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 Static detection of software vulnerabilities using graph neural networks

1.1 *Project Code:*

CS-Project-1

1.2 *Individual or group project?*

Both suitable

1.3 *Research Area:*

Programming Languages, Security and Software Engineering

1.4 *Pre-requisites:*

Some understanding of machine-learning and software security analysis

1.5 *Description:*

Static bug detection, which approximates the runtime behaviour of a program without running it, is the major way to pinpoint bugs at the early stage of software development cycle, thus reducing software maintenance cost. The existing static analysis techniques (e.g., Coverity, Fortify Flawfinder, RATS, Checkmarx and SVF) have shown their successes in detecting traditional vulnerabilities (e.g., buffer overflows, memory leaks and use-after-frees). However, these approaches that rely on conventional static analysis theories (e.g., data-flow and abstract interpretation) are still ineffective in detecting non-traditional bugs, such as insufficient control flow management (CWE-691), business logic errors (CWE-840), and program behavioural problems (CWE-438), which are often caused by bad programming practices.

It is non-trivial for human experts to define customised rules for detecting CFR vulnerabilities. This is not only because vulnerability detection itself is difficult in a complex software system, but also because different patterns may require different detecting rules. The quality of each rule also varies with individuals. Thus, the results are often limited by their existing experience in summarizing vulnerability patterns. Simply designing an unsound analysis using user-defined pattern matching may result in a large number of false positives and/or false negatives.

This project aims to develop a new deep learning-based code embedding approach to detecting a wide variety of software vulnerabilities using graph embedding. We aim to make a new attempt by applying a recent graph convolutional network to embed code fragments in a compact and low-dimensional code representation that preserves high-level control-flow information of a vulnerable program, without the needs of manually defined rules. You are expected to apply the developed technique to detect vulnerabilities in real-world open-source projects hosted on GitHub.

2 Information flow analysis for mobile applications

2.1 *Project Code:*

CS-Project-2

2.2 *Individual or group project?*

Both suitable

2.3 *Research Area:*

Security, Static Analysis and Software Engineering

2.4 *Pre-requisites:*

Some understanding of mobile application development, good software programming skills

2.5 *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.

3 Interactive online code analysis to improve software reliability and security

3.1 *Project Code:*

CS-Project-3

3.2 *Individual or group project?*

Both suitable

3.3 *Research Area:*

Visualisation, Programming and Software Engineering

3.4 *Pre-requisites:*

Software visualisation and some understanding of software security analysis

3.5 *Description:*

Nearly 90% of the software being developed and used today is based on open source code. Due to the lack of adequate support for open source projects, the security vulnerabilities being discovered in these projects compounds annually at a staggering rate. A single vulnerability in software can be devastating, and developers can only fix issues which they can discover. WebSVF addresses this by offering interactive code analysis to improve software quality, reliability, and security.

Static Value-Flow Analysis (SVF), an open source tool for programs written in C/C++ provides analyses by scanning the flow of information and data individually and their interdependence for increasingly improved precision. Among its derived applications, it is used to visualise a program as an interactive 3D diagram which highlights files with errors and indicates the directional flow of information. The WebSVF research project, had a simple objective, to create a user-friendly Graphic User Interface (GUI) for SVF and its derived applications. Once the project achieved its initial goal of visualising the analysis/testing stage of development in an unprecedented manner using interactive bug reports and 3D diagrams, the project scope was modified to use these modular components and create an integrated online platform simplifying the remaining stages of the software lifecycle.

The next stage in research will transform WebSVF into an open source platform-as-a-service for interactive code analysis enabling users to create and share multiple, disposable online code spaces with their software projects already configured and the necessary tools and extensions pre-installed to allow seamless development, collaboration, testing and deployment.

This project will be based on our existing open-source tool WebSVF. https://github.com/SVF-tools/WebSVF.

4 Machine-learning-guide program analysis for software vulnerability detection

4.1 *Project Code:*

CS-Project-4

4.2 *Individual or group project?*

Both suitable

4.3 *Research Area:*

Programming Languages, Static Analysis and Software Engineering

4.4 *Pre-requisites:*

Some understanding of machine-learning and software security analysis

4.5 *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) and null pointer dereferences. 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.

5 Source code summarisation using deep reinforcement learning

5.1 *Project Code:*

CS-Project-5

5.2 *Individual or group project?*

Both suitable

5.3 *Research Area:*

Static Analysis and Software Engineering

5.4 *Pre-requisites:*

Some understanding of machine-learning and software analysis

5.5 *Description:*

In the life cycle of software development, nearly 90% of the effort is used for maintenance, and much of this effort is spent on understanding the maintenance task and related software source code via code documents. Thus, it is essential for documentation to provide a high level description of the task performed by code for software maintenance. Even though various techniques have been developed to facilitate programmers during software implementation and testing, documenting code with comments remains a labour-intensive task. In fact, few real-world software projects can adequately document code to reduce future maintenance costs.

This project aims to develop a new approach to automatically generating software documents (a.k.a source code summarisation) leveraging the recent advances in deep learning (particularly reinforcement learning) and natural language processing techniques or developing new machine learning models. The goal is to develop a new code representation to include both unstructured (e.g., textual code tokens) and structured code information (control- and data-flows of a program) to better support subsequent software engineering tasks, such as code summarisation.

6 Detecting smart contracts vulnerabilities in blockchain software

6.1 *Project Code:*

CS-Project-6

6.2 *Individual or group project?*

Both suitable

6.3 *Research Area:*

Security, Static Analysis and Software Engineering

6.4 *Pre-requisites:*

Some understanding of Blockchain and software security analysis

6.5 *Description:*

Smart contract is a computer program that can be automatically executed. The execution of the contract does not require third parties to supervise it. It can effectively simplify the transaction process and has many advantages such as security and reliability. With the development of digital currency, blockchain technology is being applied more and more in areas such as finance and logistics. Its core is a distributed database, with decentralization, transparency, openness, autonomy, information can not be modified, and anonymity.

Due to the features of the blockchain, traditional software engineering can not be able to effectively guide blockchain programming, in recent years, several detrimental software misbehaviors, which caused significant monetary loss and community splits, have posed the problem of the correct design, validation and execution of smart contracts. In 2017 a bug discovered in a smart contract library used by the Parity application, vulnerability makes it possible for hackers to become the owner of the library through library functions. Then the suicide function is called to cause the entire contract library to be destroyed. About 500K Ethers in the wallet was frozen.

The expected outcomes of the project are an open-source tool for automatically detecting bugs of the blockchain applications that can find some existing popular known vulnerabilies written in Solidity language.

7 Preventing control flow attack using points-to analysis

7.1 *Project Code:*

CS-Project-7

7.2 *Individual or group project?*

Both suitable

7.3 *Research Area:*

Security, Software Engineering and Programming Languages

7.4 *Pre-requisites:*

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

7.5 *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.

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

8.1 *Project Code:*

CS-Project-8

8.2 *Individual or group project?*

Both suitable

8.3 *Research Area:*

Software Engineering and Programming Languages

8.4 *Pre-requisites:*

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

8.5 *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.

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

9.1 *Project Code:*

CS-Project-9

9.2 *Individual or group project?*

Both suitable

9.3 *Research Area:*

Software Engineering and Programming Languages

9.4 *Pre-requisites:*

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

9.5 *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.

10 Symbolic execution for detecting system bugs on binary code

10.1 *Project Code:*

CS-Project-10

10.2 *Individual or group project?*

Individual

10.3 *Research Area:*

Software Engineering and Binary Code Analysis

10.4 *Pre-requisites:*

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

10.5 *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).

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

11.1 *Project Code:*

CS-Project-11

11.2 *Individual or group project?*

Both suitable

11.3 *Research Area:*

Programming Languages and Compilers

11.4 *Pre-requisites:*

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

11.5 *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.

12 Dynamic program analysis for bug detection using static program slicing

12.1 *Project Code:*

CS-Project-12

12.2 *Individual or group project?*

Individual

12.3 *Research Area:*

Programming Languages, Compilers and Software Engineering

12.4 *Pre-requisites:*

Some understanding about programming analysis and good software development skills

12.5 *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.

13 Incremental program analysis for software testing

13.1 *Project Code:*

CS-Project-13

13.2 *Individual or group project?*

Individual

13.3 *Research Area:*

Software Engineering and Automated Software Testing

13.4 *Pre-requisites:*

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

13.5 *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.