Course Overview

Woosuk Lee

2021 Spring Semester

CSE 6049 Program Analysis

Hanyang University, Korea

About Me

- Instructor: Woosuk Lee (이우석, woosuk@hanyang.ac.kr)
- Research Area: Program Analysis, Program Synthesis
- Homepage: http://psl.hanyang.ac.kr
- Office: Rm403, 3rd Engineering Building
- Office Hours: Thu 10:00 - 12:00

Course Information

- Course website: http://psl.hanyang.ac.kr/courses/cse6049_2021s/
- Time: Monday 16:00 — 19:00
- TA: Baljiniam Bassan Ochir (shortly Baska)
  - Email: bbumbuul@yahoo.com
- Textbook: Xavier Rival and Kwangkeun Yi, Introduction to Static Analysis: an Abstract Interpretation Perspective, MIT Press, 2020.

Why Take This Course?

- Learn principled approaches for estimating SW behaviors
- Learn how to build specialized tools for software diagnosis
    - Can be applied to improve reliability, security, performance, etc.

History of SW Bugs
— The Ariane Rocket Disaster (1996)

https://youtu.be/PK_yguLapgA?t=80s

History of SW Bugs
— The Ariane Rocket Disaster (1996)

- Caused due to numeric overflow error
  - Attempt to fit 64-bit format data into 16-bit space
- Cost
  - $100M for loss of mission
  - Multi-year set back to the Ariane program
- Read more at : http://www.around.com/ariane.html

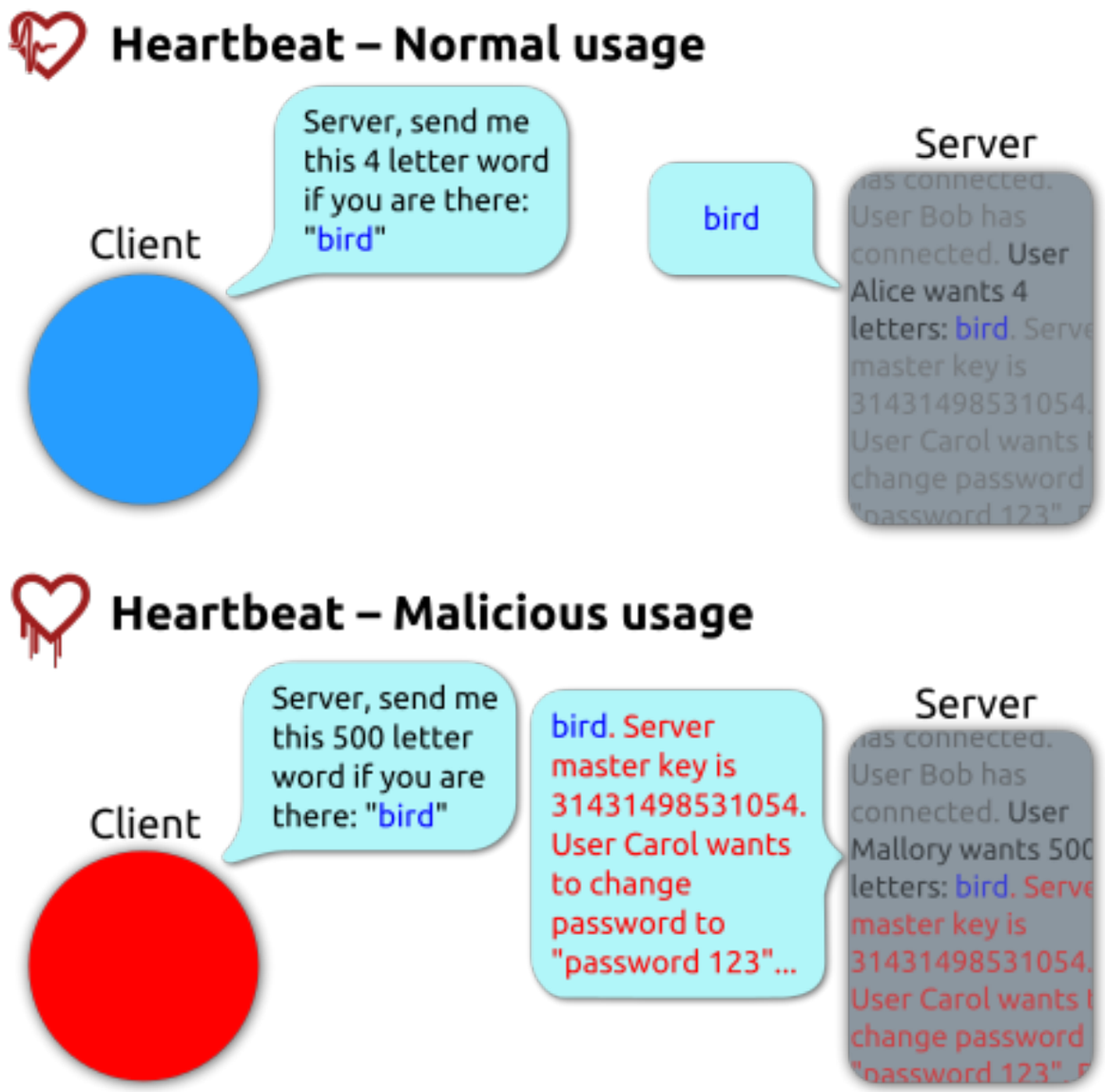
History of SW Bugs — Heartbleed (2014)

- A security bug in the OpenSSL cryptography library

- Released: 2012 Feb

- Discovered: 2014 April

- Due to buffer-overflow error



History of SW Bugs

Knight Capital Trading Glitch (2012)
— $ 440M

Nissan Airbag Malfunction (2014)
— 1 Million Vehicles Recalled

Boeing 747 Max Crashes
— 350 people died

- **(1998) NASA's Mars climate orbiter lost in space. Cost: $125 million**
- **(2000) Accidents in radiation therapy system. Cost: 8 patients died**
- **(2007) Air control system shutdown in LA airport. Cost: 6,000 passengers stranded**

. . . Countless software projects failed in history.

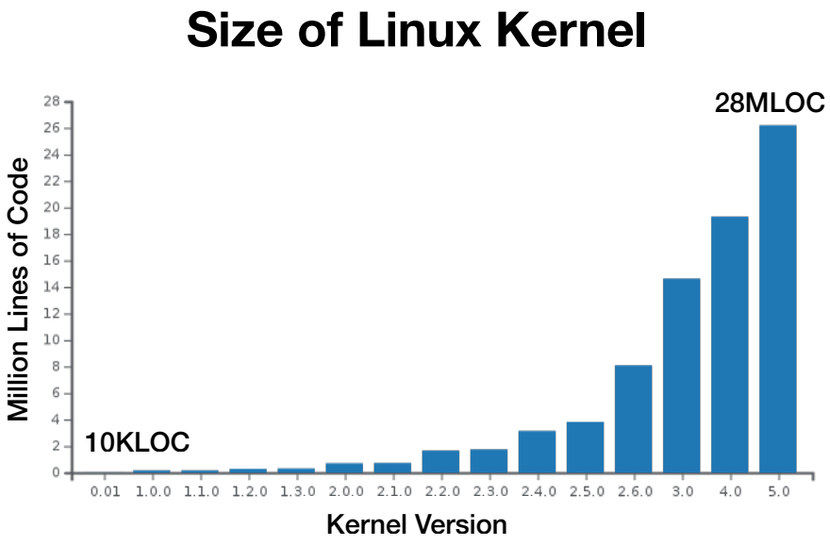
Damage of SW Bugs

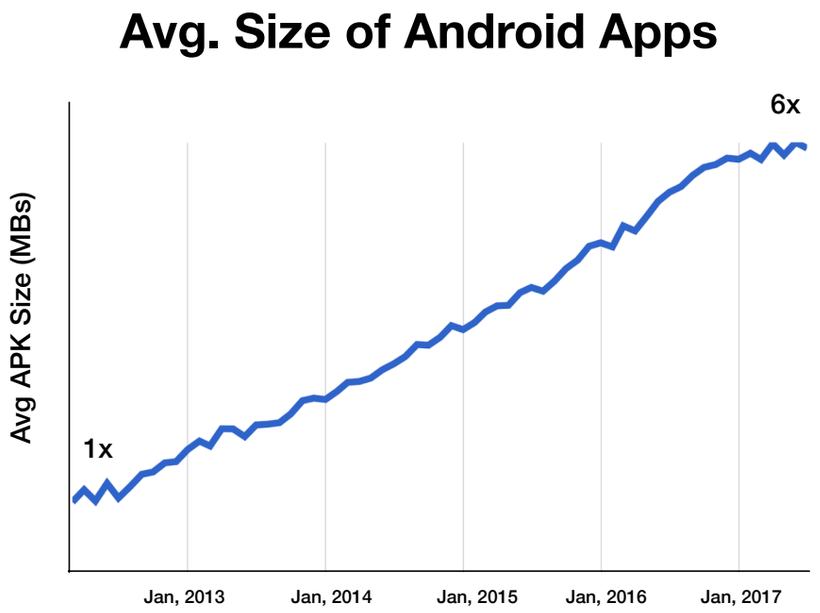
- According to CISION PR Newswire (2020. 05), SW bugs cost $ 61 Billion loss in productivity annually.

- According to Software Fails Watch (Tricentis, 2017), SW bugs lead to $ 1.7 Trillion revenue lost.

https://www.prnewswire.com/news-releases/study-software-failures-cost-the-enterprise-software-market-61b-annually-301066579.html
https://www.tricentis.com/news/tricentis-software-fail-watch-finds-3-6-billion-people-affected-and-1-7-trillion-revenue-lost-by-software-failures-last-year/

Why Software Fails?





X

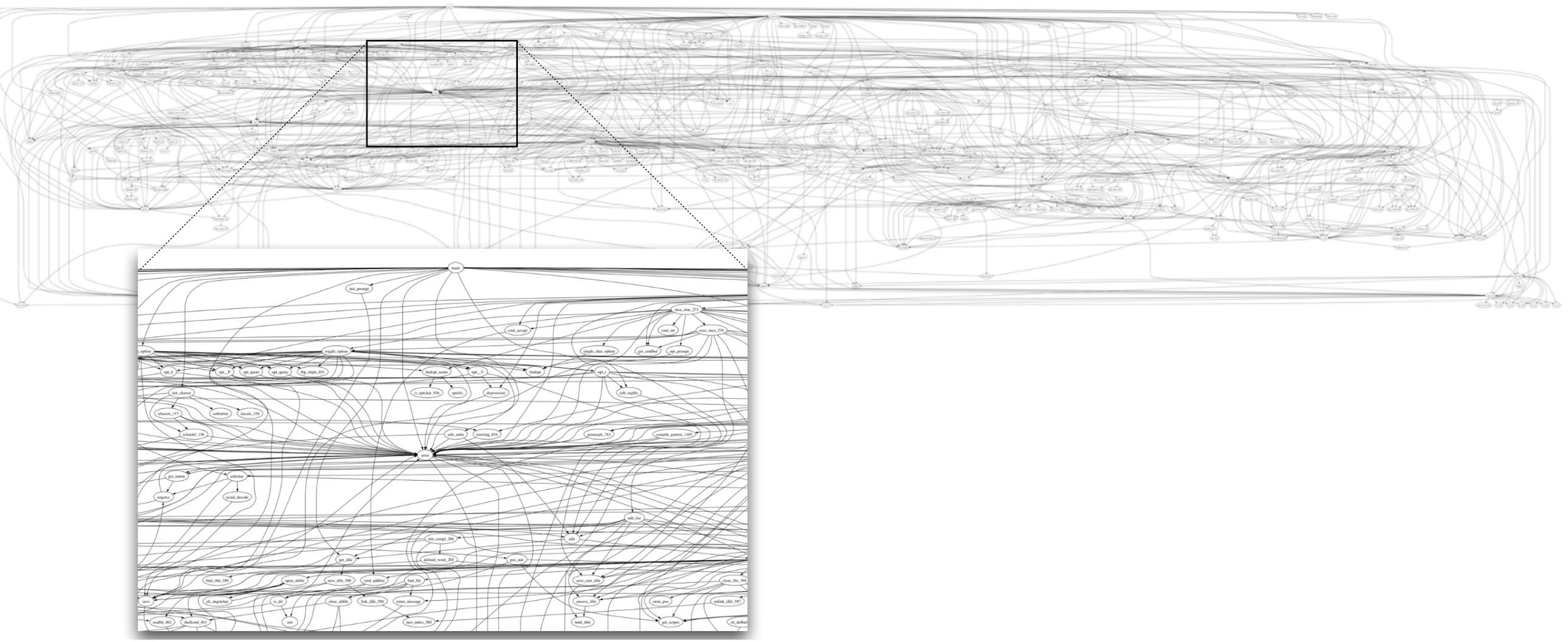
GitHub

10M+ New Developers
44M+ New Repositories
87M+ New Pull Requests
in 2019

From https://github.com/prosyslab-classroom/is593-2020-spring/blob/master/slides/lecture1.pdf

SW Complexity

less-382 (23,822 LOC)



Program Analysis

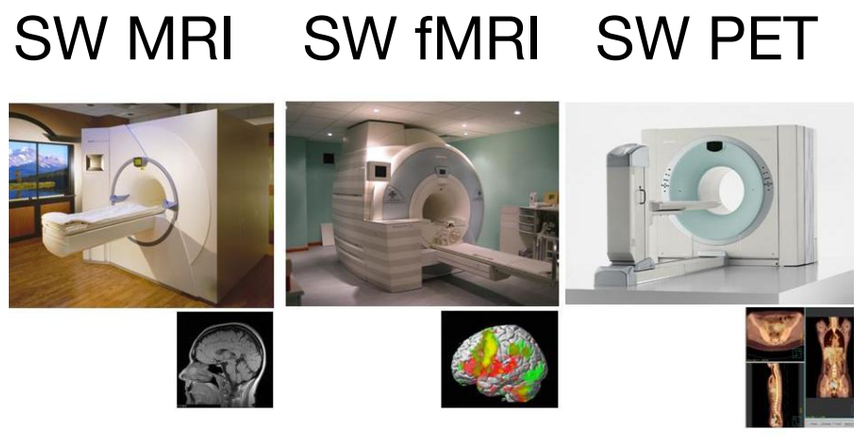
- Body of work to discover useful facts about programs
- Broadly classified into three kinds:
  - Dynamic (execution time)
  - Static (compile-time)
  - Hybrid (combines dynamic and static)

Dynamic Analysis

- Infer facts of program by monitoring its runs
- Examples:
  - Purify: array bound checking
  - Eraser: datarace detection
  - Valgrind: memory leak detection

Static Program Analysis (focus of this course)

A general method for
automatic and sound approximation of
sw run-time behaviors
before the execution



- "before": statically, without running sw
- "automatic": sw analyzes sw
- "sound": all possibilities into account
- "approximation": cannot be exact
- "general": for any source language and property
  - C, C++, C#, F#, Java, JavaScript, ML, Scala, Python, JVM, Dalvik, x86, Excel, etc
  - "buffer-overrun?", "memory leak?", "type errors?", "x = y at line 2?", "memory use $\leq \boldsymbol{2K}$?", etc

Various Static Analysis Tools

Domain-specific Verification

Windows Device Driver

Microsoft

Astrée

Airbus Controller

ENS / AbsInt

General-purpose Bug-finding

Stanford / Synopsys

Facebook

SNU / Fasoo.com

Mathworks

GrammaTech

Semmle / Github

JuliaSoft

Course Objective: Theory

Abstract Interpretation: A powerful theoretical framework for designing correct static analysis

- "**framework**" : correct static analysis comes out, reusable
- "**powerful**" : all static analyses are understood in this framework
- "**simple**" : prescription is simple
- "**eye-opening**" : any static analysis is an abstract interpretation

Course Objective: Practice

Programming assignments

- 4 main + 1 pre-requisite
- You will write static analyzers in OCaml (https://ocaml.org)
- Simple, safe, realistic and high-level programming language
- Submit yours to TA via email