(A, dishonest)).

- Rewriting rules
  - Noetherian: No infinite sequence of reductions is possible
  - Confluent: Any term is reducible to a unique normal from.
  - E.g., m(N) (term1) => term2.

- The beliefs specification
  - Format: beliefs: LIST
  - LIST is a list of equalities among state variables
  - Describes values that different parties should hold in common
    at the end of protocol
  - E.g., init_seskey = rec_seskey

The Specification Language of the NRL analyzer – cntd.

2. Transitions Rules
   - Actions taken by an honest principal in the protocol
   - Rule Types:
     - Session rules: describe each role played in the protocol
     - Subroutine rules: describe the rules invoked by other rules
     - Subprotocol rules: may fire at some particular states
   - Format:
     - The identifier: the input message: the body: the event statement

2.1 The identifier
   - RULETYPE RULENAME(NAME, ROUND, TIME)
   - E.g., Subprotocol init_name(principal(A,honest), N, T) of init_main

2.2 The input message
   - In case a rule is expecting a message to fire
   - Format: rec msg(SENDER, RECEIVER, MESSAGE, ROUND)

2.3 The rule body
   - Describes the input conditions necessary for a rule to fire
     and the resulting output
   - Simple or Complex body
     - If: INPUT MESSAGE:
       INPUT:
       Then:
       OUTPUT:
     - Complex form consists of a nested if statement
The Specification Language of the NRL analyzer – cntd.

Rule Input:
- Conditions of the form `verify(X, Y)` and `not(verify(X, Y))`
- `verify(X, Y)` causes X and Y to be narrowed with each other
- E.g., `verify(W, Z), d(S, T)`
- E.g., `verify(X, user(A, H))`
  - Guards against attacks that are dependent upon a principal is mistaking one type of word for another
- `not(verify(X, Y))` does not mean the negation of `(verify(X, Y))`
- Will succeed only if X cannot be subsumed by Y
- "," and "+" respectively stand for conjunction and disjunction

Rule Output:
- Can be Simple or Complex
- Complex form is specified as a sequence of OR statements

Rule Output:
- Can be Simple or Complex
- Complex form is specified as a sequence of OR statements

The Specification Language of the NRL analyzer – cntd.

The format of each OUTPUTPIECE
- `STATEVAR := Y`
- `init_dest := principal(B, H)`
- `SENDMSG := SENDER, RECEIVER, MESSAGE, N`

The format of each OUTPUTPIECE
- `STATEVAR := Y`
- `init_dest := principal(B, H)`
- `SENDMSG := SENDER, RECEIVER, MESSAGE, N`

The Specification Language of the NRL analyzer – cntd.

2.4 The event statement
- What happens when a rule executes
- Format:

```
(event(PRINCIPAL, PRINCIPALIST, RULENAME, TERMINLIST, N))
```

- Used in a query
- verifies that a certain sequence of rules will or will not execute
Specification Language - Prolog
(Syntax and semantics of Prolog)

- The Prolog is based on the first-order logic
  - Symbols: logical operators
    - Conjunction: \( \land \)
    - Disjunction: \( \lor \)
    - Implication: \( \rightarrow \)
    - Negation: \( \neg \)
  - Terms: constant, variable, or an expression \( tname(Y_1, \ldots, Y_n) \) where \( Y_i \) are terms
- A Prolog program: a set of clauses
  - Facts:
    - Syntax: \( pname(X_1, \ldots, X_n) \) where \( X_i \) are terms
    - Semantics: \( pname(X_1, \ldots, X_n) \) is true for any possible value of \( X_i \)
  - Rules:
    - Syntax: \( pname(X_1, \ldots, X_n) \rightarrow \text{Condition} \)
    - Semantics: if Condition is true then \( pname(X_1, \ldots, X_n) \)

Specification Language- Prolog
(Sequence Control Mechanism in Prolog)

- Sequence control in Prolog
  - Unification
    - Pattern matching unification
    - Equational unification
  - Pattern matching
    - A substitution that matches two different terms
    - Example:
      - \( X = f(A, t(B, C)) \), \( Y = f(U, V) \)
      - Substitution: \( U \rightarrow A, V \rightarrow t(B, C) \) matches \( X \) and \( Y \)

Specification Language- Prolog
(Sequence Control – Pattern Matching)

- How to recognize a pattern?
  - Example: ParentOf relation
    - Facts:
      - ParentOf(John, Mary)
      - ParentOf(Susan, Mary)
      - ParentOf(Bill, John)
      - ParentOf(Ann, John)
    - Query: An expression that contains variables
      - Query 1: \( \text{ParentOf}(X, Mary) \)
        - Result: Either John or Susan
      - Query 2: \( \text{ParentOf}(X, Y), \text{ParentOf}(Y, Mary), \neg (X=Y) \)
        - Result: John, Susan
    - Building relations that can be inferred logically
      - GrandparentOf(\( X, Y \)): \( \text{ParentOf}(X, Z), \text{ParentOf}(Z, Y) \)
Equational Unification: A more general form of unification
- Finds a fact that makes two terms equal
- Collection of terms \( T \)
- Set of equations \( E \) (set of rules)
- A substitution \( s \) is a unifier of terms \( X \) and \( Y \) with respect to \( E \) if:
  - \( sX \) and \( sY \) can be made equal using equations of \( E \)
- Example: \( \text{ParentOf} \,(X, \text{Mary}) = \text{ParentOf} \,(\text{John}, Y) \)
  \( \text{ParentOf} \,(\text{John}, \text{Mary}) \) unifies \( \text{ParentOf} \,(X, \text{Mary}) \) and \( \text{ParentOf} \,(\text{John}, Y) \)
- How does the unification work in Prolog?
  - \( \text{add}(Y, 0, X) \) :- \( Y = X \)
  - \( \text{add}(Z, X, Y) \) :- \( W \) is \( X - 1 \), \( \text{add}(U, W, Y) \), \( \text{succ}(Z, U) \)
- Query: \( \text{add}(Y, 2, 3) \)

The impediments before equational unification:
- Equational unifiers do not always exist
- The existence of a unifier depends on how we use the equations of the set \( E \)
- Equational unification problem is not always decidable

In the NRL analyzer
- Equational system is defined by reduction/rewrite rules
  - E.g., \( d(X, d(X, Y)) \rightarrow Y \), \( d(X, e(X, Y)) \rightarrow Y \)
  - \( e(X, Y) \): encryption of \( Y \) with \( X \)
  - \( d(X, Y) \): decryption of \( Y \) with \( X \)
How does the NRL analyzer work?

- Filter unreachable states
- Search the remaining states exhaustively
- How do we identify and filter unreachable states?

How do we identify and filter unreachable states?

Infinite Unreachable States

Finite Reachable States

Infinite State Space

Analysis
(Languages and Unreachability)

- Identify classes of states that an intruder can never learn (unreachable)
  - For example: States that
    - Require knowledge of a word that will not be known until the future
    - Involve use of a word before it is generated
- How can we prove that the intruder can never learn a particular class of words?
  - The answer: Define a language and prove it is unreachable
- How to prove unreachability?
  - Fact: The intruder does not initially know any words
  - Define a formal language
  - Prove an intruder can learn a word \( \text{iff} \) it knows a word

Analysis
(An example of Language Definition)

- Encryption-Decryption protocol

Protocol Rules (INTR denotes the intruder):
- \( R_1: \) INTR knows \( X \) and \( Y \) \( \rightarrow \) INTR can find \( e(X, Y) \)
  - \( e(X, Y) \): encryption of \( Y \) with \( X \)
- \( R_2: \) INTR knows \( X \) and \( Y \) \( \rightarrow \) INTR can find \( d(X, Y) \)
  - \( d(X, Y) \): decryption of \( Y \) with \( X \)

Rewriting rules for protocol words:
- \( d(X, e(X, Y)) \rightarrow Y \)
- \( e(X, d(X, Y)) \rightarrow Y \)
- What words can the intruder learn?
Analysis
(An example of Language Definition –cntd.)

- How can the intruder find word Z?
  - R1:
    - The intruder knows d(W, Z) and W, for some W  \( (F1) \)
    - The intruder knows X and Y, and Z= e(X, Y)  \( (F2) \)
  - R2:
    - The intruder knows e(W, Z) and W  \( (F3) \)
    - The intruder knows X and Y, and Z= d(X, Y)  \( (F4) \)

- Continuing (F3) with R1: How can the intruder know e(W, Z)?
  - The intruder knows W and Z  \( (F3.1) \)
  - The intruder knows X1 and d(X1, e(W, Z))  \( (F3.2) \)

- Continuing (F3) with R2: How can the intruder know e(W, Z)?
  - The intruder knows X1 and e(X1, e(W, Z))  \( (F3.3) \)
  - e(W, Z) is not a decrypted message

- Continuing (F3.2): How can the intruder know d(X1, e(W, Z))? The answer: the intruder should know X2 and e(X2, d(X1, e(W, Z))) or d(X2, d(X1, e(W, Z)))
Analysis
(An example of Language Definition –cntd.)

- A certain regularity in the pattern of words
- Define a language A depending on the state of the intruder’s knowledge
  - A \rightarrow e(L, K)
  - L: the set of all irreducible words
  - K: the set of all irreducible words not known by the intruder
  - A \rightarrow d(L, A)
  - A \rightarrow \delta(L, A)
- Prove A is unreachable
- Take each rule, and prove that
  - The intruder’s learning of a word (produced by that rule) implies that the intruder must already know a word of A

Example 1
(ISO authentication protocol)

Nonce: a unique random number to guarantee that a message has been recently sent.

1. (A, Na)
2. Nb, A, K’\sigma(Nb, Na, A)
3. N’ a, B, K’ a(N’ a, Nb, B)

Ka: The public key of A
K’a: The private key of A

Example 1
(ISO authentication protocol - A sample attack)

A concludes that A has sent a message to him (her) and N’ is a nonce from A.
A has no idea that he or she has initiated a communication with B.
Example 2
(Simmons Selective Broadcast Protocol-1986)

Definition:
- A participant encrypts a message, a specific subset of other participants can decrypt it.
- Limited decryption ability
- Security=encryption(single key)+tamperproof processors
  - Release messages after decryption to participants or store them in the processor

The protocol:
- A Participant--a tamperproof processor A with secret key q(A)
- All processors have a common key k.
- Unique public identifier for each processor e(k,q(A))

A -> B, M:
- A chooses a random identifier J for message M
- A’s processor encrypts M using the key d(q(A), d(k,J))
- A->B, encrypted msg,encrypted key
- B puts msg,key,identifier e(k,q(A)) to his processor
- Processor decrypts the msg using k, e(k,q(A)), q(B)

Example 2
(Simmons Selective Broadcast Protocol-Cntd.)

Tamperproof processor:

In encrypt msg mode:
- e(k,q(A))
- Reduced to J for verification
- e(d(q(A),d(k,J)),M) = e(d(q(A),d(k,J)),M)

In encrypt key mode:
- e(k,e(q(B),d(q(A),e(q(B),d(q(A),d(k,J)))))
- Processor
- X
- Y
- M
- e(k,q(A))
- e(k,e(q(B),d(q(A),e(q(B),d(q(A),d(k,J)))))
- Reduced to J for transfer authentication of message decrypted M

Example 2
(Simmons Selective Broadcast Protocol-Attack Scenario)

The attack:
- Intruder impersonates other members of network
- Assumption: Intruder has access to some processor x
- Honest user a sends the right to program p with Id J to the intruder:
  - Encrypted Id: e(k,q(a))
  - Encrypted key: e(k,e(q(x),d(q(a),e(q(x),d(q(x),d(q(a),d(k,j)))))))
  - Encrypted msg: e(d(q(a),d(k,j)),p)
- Intruder sends these to the processor in encrypt msg mode:
  - X = e(k,q(a))
  - Y = e(k,e(q(x),d(q(a),e(q(x),d(q(x),d(q(a),d(k,j)))))))
  - Wraps a program supplied by the intruder
  - 3rd output reduces to e(d(k,px),px)
- User b requests program p with Id J from user a, user a sends:
  - e(k,q(a))
  - e(k,e(q(b),d(q(a),e(q(b),d(q(a),d(k,j)))))
  - e(d(q(a),d(k,j)), p)
- Intruder intercepts and replaces the 3rd word by e(d(k,px),px) and sends to b
- User b inserts 3 words in his processor and decrypts the msg output:
  - e(k,q(a))
  - 2nd output reduced to J
  - 3rd output reduced to px
Conclusions

- Assessment of the NRL analyzer specification language
  - Unambiguous
    - Well-defined syntax and semantics
  - Simplicity
    - Very close to English
  - Flexibility
    - Appropriate for authentication protocols
  - Powerful
    - Just for security protocols that use encryption to ensure secure communication
  - Insightful
    - Has found flaws in many open literature protocols

Conclusions – cntd.

- What we can do:
  - Correctness with respect to the specification
  - Security with respect to a certain attack, but
  - Yet no technique to prove a system is generally secure
- What we cannot do:
  - Capture subtle properties of security protocols (e.g., susceptibility to replay attack)
  - Prove that a protocol is secure
    - (What does security mean?)
- Future work
  - Develop formal systems that have enough expressive power for the specification of security requirement
  - Combine current techniques
    - (e.g., integration of the NRL and BAN)

References


   [http://citeseer.nj.nec.com/rubin93formal.html]

   [http://citeseer.nj.nec.com/meadows92applying.html]

   [http://citeseer.nj.nec.com/meadows96language.html]