GrayC: Greybox Fuzzing of Compilers and Analysers for C

Karine Even-Mendoza², Arindam Sharma¹, Alastair F. Donaldson¹ and Cristian Cadar¹
Fuzzing

System Under Test
Compiler Fuzzing Categories

intel/yarpgen

GrayC

?
Greybox Fuzzing

- Successful for testing general software
  - Google: ~9k vulnerabilities and 28K bugs in 850 projects

- Not yet effective in compiler testing
  - Random, byte-level: High levels of invalidity
  - Tends to exercise the shallow (lexer, parser etc) parts of the compiler

- Attempts for static languages include keyword dictionaries, protobuf descriptions of language structure, regular expressions for common patterns
  - Still produce a high rate of invalid programs
  - Clang-Proto-Fuzzer: bugs are being fixed too slow (if at all)
  - No-fuss Compiler Fuzzing: code that crashes a C or C++ compiler, but that includes extensive undefined behaviour may well be ignored by developers.
GrayC

- Greybox fuzzing for testing compilers for C, a language with lots of UB

- Key idea: semantic-aware mutations
  - AST guidance
  - Modify individual programs or combine existing
  - A configurable level of aggressiveness
  - LibFuzzer: the underlying greybox fuzzing engine
Mutations

Mutators

• Duplicate Statement
• Delete Statement
• Expand Expressions
• Type Modifications

Recombiners

• Function Combination
• Function Body Replacement
• Code Fragment Addition
Recombiner Example

```c
int dest_func(int x_dest, int y_dest){
    int b_dest = x_dest + y_dest;
    b_dest = b_dest + 5;
    return b_dest;
}

int source_func(int j_src, int k_src) {
    int m_src = j_src * k_src;
    return m_src;
}
```

- Initialize variables corresponding to the src function to the args of dest function
- Interleave statements from src function
- Randomly select return from src or dest
## Evaluation in the Wild

<table>
<thead>
<tr>
<th></th>
<th>Front-end</th>
<th>Middle-/Back-end</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>LLVM</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MSVC</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Frama-C</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>8</strong></td>
<td><strong>22</strong></td>
</tr>
</tbody>
</table>
Bug: GCC (Middle-End) 11, 12 (Bugzilla: #103813)

```c
struct a d;
struct a {
    int b;
    int c[]
} main() { d.c[1]; } d.c[1]; }.c[1]; }
```
Controlled Experiments

1) GrayC Our tool
2) GrayC-No-Cov-Guidance Does coverage guidance matter?
3) GrayC-Fragments-Fuzzing Only code fragments injection, no coverage (similar to LangFuzz)
4) Clang-Fuzzer Greybox fuzzing with byte-level mutations
5) Csmith Generative, grammar-based fuzzing
6) Grammarinator Grammar-based fuzzing (ANTLR C grammar)
7) PolyGlot Language-agnostic AFL-based fuzzer, based on semantic error fixing
8) RegExpMutator LibFuzzer-based fuzzer based that uses regexp-based mutations

8 tools, 24h, 10 repetitions
GrayC is able to match the static validity rates of a generative fuzzer like Csmith.

<table>
<thead>
<tr>
<th></th>
<th>Programs/h</th>
<th>Statically-valid (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Csmith</strong></td>
<td>1,144</td>
<td>99.96%</td>
</tr>
<tr>
<td><strong>GrayC</strong></td>
<td>2,906</td>
<td>99.47%</td>
</tr>
<tr>
<td><strong>GrayC-Fragments-Fuzzing</strong></td>
<td>4,152</td>
<td>99.08%</td>
</tr>
<tr>
<td><strong>Clang-Fuzzer</strong></td>
<td>1,183</td>
<td>1.55%</td>
</tr>
<tr>
<td><strong>Grammarinator</strong></td>
<td>5,391</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
Compiler Middle-end Coverage

![Middle-end Line Coverage Graph]

Legend:
- GrayC
- GrayC-No-Cov-Guidance
- GrayC-Fragments-Fuzzing
- Clang-Fuzzer
- PolyGlot
- RegExpMutator
- No-Fuss-Fuzzer
- Initial Corpus
Compiler Backend Coverage

![Graph showing the coverage of different tools over time. The x-axis represents hours, and the y-axis represents the backend line coverage (LLVM). The graph compares GrayC, GrayC-NO-COV-GUIDANCE, GRAYC-FRAGMENTS-FUZZING, CLANG-FUZZER, POLYGLOT, REGEXP_MUTATOR, NO-FUSS-FUZZER, and INITIAL CORPUS.]
# Bugs Found in 24h

<table>
<thead>
<tr>
<th>Tool</th>
<th>Component</th>
<th>Fix Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Middle</td>
<td>Front</td>
</tr>
<tr>
<td><strong>GRAYC</strong></td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td><strong>GRAYC-No-Cov-Guidance</strong></td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td><strong>RegExpMutator</strong></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Clang-Fuzzer</strong></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td><strong>PolyGlot</strong></td>
<td>-</td>
<td>1</td>
</tr>
</tbody>
</table>
Testcase Contribution

We contributed 24 test cases to LLVM’s test suite: 16 of them getting accepted + 8 of them under review.

Unit tests to improve code coverage

Adds a batch of C tests that have been found to cover several hundred lines of Clang/LLVM that are not covered by the unit and regression tests of the main LLVM project, nor by the test suite when run with the -O3 configuration.

The tests were originally generated using our fuzzer, and were then reduced using C-Reduce and some manual inspection. They have been checked for undefined behaviour—freedom using Frama-C and CompCert, and manually checked to eliminate implementation-defined behaviour.

Differential Revision: https://reviews.llvm.org/D118234
Greybox compiler fuzzing for languages with extensive UB is feasible

Significant gains in terms of bug-finding & coverage compare to prior work

Key idea is to use AST-level semantics-aware mutations

GrayC found 30 bugs (26 fixed), with 25 previously unknown (22 fixed)

We used GrayC to contribute 24 test cases (16 accepted) to the LLVM compiler