Z3strBV: A Solver for a Theory of Strings and Bit-vectors

Murphy Berzish\textsuperscript{1}, Sanu Subramanian\textsuperscript{2}, Yunhui Zheng\textsuperscript{3}, Omer Tripp\textsuperscript{4}, and Vijay Ganesh\textsuperscript{1}

(1) University of Waterloo (2) Intel Security (3) IBM Research (4) Google

July 1, 2016
SMT Workshop
Outline

● Background
● Existing solutions, motivation
● Decidability of strings+bit-vectors
● Design of Z3strBV
● Binary search heuristics
● Library-aware SMT solving
● Experimental evaluation
● Future work
● Summary and conclusion
Background: Symbolic Execution and SMT

- Analysis of low-level programs in C/C++
- Powerful application of SMT solvers for
  - Detection of security vulnerabilities
  - Automated test case generation
- Strength of the symbolic execution engine related to expressive power, efficiency of the SMT solver backend
Why Not String-Integer Combination?

- Existing SMT solvers supporting theory of strings interpret the length of a string as an arbitrary-precision integer.
- In languages like C/C++, integer values (e.g. `strlen`) are fixed-precision.
- Relevant: semantics of overflow/underflow.
- More efficient to model as a bit-vector and not an integer.
Why Not Represent Strings as Bit-vectors?

- Strings can also be represented as (arrays of) bit-vectors
- KLEE, S2E both do this
- Performance issue: low-level bit-vector representation vs. high-level semantics of the string type
  - Path explosion: `strlen` on a symbolic string of length $N$ forks $N+1$ paths.
- Difficulty in handling unbounded / arbitrary-length strings
Motivation for a String+Bit-vector Combination

- In summary, the problems with existing solutions are:
  - Strings + natural numbers has limited ability to model overflow, underflow, bit-wise operations, pointer casting, etc. without bit-vectors
  - Bit-vector solvers are not able to perform direct reasoning on strings efficiently, and cannot handle unbounded strings

- This motivates us to build Z3strBV, a solver for strings + bit-vectors.
  - Combination of a string solver (Z3str2), bit-vector solver (Z3’s BV theory), bit-vector sorted length function (on top of Z3str2), and SMT solver framework (Z3)
  - Opportunity to apply new heuristics:
    - Binary search
    - Library-aware SMT solving
Contributions

● Solver for quantifier-free theory of strings, bit-vectors, and bit-vector-sorted string length
  ○ Built on top of the Z3str2 string solver (Zheng et al., 2015)
    ■ ...which is itself built on top of the Z3 SMT solver (de Moura, Bjorner, et al., 2008)
  ○ Extensions for bit-vector sorts, in particular \( \text{strlen}_{bv} : \text{String} \rightarrow \text{Bitvector} \)

● New solver heuristics:
  ○ Binary search pruning strategy to reach consistent length assignments
  ○ Library-aware SMT solving for improved performance

● Decidability of string+bit-vector combination
Motivating Example

```c
bool check_login(char *username, char *password) {
    if (!validate_password(password)) {
        invalid_login_attempt(); exit(-1);
    }
    const char *salt = get_salt8(username);
    uint16_t len = strlen(password) + strlen(salt) + 1;
    if (len > 32) { invalid_login_attempt(); exit(-1); }
    char *saltedpw = (char*)malloc(len);
    strcpy(saltedpw, password);
    strcpy(saltedpw, salt);
    ...
}
```
Decidability of String + Bit-Vector Combination

- The satisfiability problem for the QF theory of word equations, bit-vector length, and bit-vector terms is decidable.
- Proof sketch: by reduction to strings + regular language membership
  - Shown decidable by Schulz (1992)
- This may seem trivial -- finitely many BVs implies finitely many strings?
  - NO! Overflow semantics apply to length terms too
- Decidability is in fact non-trivial as infinitely many strings must be considered
Design Overview

- Word equation solving
- Integration of string and bit-vector theory
- Binary search heuristic for search-space pruning
String Equation Solving

- Key technique of Z3str2: recursively split equations into subproblems until the system can be solved directly
- Given an equation, identify all possible splits / “arrangements”
String Equation Solving

- Given an arrangement, generate a set of sub-equations over smaller strings

\[ X \cdot Y = M \cdot N \]

\[ M = X \cdot T \quad Y = T \cdot N \]

Equations of smaller strings
String Equation Solving

- Given an arrangement, generate a set of sub-equations over smaller strings
- New equations are split recursively until all equations are between variables and string constants
String-Bitvector Theory Integration

- Three main rules:
  - Each character has length 1, the empty string has length 0
  - $X = Y \Rightarrow \text{strlen}_{bv}(X) = \text{strlen}_{bv}(Y)$
  - $W = X \cdot Y \cdot Z \ldots \Rightarrow \text{strlen}_{bv}(W) = \text{strlen}_{bv}(X) + \text{strlen}_{bv}(Y) + \text{strlen}_{bv}(Z) + \ldots$

- These are, elegantly, of similar form to the rules for string-integer integration
- Overflow semantics handled by bit-vector theory solver
Binary Search Heuristic

- **Z3str2** performs (naive) linear search for the length of variables
  - Constraints of the form “len(X) > 15000” are checked starting at “len(X) = 0, 1, 2, 3, …”

- **Z3strBV** performs binary search over bit-vector lengths
  - e.g. searching for a 2-bit length L: midpoint is 2, branch on len(X) < 2, len(X) = 2, len(X) > 2
  - If strings are longer than the upper bound, overflow semantics come into play
  - Consistent lengths found in significantly less time
  - This is sound and very efficient

- Similar technique back-ported to the integer version
  - Main difference: no *a priori* fixed upper bound for integers
  - Choose a “floating” upper bound that the solver can choose to increase if necessary
Library-Aware SMT Solving

- Provide native solver support for library functions that are:
  - Available in popular programming languages like C/C++
  - Very commonly used by programmers
  - A frequent source of errors due to programmer mistakes
  - Expensive to analyze symbolically due to large number of potential paths

- Extend the logic of traditional SMT solvers with declarative summaries of functions such as `strlen`, `strcpy`, etc.

- Preliminary work with Z3strBV to support these functions
Experimental Results

- We evaluated our solver on 7 real buffer overflow vulnerabilities:
  - CVE-2015-3824: Google stagefright 'tx3g' MP4 atom integer overflow
  - CVE-2015-3826: Google stagefright 3GPP metadata buffer overread
  - CVE-2009-0585:libsoup integer overflow
  - CVE-2009-2463: Mozilla Firefox/Thunderbird Base64 integer overflow
  - CVE-2002-0639: Integer and heap overflows in OpenSSH 3.3
  - CVE-2005-0180: Linux kernel SCSI IOCTL integer overflow
  - FreeBSD wpa supplicant(8) Base64 integer overflow

- Handcrafted constraints for vulnerable region
- String+bit-vector generated a model for all instances
- String+integer could not solve any instances
Experimental Results

- Evaluation of library-aware SMT solving via comparison with KLEE
- Input constraints from the motivating example (check_login)
- The size of the length variable determines the total number of paths
- We consider 8-bit and 16-bit length variables
  - KLEE times out after 120 minutes with a 16-bit length
  - Z3strBV finds the bug in 0.27 seconds
- The path constraints are not hard; there are just too many paths
Experimental Results

- Binary search heuristic applied to unconstrained string variables
- Implemented a modified Z3strBV that uses linear search
- Significant gain in performance when binary search is used

<table>
<thead>
<tr>
<th>Z3strBV</th>
<th>SAT</th>
<th></th>
<th></th>
<th></th>
<th>TIMEOUT (20s)</th>
<th></th>
<th></th>
<th></th>
<th>UNKNOWNE</th>
<th>Total</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>T_{min}</td>
<td>T_{avg}</td>
<td>T_{max}</td>
<td>#</td>
<td>T_{min}</td>
<td>T_{avg}</td>
<td>T_{max}</td>
<td>#</td>
<td>T_{min}</td>
<td>T_{avg}</td>
</tr>
<tr>
<td>Binary Search</td>
<td>98</td>
<td>0.060</td>
<td>0.172</td>
<td>2.667</td>
<td>9</td>
<td>0.047</td>
<td>0.081</td>
<td>0.320</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Linear Search</td>
<td>72</td>
<td>0.060</td>
<td>0.097</td>
<td>0.618</td>
<td>9</td>
<td>0.061</td>
<td>0.111</td>
<td>0.415</td>
<td>27</td>
<td>20.000</td>
<td>20.000</td>
</tr>
</tbody>
</table>
Experimental Results

- Performance of binary search heuristic in the integer version (Z3str2)
- Compared against the previous (linear search) Z3str2, and CVC4
- Z3str2 with binary search is faster than both linear-search Z3str2 and CVC4

<table>
<thead>
<tr>
<th></th>
<th>Z3-str2 (Binary Search)</th>
<th>Z3-str2 (Naive Search)</th>
<th>CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>169</td>
<td>138</td>
<td>126</td>
</tr>
<tr>
<td>UNSAT</td>
<td>34</td>
<td>34</td>
<td>19</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Timeout</td>
<td>0</td>
<td>31</td>
<td>60</td>
</tr>
<tr>
<td>Tool reports error</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Crash</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total no. of benchmarks</td>
<td>205</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>Total time (sec)</td>
<td>41.697 (1x)</td>
<td>9569.639 (229x)</td>
<td>12014.893 (264x)</td>
</tr>
</tbody>
</table>
Future Work

- Tighter integration with symbolic execution engines
  - String + bit-vector in KLEE, S2E
  - String + integer into Jalangi
- Development of efficient function summaries for string functions in the standard libraries of several programming languages
- Integration of Z3str2 and Z3strBV into the main Z3 codebase
  - The port to the newest version of Z3 is now feature-complete and in testing.
Summary and Conclusion

• Motivation and design for a solver for strings + bit-vectors
  ○ String+integer less efficient than string+bit-vector for overflow/underflow
  ○ Bit-vector solvers are inefficient at modelling strings as arrays of bit-vectors

• Binary search heuristic for consistent length assignments
  ○ Useful for both bit-vector and integer length terms
  ○ Significant performance improvements vs. state-of-the-art solvers

• Library-aware SMT solving
  ○ Large performance improvements over traditional symbolic execution techniques
References