Honours/Master/PhD Thesis Projects Supervised by Dr. Yulei Sui

Projects

Research Environment

You will be working with a research team which has published high-quality papers and has developed a series of program analysis techniques and tools.

The projects will be conducted based on existing tools (e.g., SVF) and honours projects developed by our research team. The project will be in close cooperation with one or more researchers and PhD students working in this area.

It offers a good opportunity for you to learn about software security, machine learning and program analysis techniques for analyzing large-scale software systems. The necessary guidance will be provided, and you will be given the chances to make practical impact to solve a real-world problem.

1 Information flow analysis for mobile applications

1.1 *Project Code:*

CS-Project-1

1.2 *No. of Students:*

1 or 2

1.3 *Research Area:*

Security, Static Analysis and Software Engineering

1.4 *Pre-requisites:*

Some understanding of mobile application development, good software programming skills

1.5 *Description:*

Android grows extremely fast during recent years, dominating over 80of world smartphone market by the end of 2013. Security issues stand out. Personal information and sensitive data leaked by mobile applications has become an increasing critical problem. Mobile devices are ubiquitous. Information leakage is a serious problem for world-wide users. Finding the information leakage can greatly reduce the security risks and contribute to a safer mobile environment.

Traditional static value-flow analysis, which uses pointer/alias analysis for modelling program control and data dependences, is useful but not enough for tracking information flow for mobile applications. Special features and semantics in Android applications, such as message sending/receiving through internet, callbacks for system-event handling, UI interaction, and components with distinct life cycles, are the major obstacles for traditional program analysis.

This project aims to develop a new information flow analysis technique for detecting information leakage in Android applications.

This project will produce an information leak analysis/tool to locate potential security problems for Android applications and help improve the existing permission system in the Android OS to prevent sensitive information from being leaked.

You are expected to build a tool for automatically detecting interesting but critical security bugs such as information leakage for Android applications. It offers a good opportunity for you to learn about program analysis techniques based on large-scale software systems and also to develop your knowledge and skills in mobile security.

2 Machine-learning-guide typestate analysis for UAF vulnerabilities

2.1 *Project Code:*

CS-Project-2

2.2 *No. of Students:*

1 or 2

2.3 *Research Area:*

Security, Static Analysis and Software Engineering

2.4 *Pre-requisites:*

Some understanding of machine-learning and software security analysis

2.5 *Description:*

Use-after-free (UAF) vulnerabilities, i.e., dangling pointer dereferences (accessing objects that have already been freed) in C/C++ programs can cause data corruption, information leaks, denial-of-service attacks (via program crashes) [11], and control-flow hijacking attacks. While other memory corruption bugs, such as buffer overflows, have become harder to exploit due to various mitigation techniques, UAF has recently become a significantly more important target for exploitation.

Typestate analysis represents a fundamental approach for detecting statically temporal memory safety errors, such as use-after-free (UAF), in C/C++ programs. Typestate analysis relies on precise pointer analysis for accurate software vulnerabilities detection. This project aims to design a machine-learning guided static analysis approach that bridges the gap between the existing typestate and pointer analyses by capturing the correlations between program features and complicated aliases that are answered conservatively by the state-of-the-art pointer analysis. The proposed project aims to learn and predict complicated likelihood software bugs by steering typestate analysis using Support Vector Machine (SVM) or TensorFlow.

3 Preventing control flow attack in C++ using points-to analysis

3.1 *Project Code:*

CS-Project-3

3.2 *No. of Students:*

1 or 2

3.3 *Research Area:*

Security, Software Engineering and Programming Languages

3.4 *Pre-requisites:*

Some understanding about program analysis and good software development skills with large systems

3.5 *Description:*

Software is often subject to external attacks that aim to control its behavior. The majority of the attacks rely on some form of control hijacking to redirect program execution. For instance, a buffer overflow in an application may result in a call to a sensitive system function, possibly a function that the application was never intended to use.

Control-Flow Integrity (CFI) is a defensive technique that can disallow illegal control transfers that are not present in the applications Control Flow Graph (CFG). Many previous CFI approaches build a memory safety sandbox to achieve integrity by extending the runtime system. The enforcement is done by assigning tags to indirect branch targets and checking that indirect control transfers point to valid tags at runtime.

Fine-grained enforcement of CFI, however, can introduce significant overhead. The construction of an accurate control flow graph requires the use of a precise pointer analysis. This project aims to enable precise demand-driven points-to analysis to provide strong CFI guarantee for protecting virtual call attacks in C++. Additionally, the students also encourage to investigate how to leverage the static information to reduce overhead sandboxing by eliminating redundant instrumentations if they are proven to be unnecessary.

4 Modeling life-cycle of Android applications using static program analysis

4.1 *Project Code:*

CS-Project-4

4.2 *No. of Students:*

1 or 2

4.3 *Research Area:*

Software Engineering and Programming Languages

4.4 *Pre-requisites:*

Some understanding about program analysis and good software development skills with large systems

4.5 *Description:*

Android apps are not stand-alone applications but are plugins into the Android framework. An unusual and fundamental feature of Android is that an application process's lifetime is not directly controlled by the application itself.

An app may consist of different components with a distinct lifecycle. During an apps execution, the framework calls different callbacks within the app, notifying it of system events, which can perform start/receive/destroy/pause/resume/shutdown operations in the app.

It is important that application developers understand how different application components (in particular Activity, Service, and BroadcastReceiver) impact the lifetime of the application's process. Not using these components correctly can result in the system killing the application's process while it is doing important work, or result in poor user experience or consume limited system resources unnecessarily.

5 Static and dynamic data races detection for C/C++ programs

5.1 *Project Code:*

CS-Project-5

5.2 *No. of Students:*

1 or 2

5.3 *Research Area:*

Software Engineering and Programming Languages

5.4 *Pre-requisites:*

Some understanding about data races, dynamic analysis and good software development skills with large systems

5.5 *Description:*

In the multicore era, concurrent programming, an effective way of utilising computation resources, is more important than ever. However, concurrency bugs such as data races are one of most severe defects that hamper concurrent programming. A data race occurs when two or more threads access the same memory location and at least one of them is a write. Data races, as one of the major sources of concurrency bugs, are hard to find during software testing since multi-threaded programs usually exhibit an excessively large number of thread interleavings.

Existing dynamic race detectors (such as Google's ThreadSanitizer and Intel's Parallel Inspector) detect data races by repeatedly running such tools on different program inputs. In contrast, static analysis tools can detect data races without actually running the program, but may report a large number of false positives.

This project aims to develop a practical data-race detection tool by combining static and dynamic analysis techniques to reduce false positives and detect more data races that dynamic analysis tools cannot find alone.

This project will produce a tool to automatically detect data races in concurrent C/C++ + pthread programs. It offers a good opportunity for you to learn about program analysis techniques for large-scale software systems and also to develop your knowledge and skills in concurrent programming.

6 Symbolic execution for detecting system bugs on binary code

6.1 *Project Code:*

CS-Project-6

6.2 *No. of Students:*

1 or 2

6.3 *Research Area:*

Software Engineering and Binary Code Analysis

6.4 *Pre-requisites:*

Some understanding about reverse engineering and good software development skills with large systems

6.5 *Description:*

Binary analysis is powerful for detecting bugs and security vulnerabilities for programs whose source code is not available. However, due to the lack of source-code information, binaries are challenging to analyse. Symbolic execution, as a promising approach for software testing, is a program analysis technique that executes a program with symbolic rather than concrete inputs. Symbolic execution can be used to detect software bugs by automatically generating test cases to replay those errors. Some existing tools include KLEE, CUTE and PathFinder.

This project aims to develop new symbolic execution techniques for detecting system errors in binary code based on some existing open-source tools such as BitBlaze (http://bitblaze.cs.berkeley.edu).

7 Designing and implementing a memory-safe C language and its runtime library

7.1 *Project Code:*

CS-Project-7

7.2 *No. of Students:*

1 or 2

7.3 *Research Area:*

Programming Languages and Compilers

7.4 *Pre-requisites:*

Good understanding about programming languages, e.g., C/C++, and good software development skills with large systems.

7.5 *Description:*

C is one of the most widely used programming languages of all time. It is the foundation language of many system software components such as OS and embedded applications. However, its unsafe features, such as weak-typing, pointer arithmetic, void pointers and non-safe casting, often lead to memory corruption errors, including buffer overflow, memory leaks and dangling pointers.

This project aims to develop a new safe C language (implemented by a compiler front-end and a runtime library in LLVM) by eliminating undisciplined use of C features and extending LLVM's native executable runtime environment to guarantee memory safety.

8 Precise and efficient pointer analysis for Javascript programs

8.1 *Project Code:*

CS-Project-8

8.2 *No. of Students:*

1 or 2

8.3 *Research Area:*

Software Engineering and JavaScript Analysis

8.4 *Pre-requisites:*

Some understanding about static program analysis and good software development skills with large systems

8.5 *Description:*

Pointer analysis, as a fundamental research, is to identify the possible runtime values of a pointer at compile-time. It paves the way for a wide range of clients, such as compiler optimisation, software bug detection and program verification. A large number of modern web applications are implemented in JavaScript, a flexible dynamic scripting language that executes in a client browser.

Developing precise and efficient pointer analysis for hunting bugs and improving Just-in-time optimisations for Javascript is a key challenge faced by modern software development.

This project aims to develop a new flow-sensitive pointer analysis (by considering program execution order) for Javascript programs. Moreover, we expect to apply the new pointer analysis for optimising Javascript in the LLVM JIT compiler, detecting bugs (such as system crashes), and tracking information flow for real-world Javascript programs.

The proposed project will contribute to a precise and efficient flow-sensitive pointer analysis for the Javascript language.

This project will produce a tool for analysing real-world Javascript applications such as Gmail, Chrome, JQuery, JSON and AngularJS. This also offers a good opportunity for you to learn about program analysis based on large-scale software systems. Furthermore, the research results are expected to be published in a top conference.

9 Dynamic program analysis for bug detection using static program slicing

9.1 *Project Code:*

CS-Project-9

9.2 *No. of Students:*

1 or 2

9.3 *Research Area:*

Programming Languages, Compilers and Software Engineering

9.4 *Pre-requisites:*

Some understanding about programming analysis and good software development skills

9.5 *Description:*

Static analysis tools find bugs in a program without executing the program. By reasoning statically about all possible execution paths, they find bugs without relying on any program inputs but can report excessively many false positives. In contrast, dynamic program analysis tools find bugs for some particular program inputs. They are precise (by yielding few false positives) but must be repeatedly run for a large number of test cases (often blindly) to increase coverage.

This project aims to develop some dynamic analysis techniques for C/C++ programs to find software bugs (e.g., memory access errors) more efficiently with improved coverage based on static program slicing and recent advances on pointer analysis.

10 Incremental program analysis for software testing

10.1 *Project Code:*

CS-Project-10

10.2 *No. of Students:*

1 or 2

10.3 *Research Area:*

Software Engineering and Automated Software Testing

10.4 *Pre-requisites:*

Some understanding about static program analysis, program slicing and good software development skills with large systems

10.5 *Description:*

Modern software development involves many incremental changes. Regression testing provides a reliable means to verify that code base changes and additions don not break an application's existing functionality.

This project aims to develop techniques to perform incremental program analysis by leveraging previous analysis results and automatically reusing test cases to avoid over-analysing between different software revisions based on small program changes.

11 Live path control flow integrity (LPCFI)

11.1 *Project Code:*

CS-Project-11

11.2 *No. of Students:*

1 or 2

11.3 *Research Area:*

Security and Programming Languages

11.4 *Pre-requisites:*

Some understanding about program analysis and good software development skills with large systems

11.5 *Description:*

Control flow integrity (CFI) is an application security technique intended to prevent the diversion of the flow of control by attackers. CFI does this by attempting to enforce a control flow graph (CFG). A CFG is a representation of the programmer's intended flow of control. A statically generated CFG cannot be perfect but are improving.

Per-Input CFI (CFI) attempts to utilise some dynamic information by building an enforced CFG (ECFG) on the fly. The ECFG is a subgraph of the CFG and grows as the program is run, and more paths become legal. The ECFG grows monotonically, so some paths which were legal at one point, but became illegal at a later stage in a program's execution, are targets for attackers.

LPCFI is an experiment built on CFI in removing edges from the ECFG during program execution. LPCFI aims to ensure the ECFG only contains edges that can be legally reached and traversed from the current program state or at any future legal state. In other words, LPCFI aims to remove edges as they become illegal to traverse at any point in the future.

12 Static and dynamic analysis for detecting use-after-free vulnerabilities in C/C++

12.1 *Project Code:*

CS-Project-12

12.2 *No. of Students:*

1 or 2

12.3 *Research Area:*

Security and Programming Languages

12.4 *Pre-requisites:*

Some understanding about program analysis and good software development skills with large systems

12.5 *Description:*

Use-After-Free refers to the attempt to access memory after it has been freed, resulting in a dangling pointer to a block of dead memory, which may later be reallocated or overwritten. Use-after-free significantly affects the program reliability and security, because it can cause a program to crash or to be exploited, thereby opening the door to attackers who can access sensible data or hijack the program execution.

This project aims to evaluate an existing state-of-the-art tool developed by our research group during the past few years. Possibly, the students may also wish to design and implement new ideas/algorithms on top of the tool to improve its precision and/or scalability.

We will evaluate our tool using real-world industrial-sized programs (e.g., Chrome) to detect new or existing interesting vulnerabilities.