CS 6V81-05
Server Side Verification of Client Behavior in Online Games

Nathan McDaniel
Department of Computer Science
The University of Texas at Dallas

November 4th, 2011
Outline

1. Symbolic Execution
2. Constraints
   - Generation
   - Accumulation
   - Pruning
3. Scalability
   - Server
   - Client
4. Experiment
   - XPilot
   - Cap-Man
   - Results
Authoritative State is the amount of control or weight carried by a server or client. Used for the ultimate goal of protecting against client misbehavior.
Client-side Behavior Verification

- Constructs model of proper client execution using source code
Client-side Behavior Verification

- Constructs model of proper client execution using source code
- Exploits event loop process
Client-side Behavior Verification

- Constructs model of proper client execution using source code
- Exploits event loop process
- Symbolic execution
Client-side Behavior Verification

- Constructs model of proper client execution using source code
- Exploits event loop process
- Symbolic execution
- Reasons if client input is possible
Another Approach

Audit Server

- Probabilistic
- Maintains committed state
- Client periodically sends hash
- Server challenge results in full state to be transmitted
Why ours is Better

Symbolic Execution

- Generate vulnerability signatures
- Generate inputs that will induce error conditions
- Automating mimicry attacks
- Optimizing privacy-preserving computations
- many more...
Outline

1 Symbolic Execution

2 Constraints
   • Generation
   • Accumulation
   • Pruning

3 Scalability
   • Server
   • Client

4 Experiment
   • XPilot
   • Cap-Man
   • Results
What is Symbolic Execution?

**Symbolic Execution** is the exploration of all path during the execution of a program.

*i.e. to find bugs within programs*
Initialization

Initial inputs are allowed to be "anything" and their memory locations are marked as symbolic. Conditional branches including a symbolic variable will create constraints and fork the execution.
Symbolic Values and Constraints

\[ a \leftarrow b + c \]  
When \( a, b \) and \( c \) are marked as symbolic a logical constraint is added on \( a \) that it should be equal to the sum of \( b \) and \( c \)
Every possible path has a set of constraints that must hold true along that path of execution. A constraint solver (such as KLEE) then provides concrete values for bad input.
Outline

1 Symbolic Execution

2 Constraints
   - Generation
   - Accumulation
   - Pruning

3 Scalability
   - Server
   - Client

4 Experiment
   - XPilot
   - Cap-Man
   - Results
Simple Example

100: \texttt{loc} \leftarrow 0;
101: 
102: \textbf{while} \texttt{true} \textbf{do}
103: \hspace{1em} \texttt{key} \leftarrow \texttt{readkey}();
104: \hspace{1em} \textbf{if} \texttt{key} = \texttt{ESC} \textbf{then}
105: \hspace{2em} \texttt{endgame}();
106: \hspace{1em} \textbf{else if} \texttt{key} = \texttt{‘\uparrow’} \textbf{then}
107: \hspace{2em} \texttt{loc} \leftarrow \texttt{loc} + 1;
108: \hspace{1em} \textbf{else if} \texttt{key} = \texttt{‘\downarrow’} \textbf{then}
109: \hspace{2em} \texttt{loc} \leftarrow \texttt{loc} - 1;
110: \hspace{1em} \textbf{end if}
111: \hspace{1em} \texttt{sendlocation}(\texttt{loc});
112: \hspace{1em} \textbf{end while}

200: \texttt{prev\_loc} \leftarrow \texttt{symbolic};
201: \texttt{loc} \leftarrow \texttt{prev\_loc};
202: \textbf{while} \texttt{true} \textbf{do}
203: \hspace{1em} \texttt{key} \leftarrow \texttt{symbolic};
204: \hspace{1em} \textbf{if} \texttt{key} = \texttt{ESC} \textbf{then}
205: \hspace{2em} \texttt{endgame}();
206: \hspace{1em} \textbf{else if} \texttt{key} = \texttt{‘\uparrow’} \textbf{then}
207: \hspace{2em} \texttt{loc} \leftarrow \texttt{loc} + 1;
208: \hspace{1em} \textbf{else if} \texttt{key} = \texttt{‘\downarrow’} \textbf{then}
209: \hspace{2em} \texttt{loc} \leftarrow \texttt{loc} - 1;
210: \hspace{1em} \textbf{end if}
211: \hspace{1em} \texttt{breakpoint};
212: \hspace{1em} \textbf{end while}

(a) A toy game client . . .
(b) . . . instrumented to run symbolically
Round Constraint

\[(loc = \text{prev\_loc} + 1) \lor (loc = \text{prev\_loc} - 1) \lor (loc = \text{prev\_loc})\]
Outline

1. Symbolic Execution
2. Constraints
   - Generation
   - Accumulation
   - Pruning
3. Scalability
   - Server
   - Client
4. Experiment
   - XPilot
   - Cap-Man
   - Results
An Accumulated Constraint is a conjunction of round constraints that represents a sequence of possible paths taken over multiple client loops.
Accumulation Example

\[ g(j) = \begin{cases} 
    loc_j = loc_{j-1} + 1, \\
    loc_j = loc_{j-1} - 1, \\
    loc_j = loc_{j-1} 
\end{cases} \]
Accumulation Example

\[ loc_2 = 9 \land loc_2 = loc_1 + 1 \]
\[ loc_2 = 9 \land loc_2 = loc_1 - 1 \]
\[ loc_2 = 9 \land loc_2 = loc_1 \]
Accumulation Example

\[ loc_1 = 8 \land [loc_2 = 9 \land loc_2 = loc_1 + 1] \]
\[ loc_1 = 8 \land [loc_2 = 9 \land loc_2 = loc_1 - 1] \]
\[ loc_1 = 8 \land [loc_2 = 9 \land loc_2 = loc_1] \]
Accumulation Example

\[
C_2 = \{loc_1 = 8 \land [loc_2 = 9 \land loc_2 = loc_1 + 1]\}
= \{loc_1 = 8 \land loc_2 = 9\}
\]
Outline

1. Symbolic Execution
2. Constraints
   - Generation
   - Accumulation
   - Pruning
3. Scalability
   - Server
   - Client
4. Experiment
   - XPilot
   - Cap-Man
   - Results
Pruning

Removes duplicate and insignificant constraints from round constraints before accumulation.
Outline

1. Symbolic Execution
2. Constraints
   - Generation
   - Accumulation
   - Pruning
3. Scalability
   - Server
   - Client
4. Experiment
   - XPilot
   - Cap-Man
   - Results
Outline

1. Symbolic Execution

2. Constraints
   - Generation
   - Accumulation
   - Pruning

3. Scalability
   - Server
   - Client

4. Experiment
   - XPilot
   - Cap-Man
   - Results
Round constraints also result from server messages sent to the client.

1. Eager
2. Lazy
Eager Round Constraints

Every server message results in a constraint regardless of whether the client processes it.

- decoupled from verification and pre-computed
Lazy Round Constraints

Only client processed server messages result in constraints for the round.

- tightly coupled and model must be built by accessing client logs for viable server messages
1. Symbolic Execution

2. Constraints
   - Generation
   - Accumulation
   - Pruning

3. Scalability
   - Server
   - Client

4. Experiment
   - XPilot
   - Cap-Man
   - Results
Neither eager or lazy approach is fast enough to be implemented on the critical path for thousands of users.
Log Everything

Game operators already perform detailed logging. A log structure provides readability for GMs and allows for offline verification.
Outline

1 Symbolic Execution

2 Constraints
   • Generation
   • Accumulation
   • Pruning

3 Scalability
   • Server
   • Client

4 Experiment
   • XPilot
   • Cap-Man
   • Results
1 Symbolic Execution
2 Constraints
   - Generation
   - Accumulation
   - Pruning
3 Scalability
   - Server
   - Client
4 Experiment
   - XPilot
   - Cap-Man
   - Results
The Game

- 150,000 lines of C code
- Similar to Asteroids
  1. multi-player modes (death match, CTF, racing)
  2. better physics (fuel weight, acceleration)
- latest port in 2009 to iPhone
Client Cheating

Independent keystrokes should never result in KEYFIREDSHOT and KEYSHIELD being sent to the server simultaneously.
Modifications

- add UDP acknowledgment
- add floating point constraint library
- remove trivial client information
  1. "reliable" UDP information
  2. graphical user modification
  3. external graphics references
Lazy Verification

2000 rounds or about 1min of gameplay

90% avg=26ms 5% avg=3.4s 5% avg=14.9s
Eager Manual Tuning

\( (p = \text{PKT\_START}) \land (\text{constraints\_for(PKT\_START)}) \)
\( (p = \text{PKT\_FUEL}) \land (\text{constraints\_for(PKT\_FUEL)}) \)
\( (p = \text{PKT\_TIME\_LEFT}) \land (\text{constraints\_for(PKT\_TIME\_LEFT)}) \)
\( (p = \text{PKT\_END}) \land (\text{constraints\_for(PKT\_END)}) \)

multiple breaks in frame transmission and user keybind configuration.
Eager Verification

2000 rounds or about 1 min of gameplay

90% avg=1.6s 10% avg=11.4s
The Game

- gives client more authoritative state
- about 1000 lines of C code
- simple Pac-Man clone
With the client having most of the authoritative state it can simply tell the server an updated coord of movement regardless of making a valid move.
Cap-Man Verification

2000 rounds or about 7min of gameplay

814ms 260ms
1 Symbolic Execution

2 Constraints
   • Generation
   • Accumulation
   • Pruning

3 Scalability
   • Server
   • Client

4 Experiment
   • XPilot
   • Cap-Man
   • Results
This method of server side verification allows increased security with less bandwidth, regardless of how much authoritative state the server maintains.
Questions?