A Model-based Approach to Security Flaw Detection of Network Protocol Implementations

Yating Hsu, Guoqiang Shu and David Lee

Department of Computer Science and Engineering

The Ohio State University
{hsuya, shug, lee}@cse.ohio-state.edu

October 21, 2008

Outline

- Background
  - Fuzz Testing
  - Security Flaw Detection
- Our Approach
  - Active and Passive Protocol Synthesis Algorithms
  - Model-based Test Input Selection
- Experimental Results
- Summary and Future Works

Background

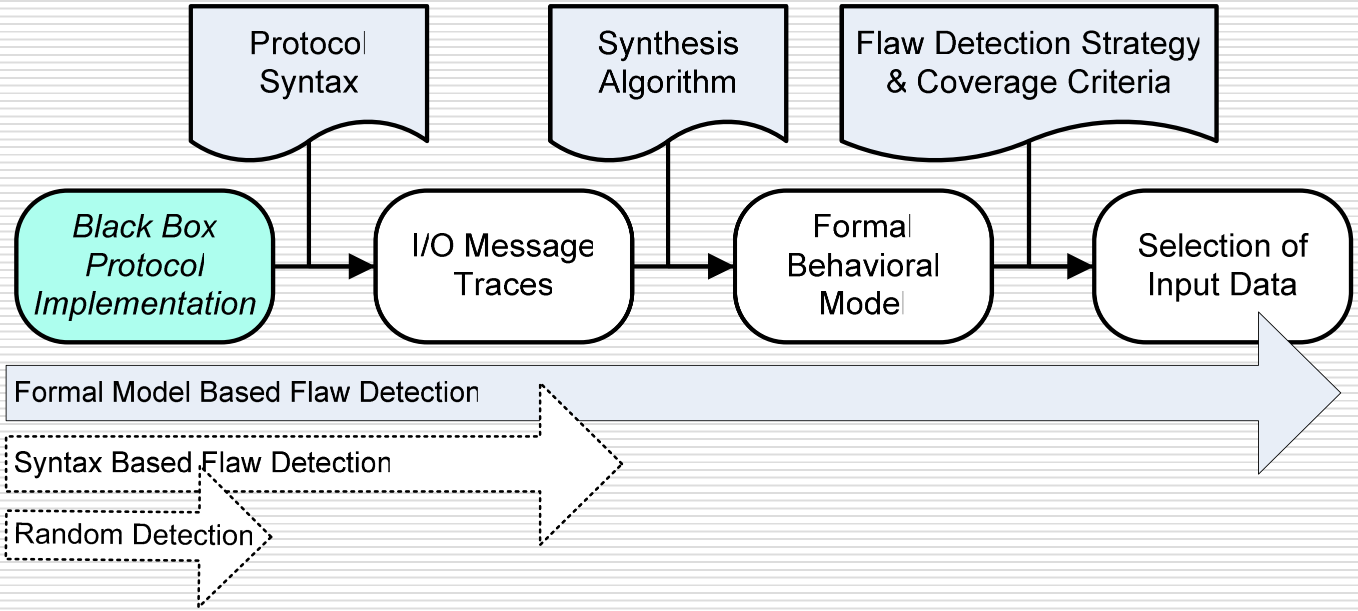
- Security Flaws in Protocol Implementation
  - Improper handling of input data
- Fuzz Testing
  - Mutating input data to reveal unwanted behaviors
  - Low cost and effective

Existing Methods and Tools

- Black-box fuzz testing
  - Protocol specific
    - FTP fuzzer, java script fuzzer
  - Random or manual selection of mutated input
  - Syntax based fuzzing
  - Incomplete specification
- Limitations
  - Low coverage
    - Same message type, different roles
  - Lack of measurability
  - Hard to automate test selection

Our Approach: Model-based Fuzzing

- Principle – use an automatically synthesized formal protocol model to guide fuzz testing
  - Fully automated
  - Improved measurability



A Formal Protocol Model

- Protocol message abstraction
  - MSG_I , MSG_O: input and output messages
  - Abstraction functions
    - $\alpha:MSG_I \rightarrow A_I$, $\beta:MSG_O \rightarrow A_O$
- Model of implementation
  - FSM $<S, s_0, A_I, A_O, f_{next}, f_{output}>$

A Formal Protocol Model

- Flaw detection problem
  - A protocol implementation *B*
  - A predefined predicate to determine which output sequence represents a flaw
    - $GOAL{:}MSG_O^* \rightarrow \{true, false\}$
  - An input selection strategy
- Assumptions
  - The implementation is deterministic
  - A protocol message parser is available
  - Input messages have practical constraints for them to be executable

Active Synthesis Algorithm

- Supervised FSM Learning
  - Based on Angluin's L^* algorithm
  - Iterative refinement of a conjectured model
    - Using queries and counter examples
  - Simulate teacher using conformance testing
  - Guaranteed progress but high cost
- Limitations
  - High cost due to teacher simulation
  - Query requires constructing arbitrary input message
  - May not be practical for fuzz testing

Passive Synthesis Algorithm

- Step (1): traces abstraction
  - Removing session related fields
- Step (2): identifying possible loops
  - Repetitive and consecutive sub-traces
  - Remove all the loops, but restore them after step (4)
- Step (3): construct a tree FSM
  - Follow a prefix of the trace that is already in the tree FSM
  - If necessary, create a new branch

Passive Synthesis Algorithm

- Step (4): tree FSM reduction
  - Identify equivalent states bottom up
    - Two states are equivalent ⇔ two subtrees are isomorphic
    - Compare and color all the states of height 1
      - Two states are equivalent ⇔ they have a same color
    - Shrink the processed subtrees into a node
      - The height of tree FSM decreases by 1
    - Repeat until the height of the tree is 1
  - Merge equivalent states top down
    - End up with a DAG FSM
  - Append loops from Step (2) to DAG FSM

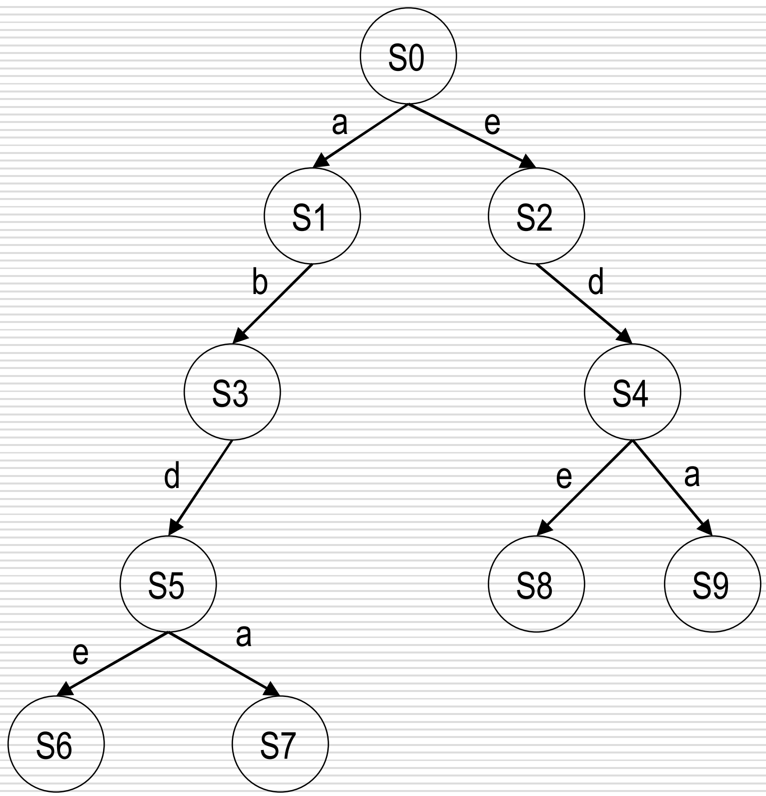
Passive Synthesis Algorithm - Example

tr_1={a,b,d,a}
tr_2={c,c,a,b,d,e}
tr_3={e,a,b,a,b,a,b,d,e}
tr_4={e,d,a}

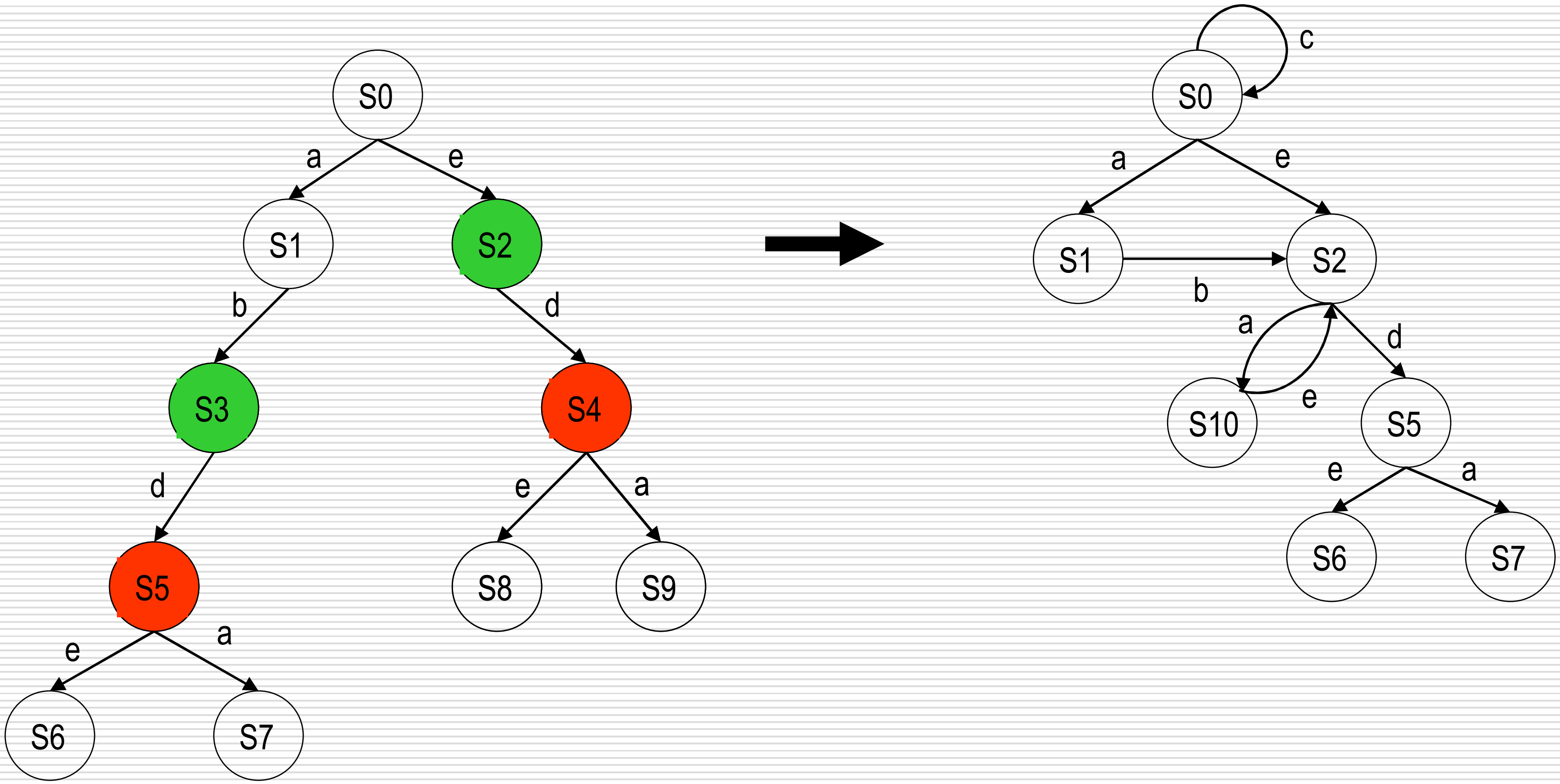
1. remove loops

tr_1={a,b,d,a}
tr_2={a,b,d,e}
tr_3={e,d,e}
tr_4={e,d,a}

2. construct a tree FSM



Passive Synthesis Algorithm – Example (cont.)



Model based Input Selection

- Coverage Criteria
  - Evaluates the portion of the model covered
    - Example: transition coverage
  - Select input message that increase the coverage metric

$$TR\_Coverage = \frac{\left|\{< s', LAST_i >| \; s' = f_{next}(s_0, PREFIX_i) \wedge f_{next}(s', LAST_i) \downarrow, 0 \le i \le K\}\right|}{\left|\{< s, i >| \; f_{next}(s,i) \downarrow, s \in S, i \in I\}\right|}$$

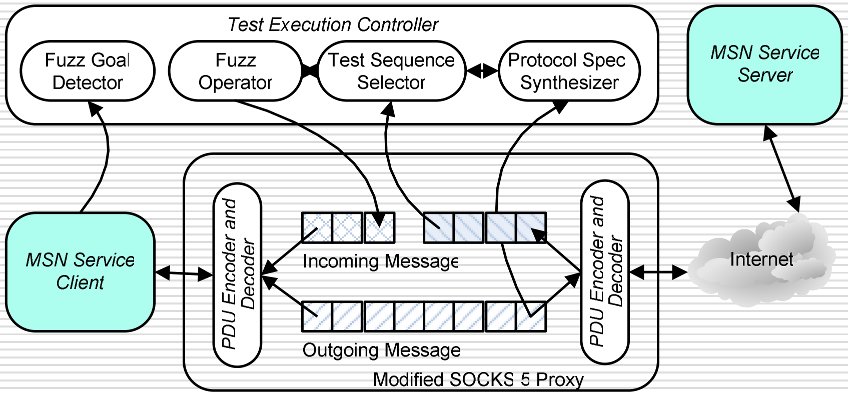
- Fuzzing function
  - $I_0 I_1 \ldots I_k \, f_{fuzz}(I_{k+1}) \ldots I_L$
  - Single transition fuzzing function

Experimental Study – MSNIM Protocol

- MSN Instant Messaging Protocol
  - Proprietary and text-based
  - Protocol syntax
    - *<msg-type> <parameter 1> <parameter 2> …*
- Implementations we test
  - aMSN and Gaim, on Linux (Ubuntu) and Windows (Windows XP)

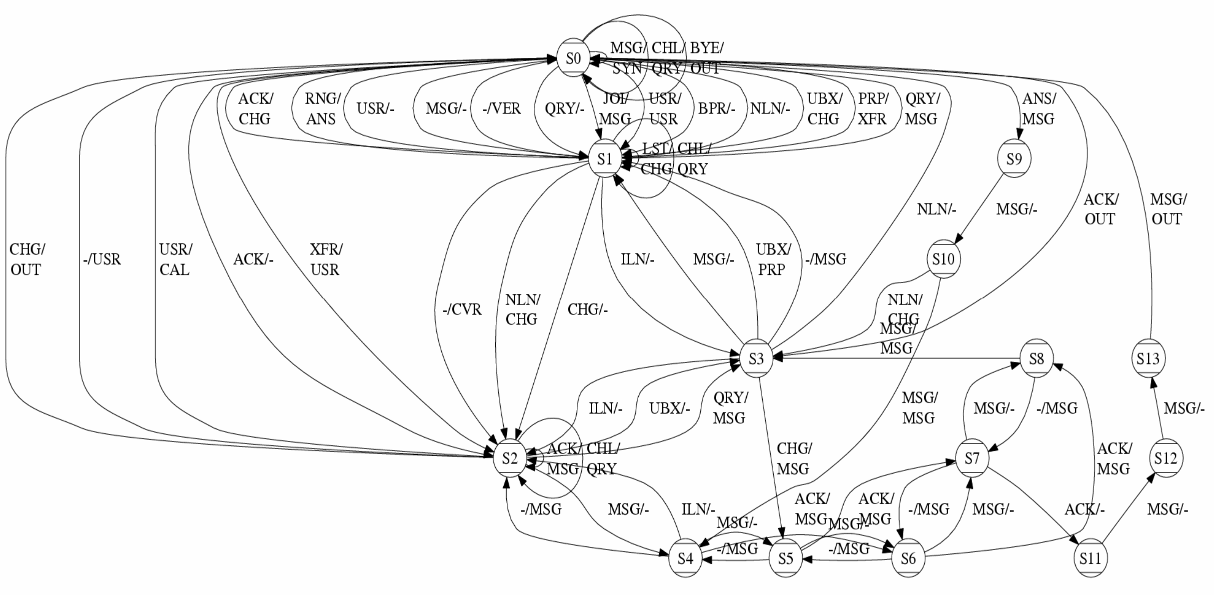
Experiment Setup

- Develop a special SOCKS proxy
  - Take over the input and output of the MSN client
  - Intercept incoming and outgoing traffics
- Message abstraction
  - Map a message to its type
- Fuzz operators
  - Data field fuzzing
  - Message type fuzzing
  - Intra-session message reordering
  - Transition substitution
- GOAL
  - Check if an implementation crashes



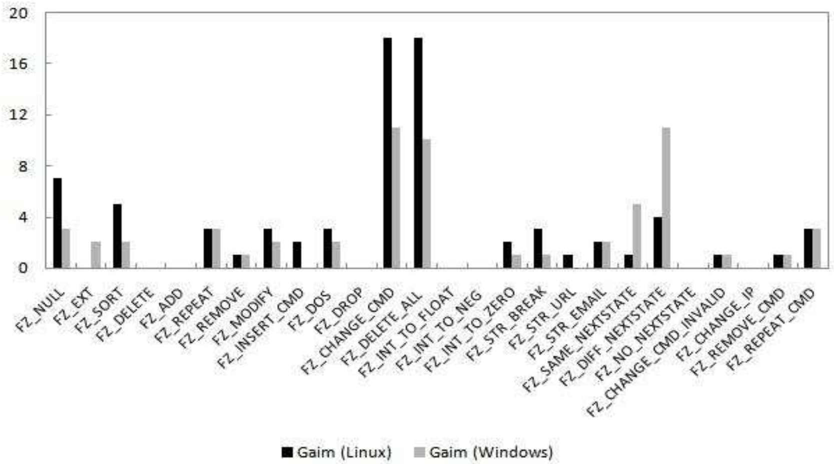
The Synthesized Model

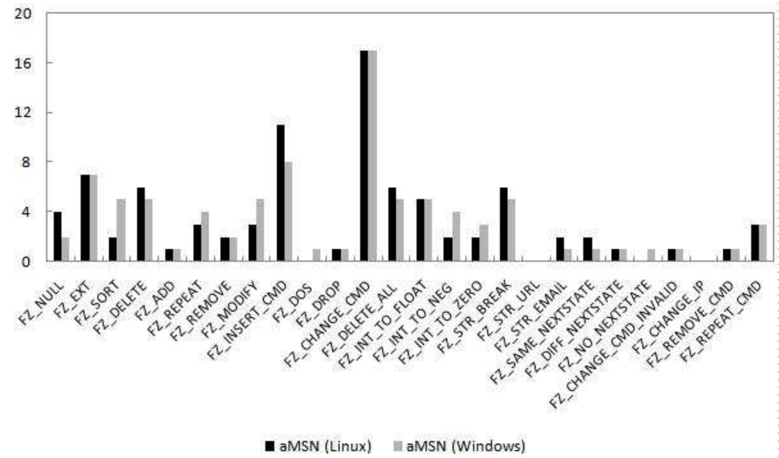
- State space reduction algorithm
  - Tree FSM: 98 states, 28 input symbols, and 14 output symbols
  - Reduced FSM: 14 states and 48 transition



Experiment Result

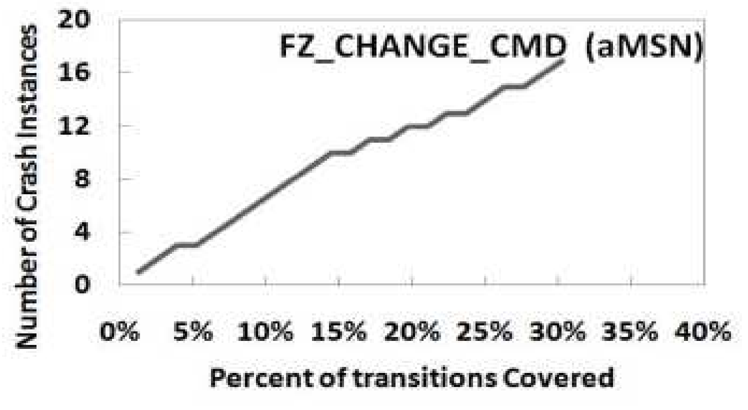
- The number of crash instances found by the fuzzing functions for each implementation

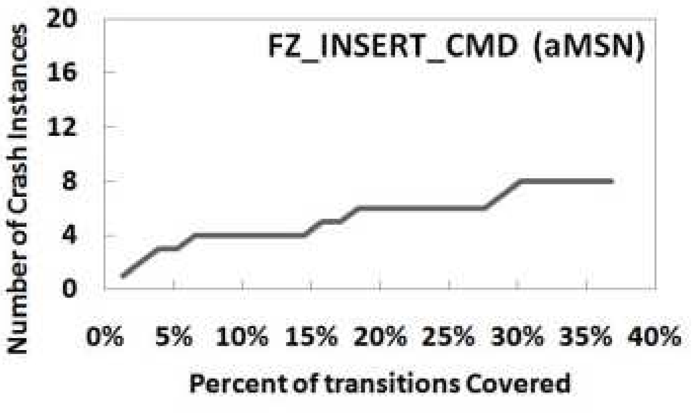




Experiment Result

- Number of crash instances found with regards to transition coverage metric





Experiment Result

- Transition coverage vs. syntax-based coverage
  - Experiment guided by transition coverage found more crash instances
    - A message type may correspond to multiple transitions
  - Behavioral model improves the measurability and coverage of black-box testing

Summary & Future Work

- We propose a new method
  - Uses a formally synthesized protocol model
  - Provides high coverage, measurability and automation
  - Promising result from experiments with real application
- A lot remains to be done
  - Refine the synthesized model using test output
  - Correlate flaws find with bugs in source code
  - Application of protocol synthesis method to other problems

- Thanks!

Contact info.

hsuya@cse.ohio-state.edu

shug@cse.ohio-state.edu

lee@cse.ohio-state.edu