Toward Online Verification of Client Behavior in Distributed Applications

Robby Cochran
Mike Reiter

University of North Carolina at Chapel Hill

NDSS 2013
Is this message from an unmodified client?
Goals

- Detect if client messages are consistent with the sanctioned client software …
  - Adversary might modify binary/memory
  - or rewrite message on the wire
- … Much more quickly than previous work
- Ideally we would do so online…
[Giffin et al. 2002] [Guha et al. 2009]

CONTROL FLOW MODEL OF CLIENT
[Vikram et al. 2009]
[Bethea et al. 2010]
Introduction

- Existing techniques to verify client behavior
  - Imprecise
  - Increase bandwidth usage
  - Computationally expensive

- Our method
  - Precise: no false negatives and no false positives
  - No additional bandwidth required
  - Validates most legitimate behavior faster than previous techniques
Overview

- Introduction
- Symbolic Execution
- Key Insight #1: Common Case Optimization
- Key Insight #2: Guided Search with History
- Case Studies
- Conclusion
A way of deriving the effects of a given program on a given system

- Constraints on input are constructed based on each execution path

Built on top of KLEE [Cadar et al. 2008]
Symbolic Environment

How can we use symbolic execution to verify a message?

```
1 loc = 0
2 while true do
3   key = symbolicReadKey()
4   if key == ESC then
5     sendQuitMsg()
6   else if key == UP then
7     loc = loc + 1
8   else if key == DN then
9     loc = loc - 1
10  end if
11  sendMsg(loc)
12 end while
```
Symbolic Environment

Symbolic State

\[ \text{loc} = 0 \land \text{key} = \text{ESC} \]

Execution Prefix

1,2,3,4,5

1. \text{loc} = 0
2. \textbf{while} true \textbf{do}
   3. \text{key} = \text{symbolicReadKey()}
   4. \textbf{if} \text{key} == \text{ESC} \textbf{then}
      5. \text{sendQuitMsg()}
   6. \textbf{else if} \text{key} == \text{UP} \textbf{then}
      7. \text{loc} = \text{loc} + 1
   8. \textbf{else if} \text{key} == \text{DN} \textbf{then}
      9. \text{loc} = \text{loc} - 1
   10. \textbf{end if}
   11. \text{sendMsg(loc)}
12. \textbf{end while}
Symbolic Environment

1. loc = 0
2. while true do
3.   key = symbolicReadKey()
4.   if key == ESC then
5.     sendQuitMsg()
6.   else if key == UP then
7.     loc = loc + 1
8.   else if key == DN then
9.     loc = loc - 1
10. end if
11. sendMsg(loc)
12. end while

Symbolic State

- loc == 0 \(\land\) key == ESC
- loc == 1 \(\land\) key != ESC \(\land\) key == UP

Execution Prefix

- 1,2,3,4,5
- 1,2,3,4,6,7,11
Symbolic Environment

Symbolic State

<table>
<thead>
<tr>
<th>loc == 0  ∧  key == ESC</th>
</tr>
</thead>
<tbody>
<tr>
<td>loc = 0</td>
</tr>
<tr>
<td>key = symbolicReadKey()</td>
</tr>
<tr>
<td>sendQuitMsg()</td>
</tr>
<tr>
<td>else if key == UP then</td>
</tr>
<tr>
<td>loc = loc + 1</td>
</tr>
<tr>
<td>else if key == DN then</td>
</tr>
<tr>
<td>loc = loc - 1</td>
</tr>
<tr>
<td>end if</td>
</tr>
<tr>
<td>sendMsg(loc)</td>
</tr>
<tr>
<td>end while</td>
</tr>
</tbody>
</table>

Execution Prefix

<table>
<thead>
<tr>
<th>1,2,3,4,5</th>
</tr>
</thead>
<tbody>
<tr>
<td>loc == 1  ∧  key != ESC  ∧  key == UP</td>
</tr>
<tr>
<td>1,2,3,4,6,7,11</td>
</tr>
<tr>
<td>loc == -1  ∧  key != ESC  ∧  key == DN</td>
</tr>
<tr>
<td>1,2,3,4,6,8,9,11</td>
</tr>
</tbody>
</table>
Client Message: \(msg_0\)

\[\text{<location>1</location>}\]

**STP**

Satisfiability Modulo Theory Solver

[ Ganesh et al. 2007 ]

---

**Symbolic State**

- \(\text{loc} = 0 \land \text{key} = \text{ESC}\)
- \(\text{loc} = 1 \land \text{key} \neq \text{ESC} \land \text{key} = \text{UP}\)
- \(\text{loc} = -1 \land \text{key} \neq \text{ESC} \land \text{key} = \text{DN}\)

**Execution Prefix**

- \(1,2,3,4,5\)
- \(1,2,3,4,6,7,11\)
- \(1,2,3,4,6,8,9,11\)

---

**Check consistency of message with formulas generated via symbolic execution.**
Verifying Client Messages

Iteratively find execution prefix $\prod$ consistent with $\text{msg}_0, \text{msg}_1, \ldots, \text{msg}_n$
Key Insight #1: Optimize for the common case of a legitimate client.
Verification: Node Selection

Key Insight #2:
Correlate message contents with previously executed paths

- Build training corpus of “execution fragments”
- Select next node to explore using training data
Execute client software to generate a set of message traces.

Split each trace into a set of execution fragments $\pi_{i,j}$

Associate each $\pi_{i,j}$ with the set of messages it is consistent with.
Cluster execution fragments $\pi$ by edit distance using k-medoids clustering.

Cluster messages by edit distance using k-medoids clustering.
Using the Clusters

Map message medoids to execution fragment medoids.

Find message medoid closest to $msg_n$ via edit distance.

Use associated execution fragment medoids to guide search.
Networked games are popular!
- Some > 10 million subscribers
- Gaming industry revenues of $67 billion (2012)

Cheaters reduce the quality of the game for honest players
- Our technique detects any cheat that requires a modification to the client binary or memory
Case Studies

- **XPilot**
  - Open-source multiplayer 2D shooter
  - 150000 lines of C
  - Little client-side state

- **TetriNET**
  - Text-based multiplayer Tetris game
  - 5000 lines of C
  - More ambiguity at server about client state
XPilot Results

- 40-fold cross validation
  - 4000 points per bin
  - Average: 32 messages/second

- Average verification time < 100ms

- Average delay at end of queue is less than 5 min

- 100x faster than previous work
TetriNET Results

- 20-fold cross validation
  - Average game 6.5 minutes
  - $20 \times 20 = 400$ points per bin
- Average verification time < 100ms
- Average Delay at end of queue is less than 2 min
- See paper for a variation that often keeps up with gameplay with a small increase in bandwidth
 Conclusion

— Key Insights
  • Optimize for the common case legitimate client
  • Use a search heuristic that correlates message contents with previously executed paths
  • Optimize symbolic execution components for our specific needs (see paper)

— Contribution: Precise client checking algorithm
  • Dramatically improves performance over previous work with similar design goals
  • In some cases, verification comes very close to keeping pace with the application (see paper)
Questions?