ARE: A System for Automated Reverse Engineering

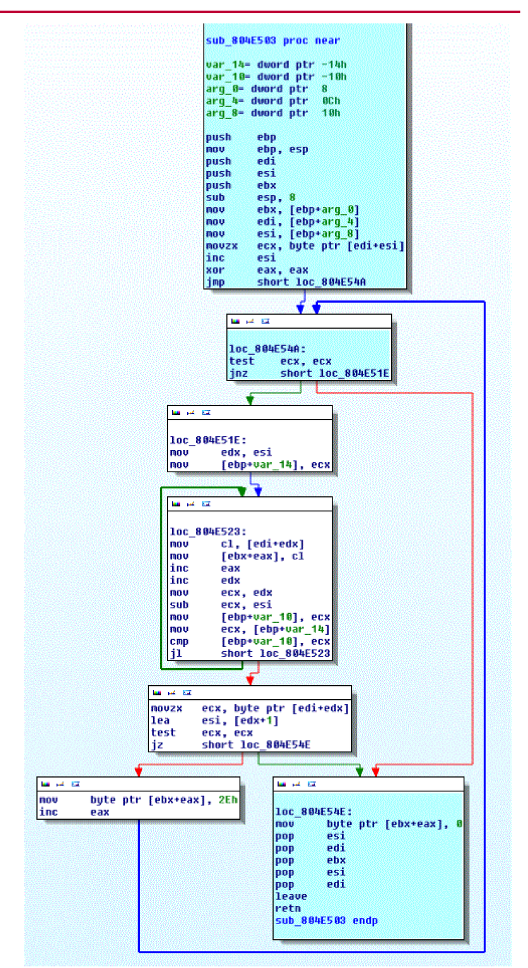
HCSS 2013

Robert Ross (robert.b.ross@baesystems.com)
Cyber Operations and Networking Group (CONG)

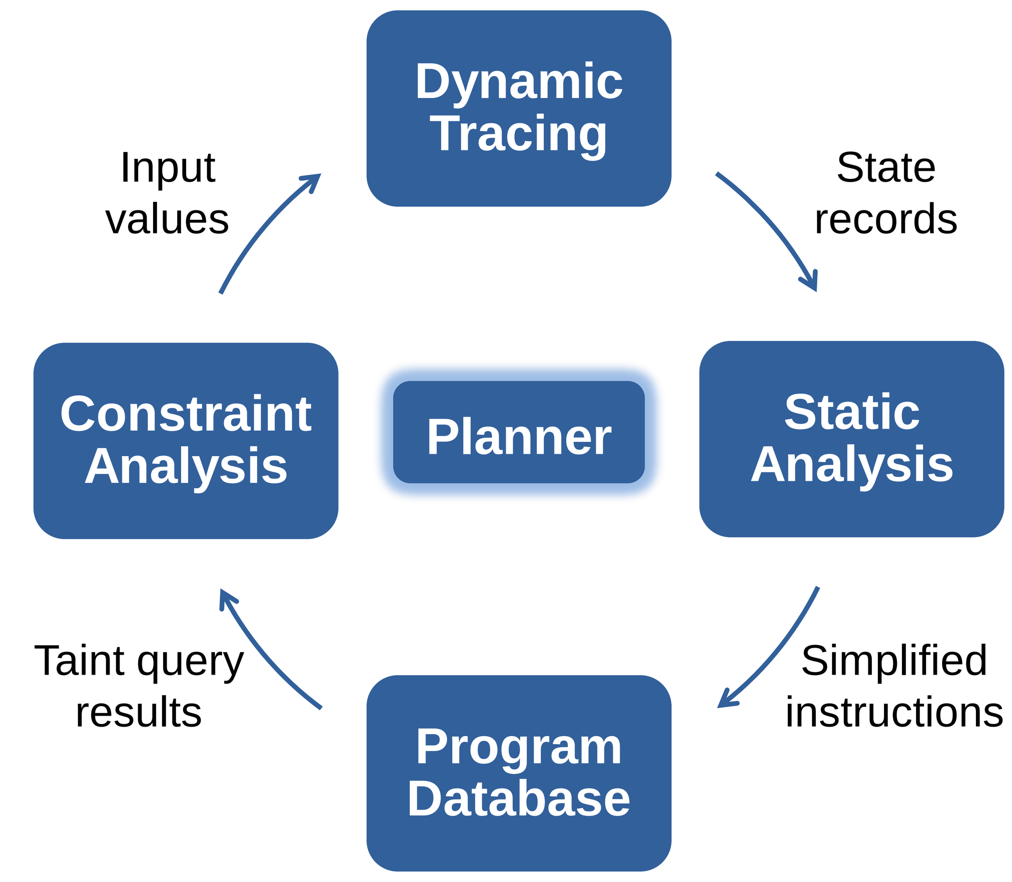
BAE SYSTEMS

Introduction

- **Goal:** Enable users or semi-automated planners to iteratively negate branches and fabricate paths to reach areas of interest, explore unvisited blocks, and test code units without the benefit of source code
- **Applications:** Program analysis (e.g., how are sockets used?), verification (e.g., are quality objectives still satisfied?), and optimization (e.g., are bounds exceeded?)
- **Note:** Analysts may further restrict the set of acceptable solutions, e.g., "all but the last byte of an array must be in `[0x20, 0x7E]`"



Approach



- **Dynamic Tracing**
  - Instrumentation, e.g., Pin
  - Emulation, e.g., TEMU
- **Static Analysis**
  - Third-party, e.g., REIL
  - First-party, e.g., PREIL
- **Program Database**
  - Relational, e.g., MySQL
  - NoSQL, e.g., HBase
- **Constraint Analysis**
  - Third-party, e.g., Vine
  - First-party, e.g., COMET

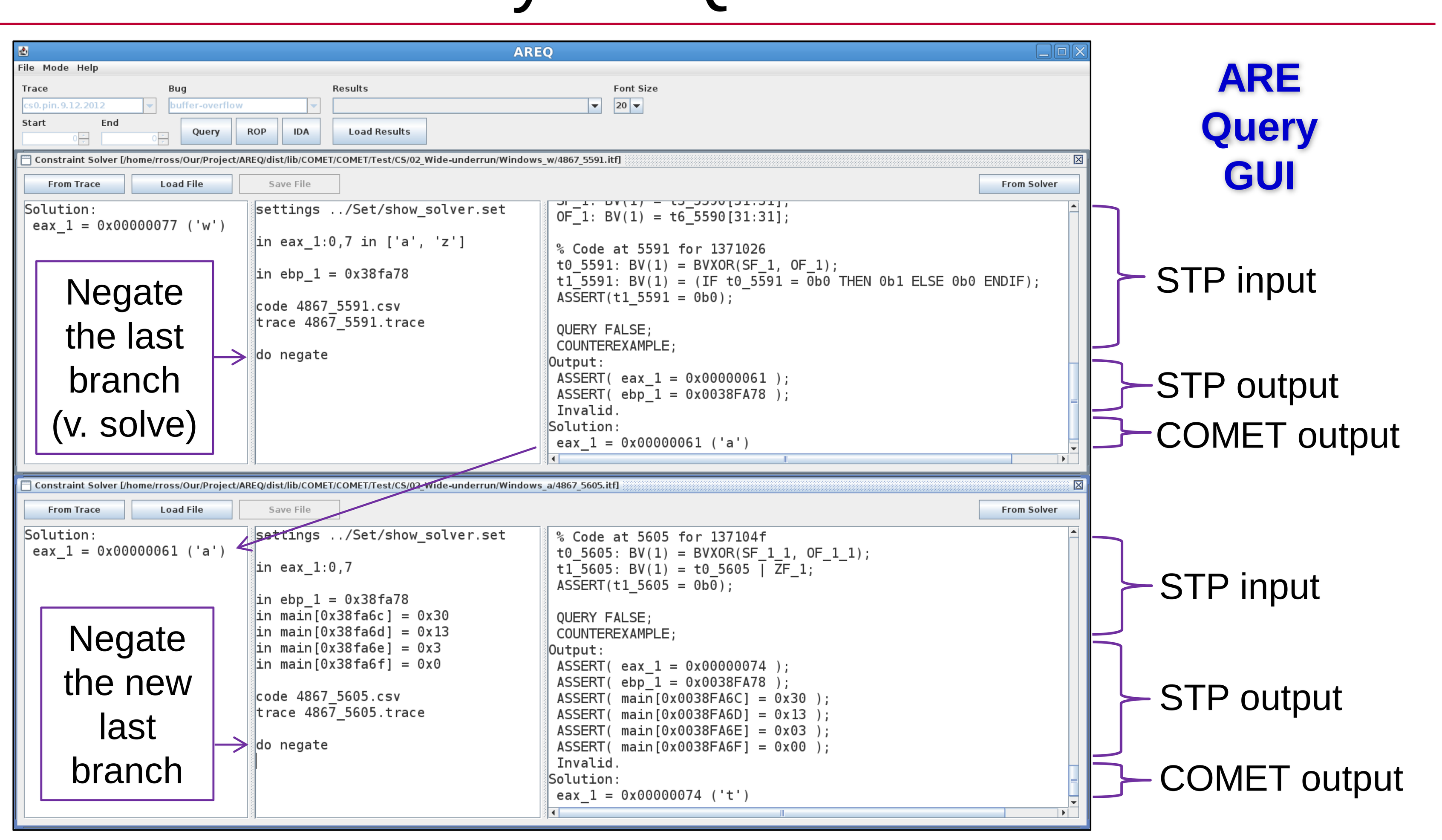
Static Analysis: Representations

- **REIL:** Reverse Engineering Intermediate Language
  - **Arithmetic:** ADD, SUB, MUL, DIV, MOD, BSH (binary shift)
  - **Bitwise:** AND, OR, XOR (can derive "NOT" from XOR)
  - **Conditional:** BISZ (Boolean is-zero), JCC (jump conditional)
  - **Data transfer:** LDM (load), STM (store), STR (store to register)
  - **Other:** UNDEF (undefined), UNKN (unknown), NOP (no-op)
- **PREIL:** Power-REIL (more precise, faster, and clearer)
  - **Arithmetic:** LSH (left shift) and RSH (right shift) instead of BSH
  - **Bitwise:** Same as REIL (but allows bit ranges, resizing, etc.)
  - **Conditional:** Adds IFM (conditional STM), IFR (conditional STR)
  - **Data transfer:** Same as REIL (but allows multiple memories, etc.)
  - **Other:** Same as REIL (but allows labels, macros, etc.)

Constraint Analysis: Inputs

- **Trace:** {(seq, ip, tid)}
  - **seq:** Sequence number (optional; for reference to full, uncut trace)
  - **ip:** Instruction pointer (raw bytes and disassembly is in full trace)
  - **tid:** Thread identifier (pid and values read/written are in full trace)
- **Code:** {(ip, size, list)}
  - **size:** Machine's instruction size (for whether branches were taken)
  - **list:** List of PREIL instructions (for a single machine instruction)
- **Patch:** {(seq, it, val)}
  - **it:** Target (i.e., "`<register>_<tid>`" or "`<memory>[<address>]`")
  - **val:** Value assigned to **it** before **seq** (for partial observability)
- **Others:** Input constraints, output constraints, and settings

Constraint Analysis: Queries



ARE Query GUI

Constraint Analysis: Components

- **COMET:** Constraint Optimization, Management, Extensions, and Translations
  - **Constraint:** Weakest preconditions for a given path
  - **Optimization:** Reduce complexity of the constraint program
  - **Management:** Services, e.g., for joining subproblems
  - **Extensions:** Additional constraints, e.g., around interesting code
  - **Translations:** Various SMT solvers, e.g., STP and Boolector
- **Optimization:** Cutting out unnecessary constraints
  - **SLICE:** Statically Limited Irrelevant Constraint Elimination
    - **Example:** Remove PREIL for unused flags during preprocessing
  - **DICE:** Dynamic Irrelevant Constraint Elimination (path specific)
  - **TMF:** Taint Modeling Function (for Input-Output Relationships)

TMF Options and Results

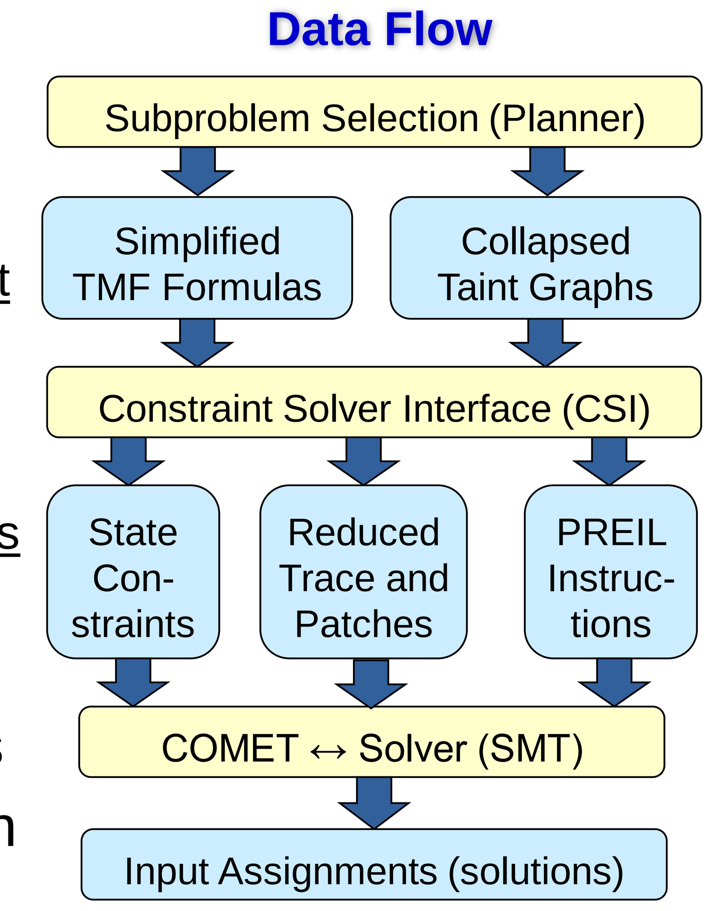
1. Temporary variable for each operation
   - **Advantage:** State of the art DICE yet easy to read and understand
2. Single expression for each branch variable
   - **Example:** `b1 = (!(!((0xffffffd0 + eax_1)` $-_{64}$ `0x9))) &`
     `(!((((0xffffffd0 + eax_1)` $-_{64}$ `0x9) & 0x100000000) >> 0x20)) = 0`
   - **Advantage:** Maximal flexibility for constraint solvers' optimizers
3. Temporaries from common subexpression elimination
   - **Example:** `t1 = (0xffffffd0 + eax_1)` $-_{64}$ `0x9;`
     `b1 = (!(!t1)) & (!((t1 & 0x100000000) >> 0x20)) = 0`
   - **Advantage:** Reduces execution time for solvers with weak optimizers

- **Result:** Order of magnitude size reduction for each problem
  - **Advantage:** Enables each constraint problem to cover a longer path

Summary and Conclusions

- **Target:** Programs without source
  - **Initial state:** A known execution path approaches yet avoids a dangerous block
  - **Static analysis:** Helps determine that the block is a relatively nearby area of interest
  - **Dynamic analysis:** Suggests paths through the area that may be feasible
  - **Constraint analysis:** Provides inputs for feasible paths or recognizes impossibilities
- **Benefit:** Directed search avoids reevaluation of known paths and the high cost of tempting yet futile tracks
- **Conclusion:** Combined analysis can effectively handle binary code paths

Data Flow



This material is based upon work supported by DARPA and AFRL under contract FA8750-12-C-0097
Approved for public release; distribution unlimited. Cleared for open publication on 4/23/2013
ARE was developed jointly, e.g., with Vu Le and Greg Sadosuk of BAE Systems CONG