Seminar

Foundations of Static Analysis
Winter Semester 24/25

TuCan-No: 20-00-1028-se

Course Type: 2SWS / 3CPs

Workload: ~90hours

Prof. Dr.-Ing. Mira Mezini

Process

- **Today:** Send your favorite 3 topics to naeumann@cs.tu-darmstadt.de

  Subject: “[FoSa24] : Topic Selection”

- **Thursday, Oct 24:** We inform you about your assigned topic via e-mail
- **Next:** Contact your supervisor and schedule a meeting to discuss the topic and requirements in detail
- **End of January:** Discuss a preliminary version with your supervisor
- **February 9th:** Send the final version (4 pages/person + appendix, acmart - https://www.acm.org/publications/proceedings-template)
- **About end of february:** Blockseminar (Date and Details will be announced)

Calling Conventions of Native-compiled Languages

High-level programming languages allow polymorphism through different “dynamic dispatch” mechanisms. Detection and resolution of virtual callsites is a core challenge of static analysis of binary code.

Task: Perform a survey of native-compiled programming languages w.r.t virtual callsite mechanisms. Study existing research & tools, classify them by approach, features, shortcomings etc.

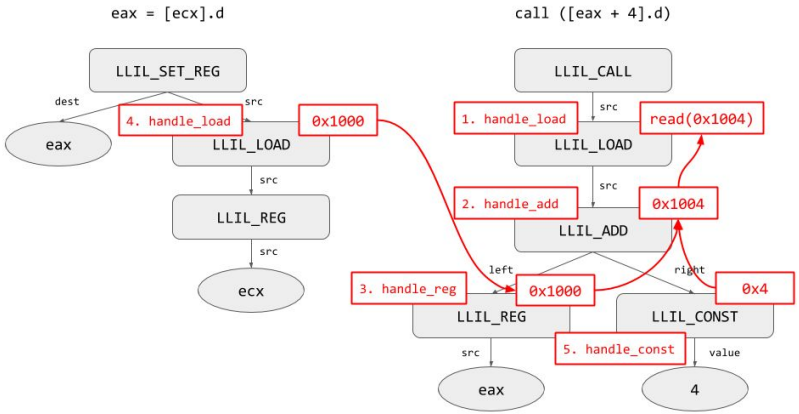
Suitable for: 1 - 2 people

Useful skills: (C++, binary reverse engineering)

Starting points:

- https://blog.trailofbits.com/2017/02/13/devirtualizing-c-with-binary-ninja
- Recovery of Object Oriented Features from C++ Binaries - APSEC 2014
- connected papers

Contact: naeumann@cs.tu-darmstadt.de



https://blog.trailofbits.com/2017/02/13/devirtualizing-c-with-binary-ninja/

Deep Learning on Binary Executables

Deep learning has been applied for various static analysis problems on binary code, such as vulnerability detection or code clone detection.

Task: Perform a survey of research applying ML to statically analyze binaries. Classify different problems and approaches, assess the approaches strengths and weaknesses.

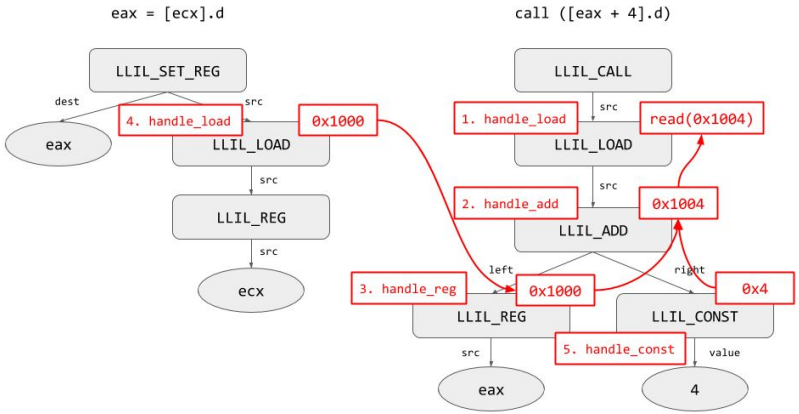
Suitable for: 1 - 2 people

Useful skills: (overview of deep learning methods)

Starting points:

- Deep-Learning-Based Vulnerability Detection in Binary Executables
- connected papers

Contact: naeumann@cs.tu-darmstadt.de



https://blog.trailofbits.com/2017/02/13/devirtualizing-c-with-binary-ninja/

Survey of Cross-Platform Frameworks

Cross platform development frameworks allow building apps for multiple platforms from a single platform-independent codebase. Currently, Flutter and React Native are the most widely used frameworks. Research existing work (publications/tools) that analyzes apps built using a cross-platform framework (reverse engineering, static/dynamic analysis etc.)

Task: Study existing research, summarize research challenges in a structured way. Study existing tools, classify them by approach, features, shortcomings etc.

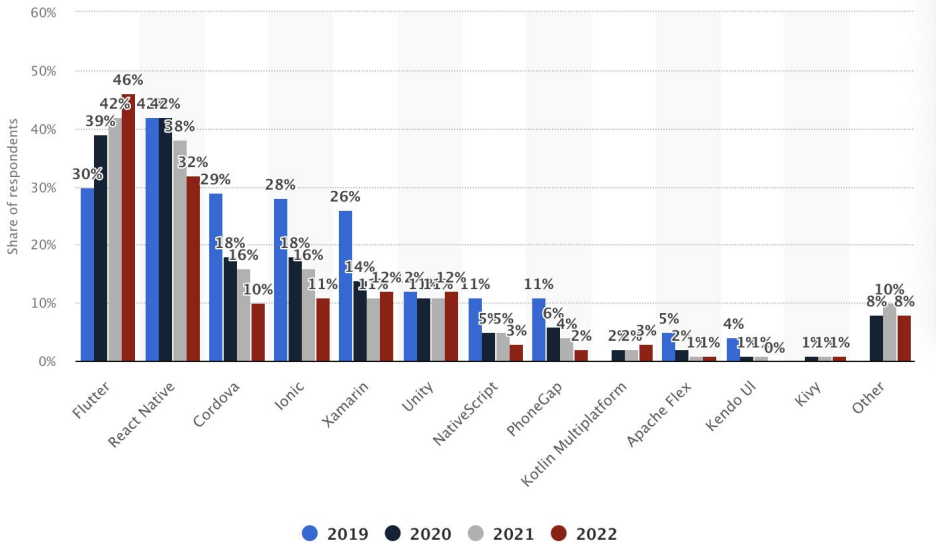
Suitable for: 1 - 2 people

Languages and Frameworks: (Java, C++)

Starting point:

- https://github.com/ptswarm/reFlutter
- https://is.muni.cz/th/goqzu/The_Security_of_Flutter_s_Ar

Contact: naeumann@cs.tu-darmstadt.de



Current State of Points-To Analysis

Points-to analyses are the prerequisite of other more complex analyses, like call-graph analyses. In this work you have to survey the state of the art.

Task: Look at the given literature. Search for newer papers. Summarize and survey them in a structured way.

Suitable for: 1 - 2 people

Starting point:

> Andersen, Lars O. (1994). “Program Analysis and Specialization for the C Programming Language”. PhD thesis. University of Copenhagen.
> url: https://www.cs.cornell.edu/courses/cs711/2005fa/papers/andersen-thesis94.pdf
>
> Steensgaard, Bjarne (1996). “Points-to analysis in almost linear time”. In: Proceedings of the 23rd ACMSIGPLAN-SIGACT symposium on Principles of Programming Languages. POPL.
> url: https://courses.cs.washington.edu/courses/cse503/10wi/readings/steensgaard-popl96.pdf

Contact: roth@cs.tu-darmstadt.de

Automated Reasoning with Separation Logic

Separation logic is a powerful framework for reasoning about programs that manipulate shared mutable data structures. This project will explore various automated techniques that leverage separation logic to enhance program analysis and verification, including **bi-abduction**, **proof generation**, and **invariant generation**. The focus will be on how these techniques can improve reasoning about program properties and contribute to the overall verification process.

Task: Explain the selected automated techniques based on separation logic, and evaluate their applications in program verification. Additionally, experiment with relevant tools that implement these techniques to assess their effectiveness in practice.

Suitable for: 1 - 2 people

Starting point:

- Separation Logic and Concurrency (OPLSS 2016)
- Grasshopper program verifier based on a decidable separation logic fragment
- Infer program verifier based on bi-abduction
- Separation logic and bi-abduction

Contact: reinhard@cs.tu-darmstadt.de

Correct Bug Finding with Incorrectness Logic

Automated bug detection often faces challenges with false positives, which can mislead developers and waste debugging efforts. One promising approach to enhancing bug finding is through **Incorrectness Logic**, which allows for formal specifications of properties that programs must uphold. This project explores Incorrectness Logic and reviews state-of-the-art techniques for correct bug detection, ensuring that identified issues correspond to real defects in the software.

Task: Investigate Incorrectness Logic and compare it with other correct bug-finding methods (e.g., bounded model checking, fuzzing). Additionally, experiment with a tool that is based on Incorrectness Logic to evaluate its capabilities in detecting real software defects.

Suitable for: 1 - 2 people

Starting point:

- Incorrectness Logic
- Finding Real Bugs in Big Programs with Incorrectness Logic
- Symbolic Model Checking without BDDs explains SAT-based model checking
- CBMC bounded model checker for C
- Directed Greybox Fuzzing

Contact: reinhard@cs.tu-darmstadt.de

Program Slicing for Software Verification

Program slicing simplifies software analysis by isolating relevant code portions and removing irrelevant parts. This project will explore how program slicing can enhance error identification and correctness in software verification. By analyzing different slicing methods, the project aims to identify effective strategies for their application.

Task: Review the state of the art in program slicing and its current applications in software verification. Additionally, identify potential use cases and experiment with combining slicing and verification tools.

Suitable for: 1 - 2 people

Starting point:

- University Stuttgart: Program Analysis course (Winter Semester 2020/21)
- Program slicing
- Evaluation of Program Slicing in Software Verification

Contact: reinhard@cs.tu-darmstadt.de