Software Model Checking for Security

Lujo Bauer
18-732
Spring 2015
[slides from Anupam Datta]
Outline

- Security Protocols
- Overview of Model Checking
- Model Checking Software
How can Alice use cryptography to send an authenticated message to Bob over an untrusted network?

- What is authentication?
- What can go wrong?
Authentication by Encryption

A’s reasoning:
The only person who could know NonceA is the person who decrypted 1

st

message

Only B can decrypt message encrypted with Kb
Therefore, B is on the other end of the line
B is authenticated!

B’s reasoning:
The only way to learn NonceB is to decrypt 2

nd

message
Only A can decrypt 2

nd

message
Therefore, A is on the other end
A is authenticated!
What Does This Protocol Achieve?

- Protocol aims to provide both authentication and secrecy
- After this exchange, only A and B know NonceA and NonceB
Anomaly in Needham-Schroeder

Evil agent B tricks honest A into revealing C’s private value Nc

C is convinced that he is talking to A!
Lessons of Needham-Schroeder

- Classic man-in-the-middle attack
- Exploits participants’ reasoning to fool them
  - A is correct that B must have decrypted $\{A,Na\}_{Kb}$ message, but this does not mean that $\{Na,Nb\}_{Ka}$ message came from B
  - The attack has nothing to do with cryptography!

- It is important to realize limitations of protocols
  - The attack requires that A willingly talk to adversary
  - In the original setting, each workstation is assumed to be well-behaved, and the protocol is correct!

- Wouldn’t it be great if one could discover attacks like this automatically?
Outline

- Security Protocols
- Overview of Model Checking
- Model Checking Software
Model Checking

- **Algorithm for checking properties about behaviors of finite state machines**
  - Given a state machine $M$ and a property $\phi$, does $M$ satisfy $\phi$?

- **Discovered independently by Clarke & Emerson and Queille & Sifakis in 1981**

---

2007 Turing Award to Clarke, Emerson & Sifakis
Example: Interaction in Class

State machine \((S, s_0, \Sigma, \delta)\)

- **Actions**, \(\Sigma\): \(\text{blase\_speaks}, \text{sean\_speaks}, \ldots, \text{haley\_raises\_hand}\),

- **States**, \(S\): record sequence of actions leading to it

- **Initial state**, \(s_0\): empty (no actions so far)

- **Transition relation**, \(\delta\): in each state, any action is possible
Example: Interaction in Class

- Property: always raise hand before speaking
- Property violated if there exists a path in the graph where someone speaks without having raised hand (a reachability property)
- Task for model checker: Search graph for such a path
  - If path found, it provides a counterexample to the property
  - Else property holds in model
Example: Interaction in Class

System Characteristics:

- **Concurrent**
  - Each execution is an interleaving of actions by different agents (components)

- **Nondeterministic**
  - From each state, there are many possible transitions

- **Finite state**
  - After suitable abstractions, e.g., what we say does not matter!
Example: Needham-Schroeder Initiator

Protocol instance consists of state machines for initiators, responders, attacker
Example: NS with 1 Initiator, 1 Responder and Adversary

Property: In all states: If I_COMMIT then R_COMMIT or R_WAIT

sendmsg1

intruder-receivemsg1
Model Checking: Pros and Cons

- **Pros:**
  - Fully automated (given a model)
  - Fast (depending on setting + relative to similar rigorous methods)
  - Counterexample useful for diagnostics

- **Cons**
  - No correctness proof
  - Applies to finite model (though some exceptions)
  - State space explosion (many techniques to address)
  - Models created manually (exception: software model checking)
Model Checking for Security

- Model checking security protocols
  - Ryan & Schneider, Modelling and Analysis of Security Protocols, Addison Wesley 2001

- Model checking secure system designs
  - Ongoing efforts on model checking TPM spec (e.g., at MITRE), security hypervisors (CMU)
Outline

- Security Protocols
- Overview of Model Checking
- Model Checking Software
  - Counterexample-guided Abstraction-Refinement
    - Security protocol code analysis (OpenSSL; CMU)
  - Bounded Model Checking
    - Security hypervisor (XMHF; CMU)
  - Focus on reachability properties: Is a “bad” (a.k.a. attack) state reachable?
Verifying Security of OpenSSL

Client Code

```c
int state = 0;
while(1) {
  if(state==0) {
    send_hello();
    state++;
  } else if(state==1) {
    ver = recv_hello();
    state++;
  } else ...
}
```

Server Code

```c
int state = 0;
while(1) {
  if(state==0) {
    recv_hello();
    state++;
  } else if(state==1) {
    send_hello(ver);
    state++;
  } else ...
}
```

Finite Client Session Model

Attacker Model

Finite Server Session Model

Infinite state system

Sound finite abstraction that is model checked
if (x) {
    y = x;
} else {
    y = x + 1;
}
assert (y);

Program: Syntax

Control Flow Graph

Model: States and transitions

Goal: Extract finite state model automatically from C code
Predicate Abstraction

Extract finite model:
Partition the state space based on values of a finite set of predicates on program variables

Typically, predicates will involve variables that appear in the property and control flow conditions

\[ y = x + 1 \]  
\[ y = x \]
Predicate Abstraction: States

- $P = (y == 0)$
- $\neg P$

States where $y \neq 0$

States where $y = 0$

ERROR

P

P = (y == 0)

States where $y = 0$

States: One copy of CFG per valuation of predicates
Predicate Abstraction: Transitions

Transition exists because the following weakest precondition is satisfiable:

\((x + 1 \neq 0)\) and \((y \neq 0)\)
Verification via Finite-State Model Checking

\[ P = (y == 0) \]

\[ \neg P \]

\[ \phi = G (\neg \text{ERROR}) \]

Note: weakest precondition of each statement considered separately.
Model Checking

\( \phi = G (\neg \text{ERROR}) \)

\( P = (y == 0) \)
Model Checking

\[ P = (y == 0) \]

\[ \neg P \]

ERROR

P = (y == 0)
Counterexample Validation & Refinement

Validation
- Simulate counterexample symbolically
- Call a theorem prover to determine if the precondition is satisfiable

Refinement
- New set of predicates \{x==0,y==0\}

Weakest preconditions of sequence of program statements considered together

Spurious counterexample because x == 0 from assignment and x != 0 from conditional, i.e. unsatisfiable precondition
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\begin{align*}
P &= (x == 0) \\
Q &= (y == 0) \\
\neg P \
\neg Q \\
P \land Q \\neg P \land Q \\

\begin{align*}
x &= 0 \quad y &= 0 \\
x \neq 0 \quad y \neq 0 \\
P &= (x == 0) \\
Q &= (y == 0) \\
\end{align*}

\begin{align*}
\text{assert (y)} \\
\text{ERROR} \\
\text{ERROR} \\
\end{align*}
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\[
P = (x == 0) \quad Q = (y == 0)
\]

\[
\neg P \quad \neg Q
\]

\[
P \quad Q
\]

\[
\neg P \quad Q
\]

\[
P \quad \neg Q
\]

\[
\neg P \quad \neg Q
\]

\[
P = (x == 0) \quad Q = (y == 0)
\]

\[
x = 0 \quad y = 0
\]
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\begin{align*}
&D = (x == 0) \quad E = (y == 0) \\

&y = x + 1 \quad y = x \\

&\text{assert (y)}
\end{align*}

<table>
<thead>
<tr>
<th>no</th>
<th>yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y = x + 1)</td>
<td>(y = x)</td>
</tr>
</tbody>
</table>

| \(\neg P \quad \neg Q\) | \(\neg P \quad Q\) | \(P \quad \neg Q\) | \(P \quad Q\) |

| x \neq 0 \quad y \neq 0 | \(P = (x == 0)\) \quad Q = (y == 0) | x = 0 \quad y = 0 |
Predicate Abstraction: 2\textsuperscript{nd} Iteration

\[
P = (x == 0) \quad Q = (y == 0)
\]

\[\neg P \quad \neg Q\]

\[P \quad \neg Q\]

\[\neg P \quad Q\]

\[x \neq 0 \quad y \neq 0\]

\[P = (x == 0) \quad Q = (y == 0)\]

\[x = 0 \quad y = 0\]
Model Checking: 2nd Iteration

\[ \phi = G (\neg ERROR) \]

\[ P = (x == 0) \quad Q = (y == 0) \]

SUCCESS
Iterative Refinement: Summary

Choose an initial set of predicates, and proceed iteratively as follows:

1. **Abstraction**: Construct an abstract model $M$ of the program using predicate abstraction.

2. **Verification**: Model check $M$. If model checking succeeds, exit with success. Otherwise, get counterexample $CE$.

3. **Validation**: Check $CE$ for validity. If $CE$ is valid, exit with failure.

4. **Refinement**: Otherwise, update the set of predicates and repeat from Step 1.
Soundness of Abstraction

**Soundness**: If property holds on abstract finite model, then it also holds in the concrete infinite model.
Software Model Checking Tools

- **Iterative Refinement**
  - SLAM, BLAST, MAGIC, Copper, …
  - SLAM successfully applied to device driver verification at Microsoft

- **Bounded Model Checking**
  - CBMC, …

- **Others**
  - Engines: MOPED, BEBOP, BOPPO, …
  - Java: Java PathFiner, Bandera, BOGOR, …
  - C: CMC, …
Comparison with DART & EXE

- DART/EXE does not perform abstraction (work with constraints involving all variables) unlike CEGAR.
- Complex constraints problematic for CEGAR; no fall back to concrete random inputs like DART.
- EXE useful for finding bugs; does not guarantee absence of bugs (reachability) unlike CEGAR and DART if they terminate.
- CEGAR and DART may not terminate; EXE terminates.

process we discussed today