STPA For Use by Compiler and Binary Analysis Tools

MIT STAMP Conference 3/24/2018

Gregory Pope

John Breneman

Lawrence Livermore National Laboratory

LLNL-PRES-747315

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

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++

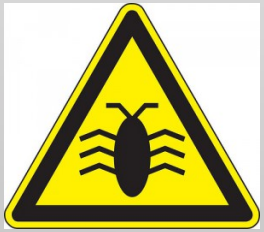
40 More
http://www.softwaretestinghelp.com/tools/top-40-static-code-analysis-tools/

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

Requirements	8.1%	
Features and Functionality	16.2%	
Structural Bugs	25.2%	25.2%
Data	22.4%	15%
Implementation and Coding	9.9%	7%
Integration	9.0%	4.5%
System Software Architecture	1.7%	
Test Definition and Execution	2.8%	
Other	4.7%	51.7%

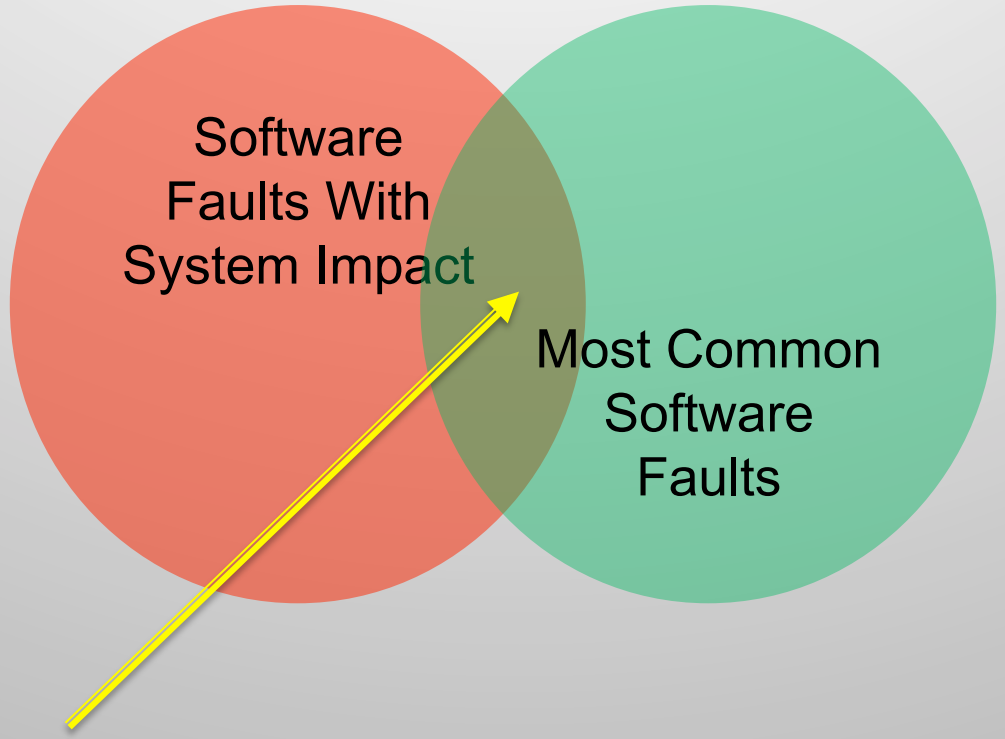
Sample size 6,877,000 statements (comments included)
Total defects 16,209, Bugs per 1000 statements 2.3
Software Testing Techniques, Boris Beizer, Second Edition, Van Nostrand Reinhold, page 57, Table 2.1
Using Automated Static Analyzers to Debug Your Code, Pope, Ferrari, Oliver Better Software Magazine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

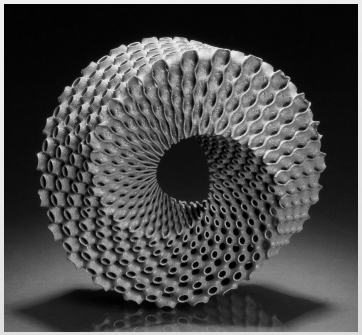


This group would seem to be important to find and remove from software using automated tools

Looked at STPA Analysis over Multiple Industries Using Software

- Additive Manufacturing

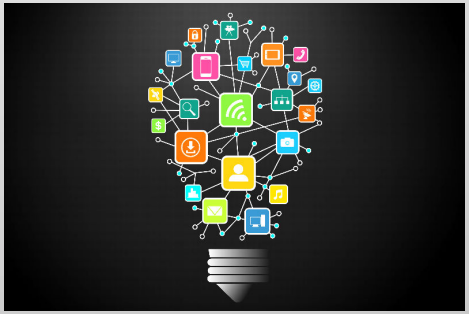
https://psas.scripts.mit.edu/home/2016-stamp-workshop-presentations Paper 11



2015

- Internet of Things

https://psas.scripts.mit.edu/home/2017-stamp-presentations/
STPA for IOT



2016

- Transportation

https://psas.scripts.mit.edu/home/2017-stamp-presentations/
STPA for IOT

2017

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

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

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)

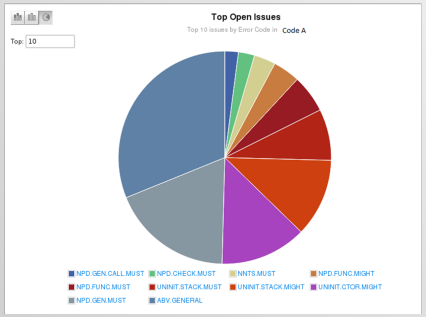
TABLE 13
Pareto Effect in ASA Faults

	% all faults	% critical faults
Top 1 fault: Possible use of NULL Pointer	45.92	60.86
Top 5 faults: Top 1 fault + Possible Access Out-of-Bounds (Custodial) pointer not freed or returned Memory leak Variable not initialized before using	77.26	85.11
Top 10 types: Top 5 faults + Inappropriate deallocation Suspicious use of ; Data overrun Type mismatch with switch expression Control flows into case/default	89.87	90.42

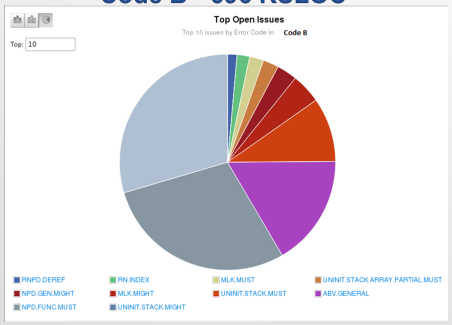
On the Value of Static Analysis for Fault Detection in Software Jiang Zheng, Student Member, IEEE, Laurie Williams, Member, IEEE, Nachiappan Nagappan, Member, IEEE, Will Snipes, Member, IEEE, John P. Hudepohl, and Mladen A. Vouk, Fellow, IEEE https://collaboration.csc.ncsu.edu/laurie/Papers/TSE-0197-0705-2.pdf

LLNL ASA Code Research (Klocwork)

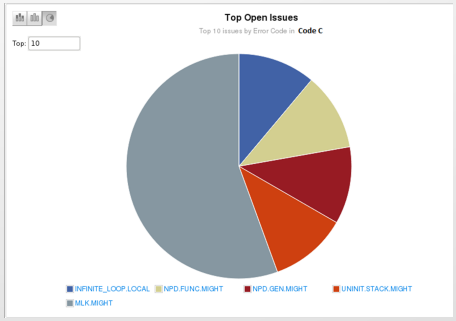
Code A - 937 KSLOC



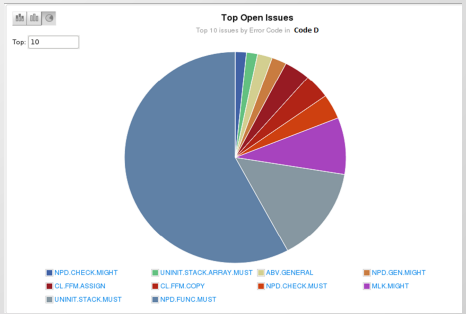
Code B - 696 KSLOC



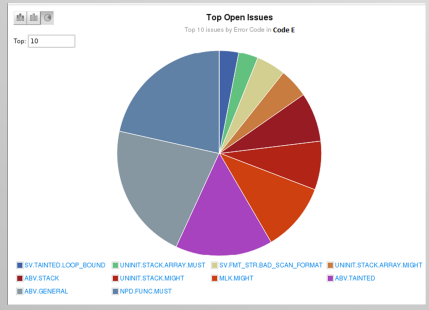
Code C – 35 KSLOC



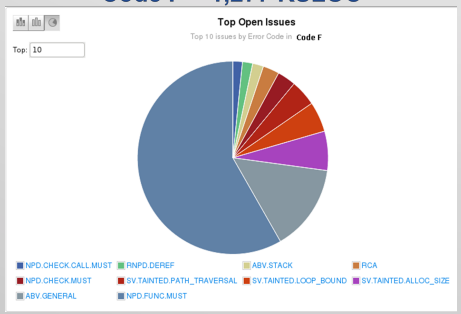
Code D – 509 KSLOC



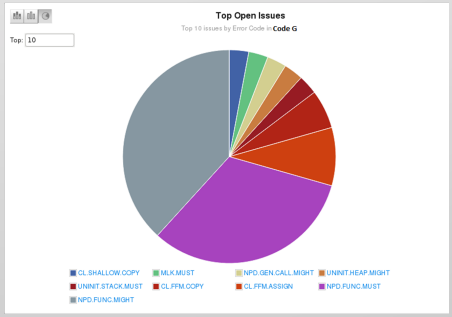
Code E – 25 KSLOC



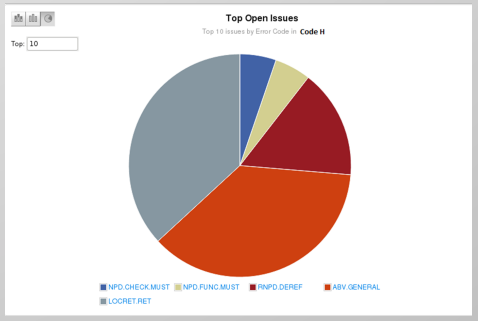
Code F – 1,277 KSLOC



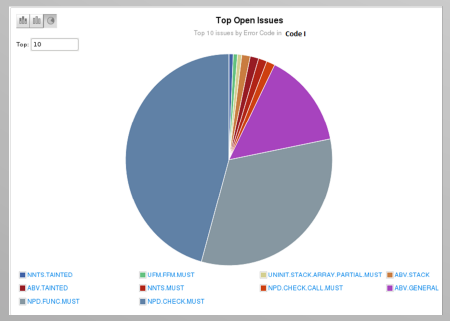
Code G – 143 KSLOC



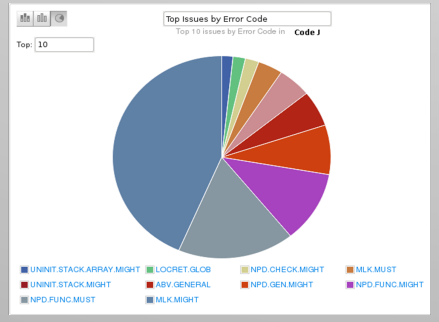
Code H – 191 KSLOC



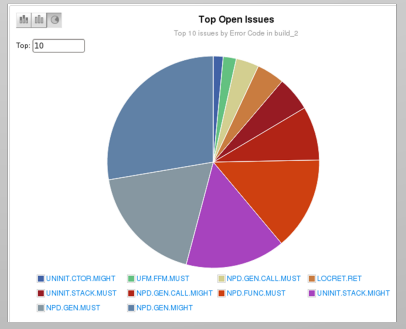
Code I – 124 KSLOC



Code J – 104 KSLOC



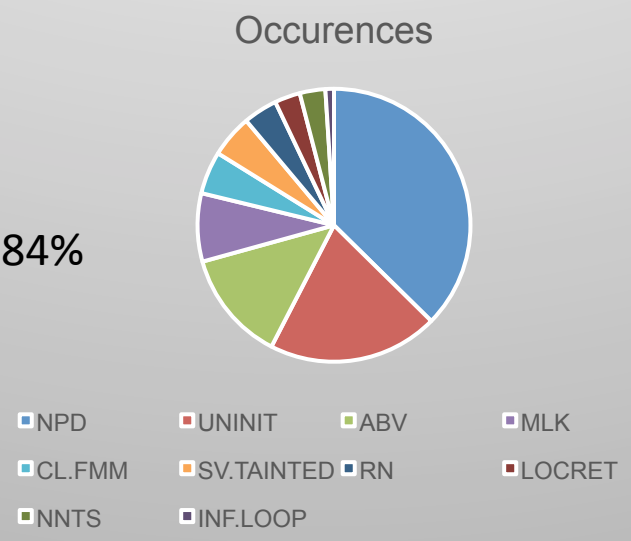
Code K – 1790 KSLOC



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

Top 5 83.84%



Code	KSLOC
A	973
B	696
C	35
D	509
E	25
F	1277
G	143
H	191
I	124
J	104
K	1780
Total	5857

Research Comparisons
Top 5 Agree

IEEE Research Faults

Top 1 fault:
Possible use of NULL Pointer

Top 5 faults:
Top 1 fault +
Possible Access Out-of-Bounds
(Custodial) pointer not freed or returned
Memory leak
Variable not initialized before using

Top 10 types:
Top 5 faults +
Inappropriate deallocation
Suspicious use of ;
Data overrun
Type mismatch with switch expression
Control flows into case/default

LLNL Research Faults

Occurences	Issue
37	Null Pointer Deref
20	Uninitialized Variable
13	Buffer Overflow
8	Memory Leak
5	Freeing Freed Memory
5	Unvalidated Loop Iterator
4	Suspicouis Use before null check
3	Return Local Var
3	Non Null Terminated
1	Infinite Loop

Top 5 Faults in C ++ and C Codes Could Cause Computer Systems To Reboot

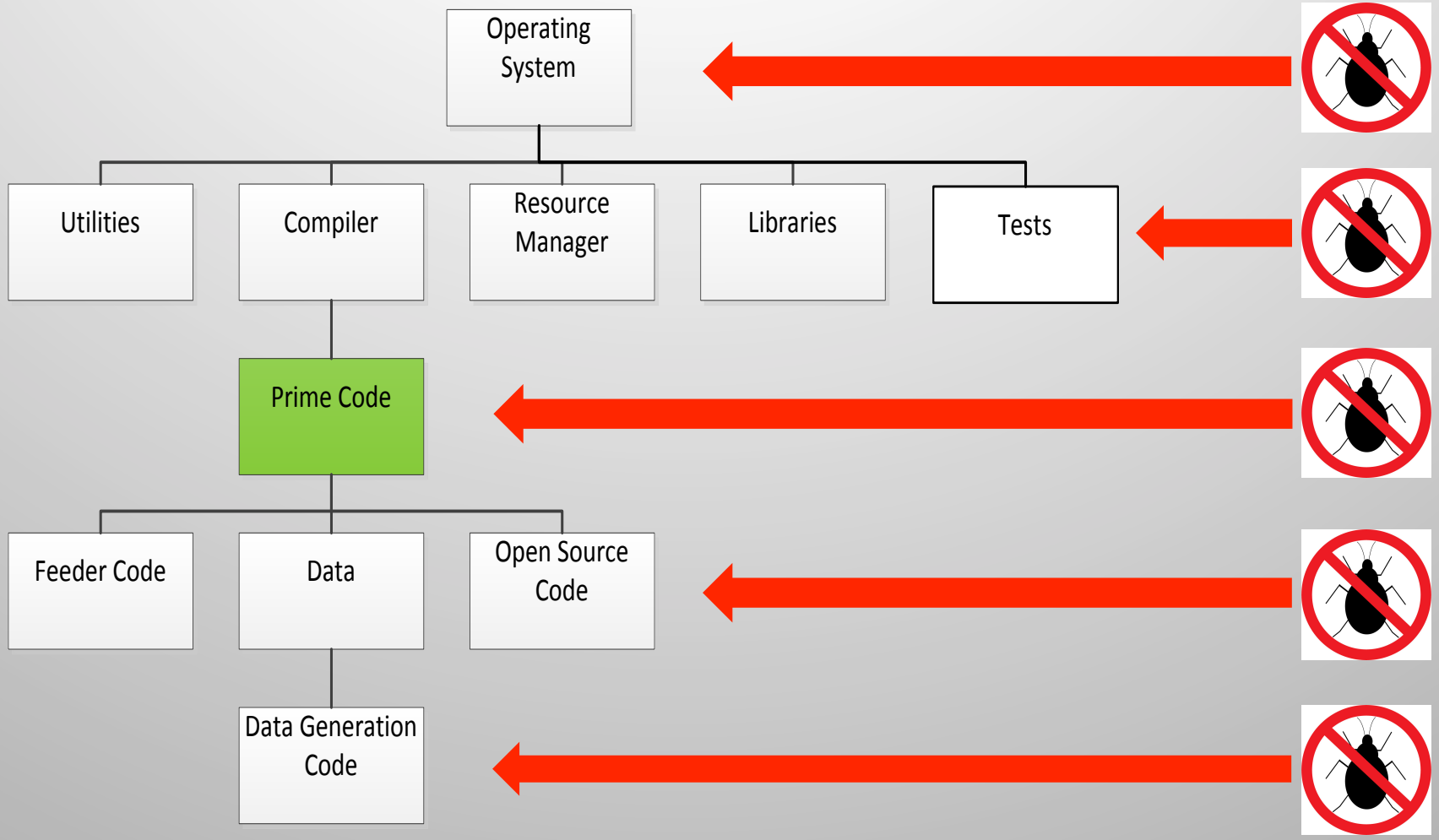
1. Null or Stale Pointer Use
2. Memory Out of Bounds
3. Memory Leak
4. Variable Not Initialized Before Use
5. Inappropriate Deallocation

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

STPA Takes a Systems View



Real World Example, Left In Test Code

HP laptops found to have hidden keylogger

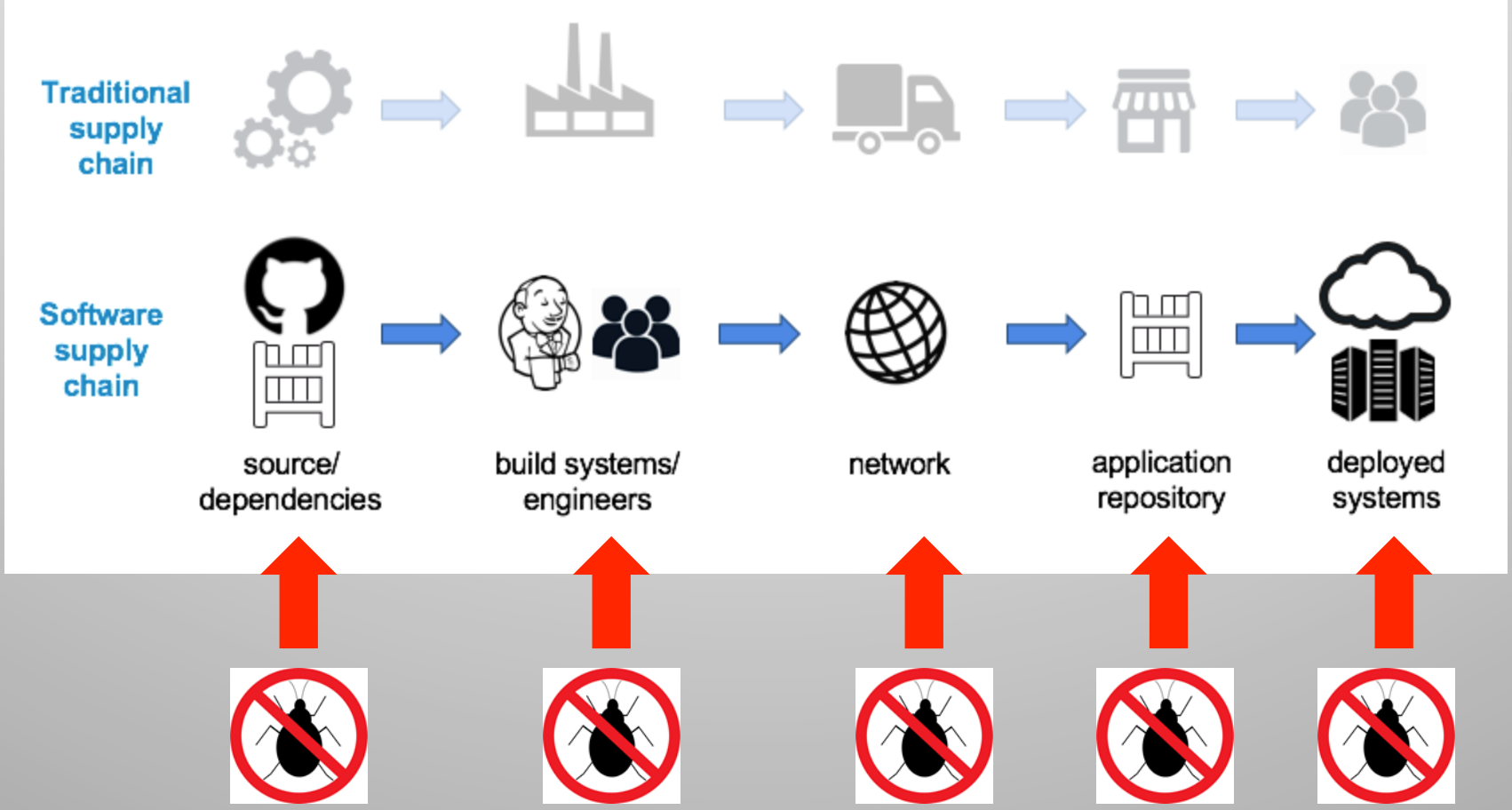
11 December 2017

Share

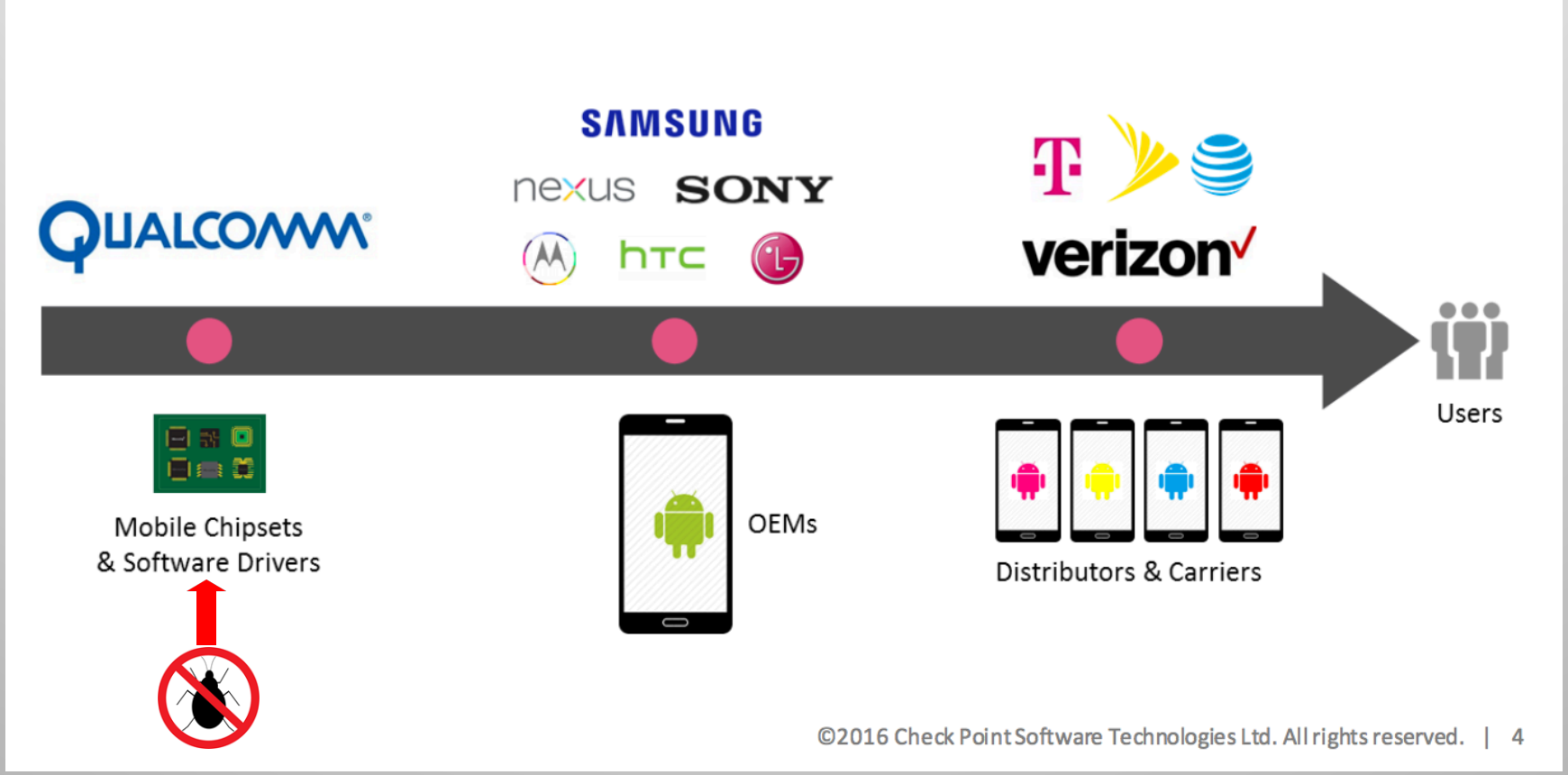
Hundreds of HP laptop models were affected

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

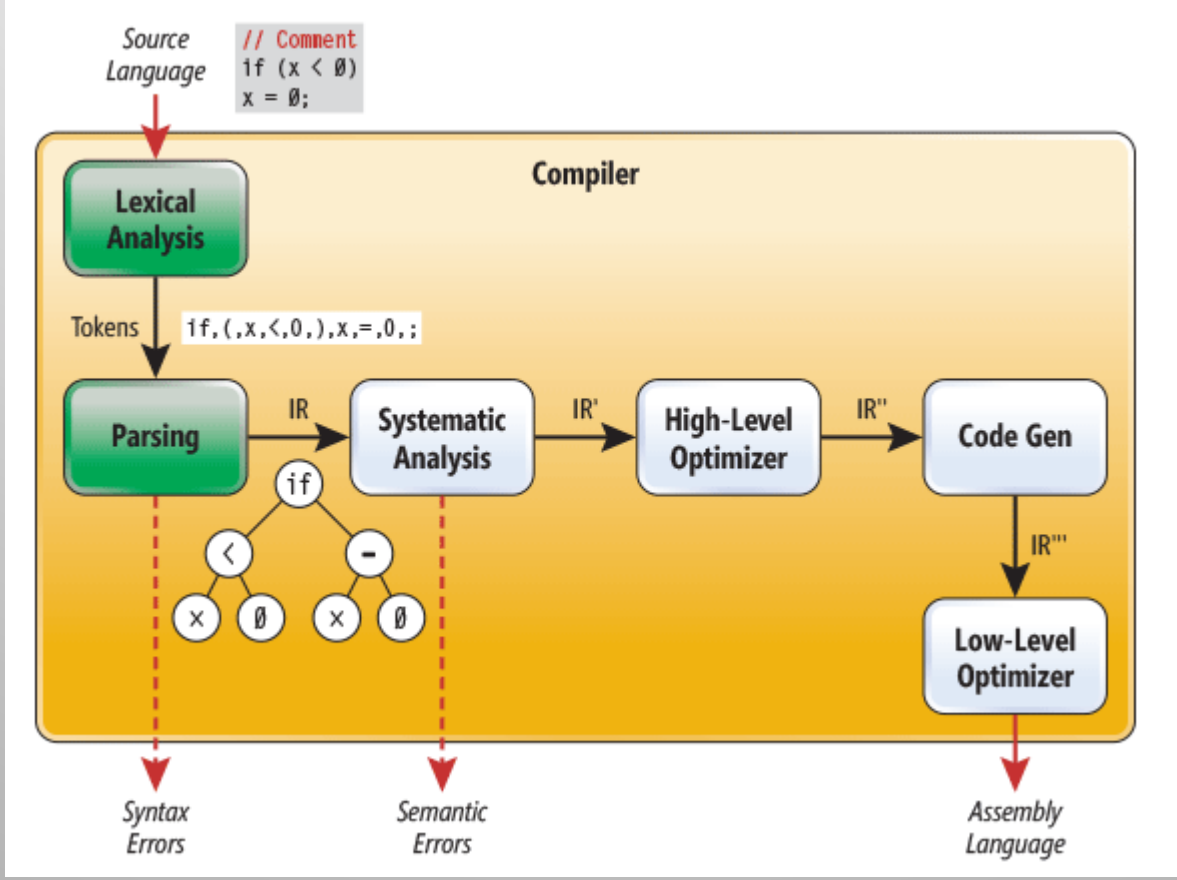


Supply Chain Example – QuadRooter 900 Million Android Phones Impacted



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

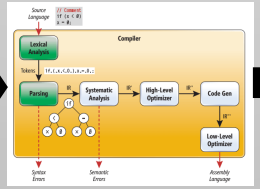
Calculate the value of n!, where n is an positive integer, from 1 to 8.

Example: 6! = 1x2x3x4x5x6= 720

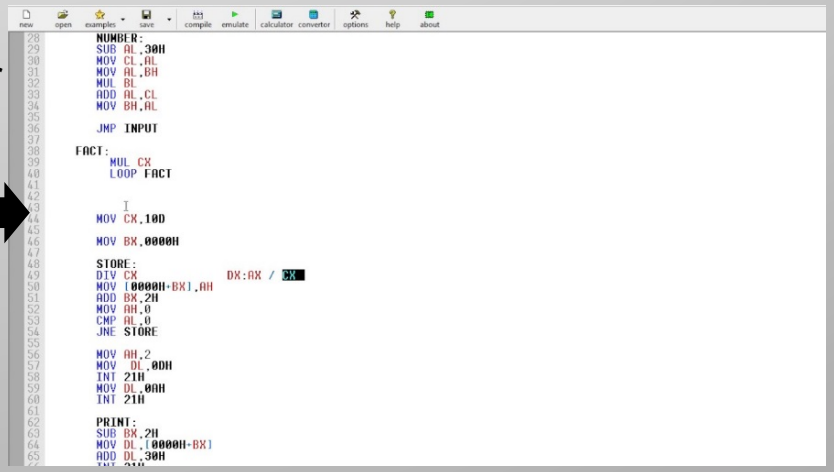
Understandable to an Software Engineer - C++

```cpp
#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=factorial*a;
    }

cout<<" The Factorial of "<<num<<" ="<<factorial<<endl;

    return 0;
}
```

C++ Compiler



Understandable to a Computer - Assembler



```
        NUMBER:
        SUB AL,30H
        MOV CL,AL
        MOV AL,BH
        MUL BL
        ADD AL,CL
        MOV BH,AL

        JMP INPUT

    FACT:
          MUL CX
          LOOP FACT


        MOV CX,10D

        MOV BX,0000H

        STORE:
        DIV CX              DX:AX / CX
        MOV [0000H+BX],AH
        ADD BX,2H
        MOV AH,0
        CMP AL,0
        JNE STORE

        MOV AH,2
        MOV DL,0DH
        INT 21H
        MOV DL,0AH
        INT 21H

        PRINT:
        SUB BX,2H
        MOV DL,[0000H+BX]
        ADD DL,30H
```

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 priory 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

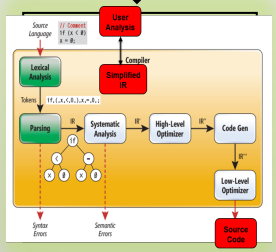
OSS Tool ROSE Framework Operates on Source or Binary

Input Source Code

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

Source
or Binary
Code

Code to
Operate on
Syntax and
Semantics (IR)



Output Source Code

Source Code Output is unique to ROSE

Transformed
Source Code
or Report

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.

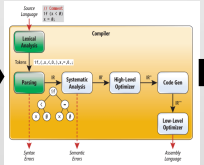
* *Reflections of Trusting Trust*, Ken Thompson, 1984 Turing Award Speech: https://dl.acm.org/citation.cfm?doid=358198.358210

Compiler Trust and Verify Approach

Compiler's Source Code- C++

```
compile(s)
char *s;
{
        ...
}
```

C++ Compiler



Compiler's Binary Code
Really Has This Source

```
compile(s)
char *s;
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        ...
}
```

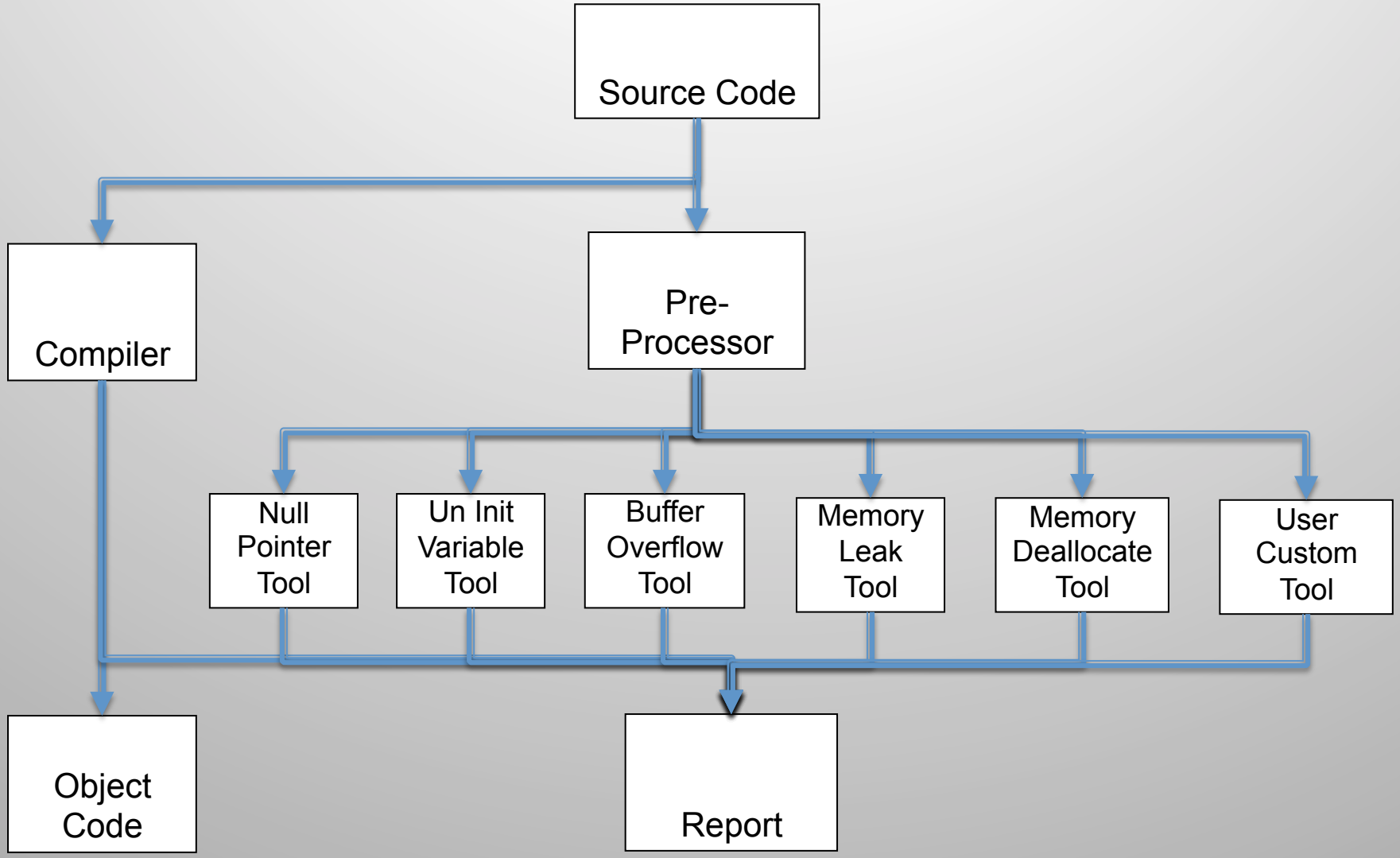
Source

ROSE Compiler Framework

Binary

Report

Meta Compilers Of The Future



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.