EmbedSanitizer: Runtime Race Detection Tool for 32-bit Embedded ARM

Hassan Salehe Matar, Serdar Tasiran, Didem Unat
Department of Computer Engineering
Koç University

The 17th International Conference on Runtime Verification
13 - 16 September 2017
Outline

1. Introduction
2. Motivation
3. Related work
4. Our solution
5. Experimental evaluation
6. Future work
1 Introduction

● We motivate data race detection in embedded systems natively;
  ◆ No virtualization ◆ No emulation ◆ No alternative hardware

● We propose tool for detecting data races in 32-bit applications. Many solutions do not take embedded systems nature into consideration
  ◆ E.g; ThreadSanitizer - 64bit ◆ Intel Inspector - x86_64

● We promote utilization of existing tools by adapting them to embedded systems architectures
2 Motivation

- Embedded Systems are becoming multicore → multi-threaded programs

- We aim to detect races as they can:
  - Cause crashes
  - Degrade quality of service

- No tools to detect races for 32-bit ARM applications
3 Related Work

   - Static analysis using virtualization

2. Race detection for interrupt-based systems
   - No multithreading

3. Goldilocks
   - For Java, needs JVM

4. Intel Inspector, Valgrind DRD, ThreadSanitizer
   - Limited support for 32-bit ARM

**Solution:** Leverage ThreadSanitizer for 32-bit ARM → EmbedSanitizer
4 ThreadSanitizer: Architecture & Workflow

```
$ clang Program64.cpp -fsanitize=thread
```

ThreadSanitizer in LLVM/Clang
4 ThreadSanitizer: Architecture & Workflow

ArgParser fails if target is not 64-bit
4 ThreadSanitizer: Architecture & Workflow

```bash
$ clang Program64.cpp -fsanitize=thread
```

ThreadSanitizer in LLVM/Clang
4 ThreadSanitizer: Architecture & Workflow

```bash
$ clang Program64.cpp -fsanitize=thread
```

ThreadSanitizer in LLVM/Clang
4 ThreadSanitizer: Architecture & Workflow

```bash
$ clang Program64.cpp -fsanitize=thread
```

ThreadSanitizer in LLVM/Clang
4 EmbedSanitizer: Architecture & Workflow

```$ clang Progam64.cpp -fsanitize=thread
```

```$ clang ProgamARMv7.cpp -fsanitize=thread
```

**LLVM/Clang**

- **ArgParser**
- **Race detection runtime**
- **ThreadSanitizer instrumentor**
- **Linker**

**Program64.exe**

**ARMv7 support**

**ThreadSanitizer**

**Sync support**

**ProgramARMv7.exe**

**EmbedSanitizer** in LLVM/Clang

**ThreadSanitizer in LLVM/Clang**
4 (a) Enabling ARMv7 Instrumentation

- Clang does not allow instrumentation of 32-bit ARM applications
- We “hack” it so it enables instrumentation of such programs

```$ clang ProgramARMv7.cpp -fsanitize=thread```

**EmbedSanitizer** in LLVM/Clang
4 (b) Detecting Synchronization Operations

- For instrumenting synchronization function calls, e.g., `pthread_create`, `pthread_barrier_wait`, etc.

- ThreadSanitizer uses interceptors:
  - Hard to interpret
  - Low level

EmbedSanitizer in LLVM/Clang

```bash
>$ clang ProgramARMv7.cpp -fsanitize=thread
```
4 (c) Implementing Race Detection Runtime

- For detecting data races
- Complex runtime in ThreadSanitizer only for 64-bit architectures
  - Shadow memory
- Implement FastTrack algorithm
  - Precise
  - Happens-before based

$\texttt{clang ProgamARMv7.cpp -fsanitize=thread}$

**Race detection runtime**

**Sync support**

**ThreadSanitizer instrumentor**

**ArgParser**

**ARMv7 support**

**ProgramARMv7.exe**
4 (c) Implementing Race Detection Runtime

- For detecting data races
- Complex runtime in ThreadSanitizer only for 64-bit architectures
  - Shadow memory
- Implement FastTrack algorithm
  - Precise
  - Happens-before based

```bash
$ clang ProgamARMv7.cpp -fsanitize=thread
```

**EmbedSanitizer in LLVM/Clang**

- Race detection runtime
  - ThreadSanitizer instrumentor
  - Sync support
  - ARMv7 support
  - ArgParser

- Linker

- ProgramARMv7.exe

14
4 (d) Automated Installation

1. Download LLVM/Clang source code
2. Replace Clang argument parser & instrumentation with modified versions for 32-bit ARM
3. Build the modified version of LLVM/Clang as cross-compiler
4. Build our custom race detection runtime
5. Install the built cross-compiler, ready for instrumenting ARM’s 32-bit applications
## 5 Experimental Evaluation

- **Precision evaluation**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input size</th>
<th>Threads</th>
<th>Addresses</th>
<th>Locks</th>
<th>ThreadSanitizer (x86_64)</th>
<th>EmbedSanitizer (ARMv7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>4K options</td>
<td>2+1</td>
<td>28686</td>
<td>0</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>5K particles</td>
<td>2+1</td>
<td>149711</td>
<td>790</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>streamcluster</td>
<td>512 points</td>
<td>2+1</td>
<td>11752</td>
<td>2</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>swaptions</td>
<td>400 simulations</td>
<td>2+1</td>
<td>243945</td>
<td>0</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

**Summary:** EmbedSanitizer detected races in programs ThreadSanitizer did.
5 Experimental Evaluation

- Performance: race detection on ARMv7 vs on Qemu emulator

Summary: Race detection in target hardware is faster than in emulator
6 Future Work

● Further performance improvements
  ◆ Fine-grain locking

● Further usability improvements
  ◆ For bug reports

● Extended evaluation

● Analysing precision of bugs reported

● Evaluating embedded Smart TV software components
Thank You

https://github.com/hassansalehe/embedsanitizer