STPA For Use by Compiler and Binary Analysis Tools

MIT STAMP Conference 3/24/2018

Gregory Pope

John Breneman
These Tools Analyze Software Code Structure (Not Requirements)

- Static Analyzers
- Dynamic Analyzers
- Binary Analyzers
- Debuggers
- Compilers

What Faults Should These Tools Be Looking For?

Static Analysis By language:
https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#C,_C++

40 More
Can STPA Be Useful To Inform Automated Tools What to Look For and Where?

Fault:
- Software security vulnerabilities
- Software bug
- Potential system hazard
These Tools Could Find 50% of Faults Pre-Test

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>8.1%</td>
</tr>
<tr>
<td>Features and Functionality</td>
<td>16.2%</td>
</tr>
<tr>
<td>Structural Bugs</td>
<td>25.2%</td>
</tr>
<tr>
<td>Data</td>
<td>22.4%</td>
</tr>
<tr>
<td>Implementation and Coding</td>
<td>9.9%</td>
</tr>
<tr>
<td>Integration</td>
<td>9.0%</td>
</tr>
<tr>
<td>System Software Architecture</td>
<td>1.7%</td>
</tr>
<tr>
<td>Test Definition and Execution</td>
<td>2.8%</td>
</tr>
<tr>
<td>Other</td>
<td>4.7%</td>
</tr>
</tbody>
</table>

25.2%  
15%  
7%  
4.5%  
51.7%

Sample size 6,877,000 statements (comments included)
Total defects 16,209, Bugs per 1000 statements 2.3
Software Testing Techniques, Boris Beizer, Second Edition, Van Nostrand Reinhold, page 57, Table 2.1
Using Automated Static Analyzers to Debug Your Code, Pope, Ferrari, Oliver Better Software Magazine July/August 2008, page 36
Two Approaches

1. Conduct STPA analysis on software to identify structural fault types that could have system impacts.

2. Research most frequent structural software fault types.
Cross Section of Approaches

Software Faults With System Impact

Most Common Software Faults

This group would seem to be important to find and remove from software using automated tools
Looked at STPA Analysis over Multiple Industries Using Software

- **Additive Manufacturing**
  - 2015

- **Internet of Things**
  - 2016
  - [https://psas.scripts.mit.edu/home/2017-stamp-presentations/](https://psas.scripts.mit.edu/home/2017-stamp-presentations/) STPA for IOT

- **Transportation**
  - 2017
  - [https://psas.scripts.mit.edu/home/2017-stamp-presentations/](https://psas.scripts.mit.edu/home/2017-stamp-presentations/) STPA for IOT
Drilling Deeper

- The Tri-Industry STPA studies all showed “software fault” as a casual factor, but can we get more specific?
- What type of “software faults” would fail the STPA guide questions?
Software Guide Phrases

- Identify software faults that can cause system vulnerabilities/bugs/hazards using guide phrases:
  1. A software control action is not provided or is not followed.
  2. An unsafe software control action is provided.
  3. A software control action is provided too late or out of sequence.
  4. A software control action is stopped too soon or applied too long.
Guide Phrase Analysis

A software control action is not provided or is not followed:
- Software hangs in infinite loop
- Software incorrectly overwrites data
- Software executes an unrecoverable error
- Software data value overflows
- Software makes erroneous calculation
- Software data not initialized properly
- Software goes into test mode
- Software does not return from system call

A software control action is stopped too soon or applied too long:
- Missing / Incorrect Requirement
- Incorrect or Corrupted Data
- Incorrect Delay Value
- Deadlock
- Incorrect Loop Index

An unsafe software control action is provided:
- Missing / Incorrect Requirement
- Logic error
- Unhandled exception case

A software control action is stopped too soon or applied too long:
- Missing / Incorrect Requirement
- Incorrect or Corrupted Data
- Incorrect Delay Value
- Deadlock
- Incorrect Loop Index
There is an Almost Infinite Set of Software Possible Faults

- Since we can not easily identify all faults:
- Identify the most common C++, C faults*:
  - Research LLNL scientific C,C++ codes
  - Research Industry Codes C, C++ codes

* According to a 2016 survey by IEEE Spectrum, C and C++ took the top two spots for being the most popular and used programming languages in embedded systems.
The Pareto Principle Shows Up In Fault Detection In Software (IEEE)

On the Value of Static Analysis for Fault Detection in Software Jiang Zheng, Student Member, IEEE, Laurie Williams, Member, IEEE, Nachiappan Nagappan, Member, IEEE, Will Snipes, Member, IEEE, John P. Hudepohl, and Mladen A. Vouk, Fellow, IEEE

https://collaboration.csc.ncsu.edu/laurie/Papers/TSE-0197-0705-2.pdf

<table>
<thead>
<tr>
<th>Table 13: Pareto Effect in ASA Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1 fault:</td>
</tr>
<tr>
<td>Possible use of NULL Pointer</td>
</tr>
<tr>
<td>% all faults</td>
</tr>
<tr>
<td>45.92</td>
</tr>
<tr>
<td>% critical faults</td>
</tr>
<tr>
<td>60.86</td>
</tr>
<tr>
<td>Top 5 faults:</td>
</tr>
<tr>
<td>Top 1 fault +</td>
</tr>
<tr>
<td>Possible Access Out-of-Bounds</td>
</tr>
<tr>
<td>(Custodial) pointer not freed or returned</td>
</tr>
<tr>
<td>Memory leak</td>
</tr>
<tr>
<td>Variable not initialized before using</td>
</tr>
<tr>
<td>% all faults</td>
</tr>
<tr>
<td>77.26</td>
</tr>
<tr>
<td>% critical faults</td>
</tr>
<tr>
<td>85.11</td>
</tr>
<tr>
<td>Top 10 types:</td>
</tr>
<tr>
<td>Top 5 faults +</td>
</tr>
<tr>
<td>Inappropriate deallocation</td>
</tr>
<tr>
<td>Suspicious use of ;</td>
</tr>
<tr>
<td>Data overrun</td>
</tr>
<tr>
<td>Type mismatch with switch expression</td>
</tr>
<tr>
<td>Control flows into case/default</td>
</tr>
<tr>
<td>% all faults</td>
</tr>
<tr>
<td>89.87</td>
</tr>
<tr>
<td>% critical faults</td>
</tr>
<tr>
<td>90.42</td>
</tr>
</tbody>
</table>
LLNL ASA Code Research (Klocwork)

Code A - 937 KSLOC

Code B - 696 KSLOC

Code C - 35 KSLOC

Code D - 509 KSLOC

Code E - 25 KSLOC

Code F - 1,277 KSLOC

Code G - 143 KSLOC

Code H - 191 KSLOC

Code I - 124 KSLOC

Code J - 104 KSLOC

Code K - 1790 KSLOC
Research Results

- IEEE (NCSU) Study ~ 33 million LOC C, C++, since 2001 NORTEL (Network Services Code)
- LLNL Study ~ 6 million LOC C,C++, Scientific Codes since 2006
# Research Comparisons

**Top 5 Agree**

### IEEE Research Faults
- **Top 1 fault:** Possible use of NULL Pointer
- **Top 5 faults:**
  - Possible Access Out-of-Bounds (Custodial) pointer not freed or returned
  - Memory leak
  - Variable not initialized before using
- **Top 10 types:**
  - Inappropriate deallocation
  - Suspicious use of ;
  - Data overrun
  - Type mismatch with switch expression
  - Control flows into case/default

### LLNL Research Faults

<table>
<thead>
<tr>
<th>Occurences</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Null Pointer Deref</td>
</tr>
<tr>
<td>20</td>
<td>Uninitialized Variable</td>
</tr>
<tr>
<td>13</td>
<td>Buffer Overflow</td>
</tr>
<tr>
<td>8</td>
<td>Memory Leak</td>
</tr>
<tr>
<td>5</td>
<td>Freeing Freed Memory</td>
</tr>
<tr>
<td>5</td>
<td>Unvalidated Loop Iterator</td>
</tr>
<tr>
<td>4</td>
<td>Suspicious Use before null check</td>
</tr>
<tr>
<td>3</td>
<td>Return Local Var</td>
</tr>
<tr>
<td>3</td>
<td>Non Null Terminated</td>
</tr>
<tr>
<td>1</td>
<td>Infinite Loop</td>
</tr>
</tbody>
</table>

---

*Lawrence Livermore National Laboratory*
Top 5 Faults in C ++ and C Codes Could Cause Computer Systems To Reboot

1. Null or Stale Pointer Use
2. Memory Out of Bounds
3. Memory Leak
4. Variable Not Initialized Before Use
5. Inappropriate Deallocation

FAA orders Boeing 787 safety fix: Reboot power once in a while – Seattle Times 12/1/2016
Where Should We Look:

Narrow View of Developer

Prime Code
STPA Takes a Systems View

Operating System

Utilities

Compiler

Resource Manager

Libraries

Tests

Prime Code

Feeder Code

Data

Open Source Code

Data Generation Code

No Bugs
Real World Example, Left In Test Code

HP laptops found to have hidden keylogger

11 December 2017

Hundreds of HP laptop models were affected

Hidden software that can record every letter typed on a computer keyboard has been discovered pre-installed on hundreds of HP laptop models.
And This: Software Supply Chain Vulnerability

Traditional supply chain

Software supply chain

source/dependencies build systems/ engineers network application repository deployed systems

制止虫害
Supply Chain Example – QuadRooter
900 Million Android Phones Impacted
Compiler Definition

a computer program that translates a program written in a high level language into another language usually machine language.
**C++ Source Code Compiler**

Calculate the value of $n!$, where $n$ is an positive integer, from 1 to 8.

Example: $6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$

Understandable to an Software Engineer - C++

```
#include <iostream>
using namespace std;

int main()
{
    int num, factorial = 1;
    cout << "Enter Number To Find Its Factorial: ";
    cin >> num;
    for (int a = 1; a <= num; a++)
        factorial *= a;
    cout << "The Factorial of " << num << " = " << factorial << endl;
    return 0;
}
```

Understandable to a Computer - Assembler
Compilers Touch Every Line of Code and Focus On:

- Speed of compilation
- Efficiency of object code produced
- Good error detection / error messages
- Fixing problems programmers find
- Because of heavy usage tend to be robust
Where Do Compilers Come From?

Assembler

 Compiler v1

 Compiler v2

 Compiler v3

 Compiler v4

 Compiler v5

 Compiler v6

 Compiler v7

= Compiles
May Not Have the Source Code e.g. Binary Analysis Examples

- A priori virus and worm signature detection
- Suspicious file tampering
- Similarity / Differences
- Big Five fault detection
- Obstification techniques detection
- Dead code detection
- Back door detection
- Alteration detection
- Third party code detection
OSS Tool ROSE Framework Operates on Source or Binary

Input Source Code

Source Code: C, C++, Java, Fortran, Python, PHP, OpenMP or Binary

Code to Operate on Syntax and Semantics (IR)

Transformed Source Code or Report

Output Source Code

Source Code Output is unique to ROSE

Binary code (or Machine Language) is what compilers (and then Assemblers) turn source code into so computers can understand it.

Also known as Firmware when inside little black boxes.

http://rosecompiler.org/
Example: What If the Compiler Adds Something? *

- How would we know?
- Proliferate indefinitely over compiler generations.
- Open Source compilers help (No).
- Compiler Industry based on trust.
- Perhaps we should trust and verify.

* Reflections of Trusting Trust, Ken Thompson, 1984 Turing Award Speech: https://dl.acm.org/citation.cfm?doid=358198.358210
Compiler Trust and Verify Approach

Compiler’s Source Code - C++

Compiler’s Binary Code Really Has This Source

C++ Compiler

Source

ROSE Compiler Framework

Binary

Report
Meta Compilers Of The Future

Source Code

Pre-Processor

Compiler

Null Pointer Tool
Un Init Variable Tool
Buffer Overflow Tool

Memory Leak Tool
Memory Deallocate Tool
User Custom Tool

Object Code

Report
Lessons Learned from STPA

1. STPA is useful informing automated software tools which faults (potential vulnerabilities and hazards) to look for.

2. STPA shows how the software ecosystem and supply chain also can contribute faults.

3. STPA indicates the mixed blessings of compilers.

4. STPA shows common software faults transcend multiple industries.

5. Five common C++ and C code faults (potential vulnerabilities and hazards) represent 80% of faults identified.