Software Security
List of projects/seminars

Title:
Develop and Evaluate a Security Analyser for Finding Vulnerabilities in C Programs

Context:
Security has become a widespread concern for all the major headlines in the dominant society. As the Internet of Things (IoT) is now spreading for all technology sections, ranging from consumer, industrial, and even government operations, the possibility of security breach has substantially widened to a considerable extent. This project aims to develop and evaluate a security analyser for finding vulnerabilities in C programs on top of the Efficient SMT-Based Context-Bounded Model Checker (ESBMC), a bug hunting tool automatically checks safety properties in C programs.

Deliverables:
- Exploit the existing verification strategies in ESBMC to find vulnerabilities (e.g., buffer overflow) in standard C applications (e.g., OpenSSH, OpenSSL, and PuTTY).
- Identify and implement suitable extensions to verify such open-source applications.

Prerequisites:
- Reasonable C++ programming skills and some interest/knowledge of logic and security are required.

What you'll learn:
- Work with a security analyzer for verifying vulnerabilities in C programs.

Resources:
- These papers give a good overview of the verification algorithms implemented in the ESBMC tool: https://ssvlab.github.io/lucasccordeiro/papers/tse2012.pdf and https://ssvlab.github.io/lucasccordeiro/papers/sttt2017.pdf.

Title:
Fuzzing a Software Verifier

Context:
Fuzzing is a verification technique that generates inputs for a program to trigger a bug or achieve an objective such as coverage. A fuzzing target may be from textual input to full-blown programs, which are a good target for program verifiers if not for its complexity. This projects aim to integrate fuzzing targets into the Efficient SMT-Based Context-Bounded Model Checker (ESBMC), which is a mature bounded model checking (BMC) tool based on Satisfiability Modulo Theories (SMT) solvers for verifying multi-threaded C programs. The fuzzing targets in this context will range from various domains, e.g., C99 programs, SSAs (from the inner language of ESBMC), SMT formulas, and unit tests. The **goal** of this project is to implement a fuzzing framework using libFuzzer on top of ESBMC.

Deliverables:
- A fuzzing suite for ESBMC, including Sanitizers and adequate coverage.
- Integration with OSS-Fuzz and ESBMC's CI/CD.

Prerequisites:
- C++ programming skills (mandatory).
- Interest in program verification (mandatory).
- Experience in fuzzing with libFuzzer or similar frameworks (optional).
- Experience with Clang Sanitizers (optional).

What you'll learn:
- Work with libFuzzer, a state-of-the-art fuzzer.

Resources:
- libFuzzer page: https://llvm.org/docs/LibFuzzer.html
- Fuzzing tutorial by Google: https://github.com/google/fuzzing/blob/master/tutorial/libFuzzerTutorial.md
- OSS-Fuzz: https://google.github.io/oss-fuzz/
- ESBMC: http://esbmc.org/

Title:
Integration and Evaluation of a String Solver to Verify Security Vulnerabilities in C Programs

Context:
Strings are widely used as a parameter for many C functions or Java classes, e.g., for opening network connection, for opening database connection, and opening files. As a result, strings are a significant source of security issues. One of the biggest challenges in verifying C and Java programs is the widespread use of character strings, which makes verification problems resulting from those programs highly complex. Solving such constraints is an active area of research. This project aims to integrate existing String solvers into the Efficient SMT-Based Context-Bounded Model Checker (ESBMC), which is a mature bounded model checking (BMC) tool based on Satisfiability Modulo Theories (SMT) solvers for verifying multi-threaded C programs. The SMT solvers Z3 and CVC4 support the most common primary accesses (e.g., obtain the length of a string and a character at a given position). Comparisons (e.g., lexicographic comparison and equality). Transformations (e.g., insertion, concatenation, replacement, and removal). Conversions (e.g., converting the primitive data types into a string and parsing them from a string).

Deliverables:
- The SMT solvers Z3 and CVC4 should be integrated into ESBMC to determine the satisfiability of a set of constraints involving string operations in C programs.

Prerequisites:
- Reasonable C++ programming skills and some interest/knowledge of logic and security are required.

What you'll learn:
- Work with a security analyzer for verifying vulnerabilities in C programs that heavily rely on strings.

Resources:
- This paper gives a good overview of the ESBMC tool: https://ssvlab.github.io/lucasccordeiro/papers/tse2012.pdf

Title:
Security Verification of CUDA Deep Neural Networks

Context:
Compute Unified Device Architecture (CUDA) is a parallel computing platform and Application Programming Interface (API) model created by NVIDIA, which extends C/C++ and Fortran, to create a computational model that aims to harness the computational power of Graphical Processing Units (GPUs). As in other programming languages, errors in CUDA programs eventually occur, in particular, due to array out-of-bounds, arithmetic overflow, and division by zero violations. Additionally, since CUDA is a platform that deals with parallel programming, specific concurrency errors related to data race and barrier divergence can be exposed due to the non-deterministic behavior of the threads interleavings. This project aims to develop and evaluate an approach for verifying CUDA programs based on the Efficient SMT-Based Context-Bounded Model Checker (ESBMC), named as ESBMC-GPU, using a CUDA operational model (COM), which is an abstract representation of the standard CUDA libraries (i.e., the native API) that conservatively approximates their semantics.

Deliverables:
- Model the CUDA Deep Neural Network library (cuDNN) and Basic Linear Algebra Subprograms (cuBLAS).
- Verify security properties that rely on such libraries using Bounded Model Checking (BMC) and Satisfiability Modulo Theories (SMT) techniques.

Prerequisites:
- Practical knowledge in graphical processing units (GPU).
- Reasonable C/C++ programming skills and some interest/knowledge of logic and CUDA are required.

What you'll learn:
- Work with a security analyzer for verifying security vulnerabilities in GPU programs.

Resources:
- These papers give a good overview of the ESBMC-GPU tool: https://ssvlab.github.io/lucasccordeiro/papers/jscp2017.pdf and https://ssvlab.github.io/lucasccordeiro/papers/sac2016.pdf

Title:
Scaling up bounded model checking to find software vulnerabilities in the Internet of Things

Context:
Bounded model checking (BMC) is a successful verification technique that has been applied to find vulnerabilities in software. The basic idea of BMC is to check the negation of a given property at given depth: given a transition system M, a property P, and a bound k, BMC unrolls the system k times and translates it into a verification condition (VC) ψ such that ψ is satisfiable if and only if P has a counterexample of depth k or less. The goal of this project is to scale up BMC by simplifying the constraint problem passed to the BMC's solver to find software vulnerabilities in IoT devices. To that end, we propose using domain partitioning to reduce the search space associated with a given constraint problem. Intuitively, carefully adding new constraints to the problem simplifies the search for solutions. For illustration, suppose that the BMC front-end produced the formula *P(x,y,z)* encoding the property to be checked. In that case, we need to decide which of the three variables to pick and how to partition its domain. If we select variable *x* with associated domain *D_x=1..100* and split the domain in half we would obtain *partitions { P & x > 50, P & !(x > 50)},* which individually are not more complex to solve than the original problem.

Deliverables:
- This project will answer two main research questions:
  - (i) which variables should be selected for partitioning?
  - (ii) how to partition the domain of variables?

Prerequisites:
- Reasonable C++ programming skills and some interest/knowledge of logic and security are required.

What you'll learn:
- Work with state-of-the-art verification techniques based on abstract interpretation for analyzing C programs.

Resources:
- This paper gives a good overview of BMC: https://ssvlab.github.io/lucasccordeiro/papers/tse2012.pdf and https://ssvlab.github.io/lucasccordeiro/papers/tacas2019_2.pdf.

Title:
Understanding Programming Bugs in Java Programs Using Counterexamples

Context:
One of the main challenges in software development is to ensure the correctness and reliability of software systems. In this sense, system failure or malfunction can result in a catastrophe, especially in critical embedded systems. In the context of software verification, bounded model checkers (BMCs) have already been successfully applied to discover subtle errors in real software projects. When a BMC tool finds an error, it produces a counterexample. On the one hand, the value of counterexamples to debug software systems is widely recognized in the state-of-the-practice. On the other hand, BMC tools often produce counterexamples that are either too large or difficult to be understood mainly because of the software size and the values chosen by the respective solver.

Deliverables:
- Develop a method to automatically collect and manipulate counterexamples produced by a BMC tool to generate new instantiated code to reproduce the identified error.
- Employ a Bounded Model Checker for Java Bytecode called JBMC to show the effectiveness of the proposed method over publicly available benchmarks from the Software Verification Competition (SV-COMP)?

Prerequisites:
- Reasonable Java programming skills and some interest/knowledge of logic are required.

What you'll learn:
- Work with state-of-the-art software verifiers for Java bytecode.

Resources:
- This paper gives a good overview of the JBMC tool: https://ssvlab.github.io/lucasccordeiro/papers/cav2018.pdfThe Java benchmarks from SV-COMP are available here: https://github.com/sosy-lab/sv-benchmarks/tree/master/java

Title:
Verifying Information Flow Security for Blockchain-based Smart Contracts

Context:
One of the main challenges in software development is to ensure the correctness and reliability of software systems. Software failure or malfunction can result in a catastrophe, thereby leading to human or monetary losses. Smart contracts based on blockchain technology represent a piece of software with critical formal correctness needs. Generally speaking, a contract consists of conditions or rules that need to be (formally) verified. In particular, a smart contract represents a contract that is developed for a specific blockchain technology, which is publically verifiable and executes automatically when the conditions or rules are met. This project will develop a static security analysis method of smart contracts based on existing software verifiers developed by the Formal Methods group (e.g., JBMC, ESBMC).

Deliverables:
- Check for information-flow principles in blockchain-based smart contracts to identify untrusted operation. This security analysis method will operate based on Ethereum (https://www.ethereum.org/), which is the dominant smart contract blockchain platform
- Identify unique, novel types of attacks to smart contracts, and it should also be applied to realistic examples of deployed smart contracts.

Prerequisites:
- Reasonable C++ programming skills and some interest/knowledge of logic are required.

What you'll learn:
- Work with state-of-the-art software verifiers for programs to find vulnerabilities in smart contracts.

Resources:
- This paper gives a good overview of a scalable security analysis framework for smart contracts: https://arxiv.org/pdf/1809.03981.pdf.
- This paper gives a good overview of the JBMC tool: https://ssvlab.github.io/lucasccordeiro/papers/cav2018.pdf.
- This paper gives a good overview of the ESBMC tool: https://ssvlab.github.io/lucasccordeiro/papers/tse2012.pdf.

Title:
Speeding Up Bounded Model Checking by Checking Verification Conditions in Parallel.

Context:
Bounded model checking (BMC) is a successful verification technique that has been applied to find vulnerabilities in software. The basic idea of BMC is to check the negation of a given property at a given depth. Given a transition system M, a property P, and a bound k, BMC unrolls the system k times and translates it into a verification condition (VC) ψ such that ψ is satisfiable if and only if P has a counterexample of depth k or less. Furthermore, such VCs are usually checked by SMT/SAT solvers using a sequential approach, which might lead to higher verification times. However, if one considers that VCs are independent of one another, if they are verified in parallel, BMC is likely to have speedups in the overall running time. This project aims to develop a parallel VC check module, in which various solver instances are created, and each one handles different VCs. If a violation is found, *i.e.*, a formula is satisfiable, the counterexample is generated, and the BMC algorithm stops.

Deliverables:
- A parallel VC check module integrated with ESBMC, preferably written using C++11 threading model, *i.e.*, *std::thread*;
- A comparison, in terms of time and memory usage, with the sequential approach is performed over publicly available benchmarks from the Software Verification Competition (SV-COMP).

Prerequisites:
- Practical knowledge in multi-threading programming;
- Reasonable C++ programming skills and some interest/knowledge of logic and security are required.

What you will learn:
- Work with state-of-the-art software verifiers for C programs.

Resources:
- ESBMC open-source repository: https://github.com/esbmc/esbmc.

Title:
Security Verification of Linux Device Drivers

Context:
Avoiding kernel vulnerabilities is critical to achieving security of many systems because the kernel is often part of the trusted computer-based system. This project will evaluate the current state-of-the-art verification techniques concerning kernel protection. Also, this project will introduce a technique to verify and exploit Linux Device Drivers vulnerabilities using software model checking techniques.

Deliverables:
- Exploit the existing verification strategies in Linux Kernel Drivers to find vulnerabilities.
- Identify and implement a technique to verify security vulnerabilities.

Prerequisites:
- C/C++ programming skills (mandatory).
- Interest in program verification (mandatory).
- Experience in Linux drivers (optional).

What you'll learn:
- Work with a security analyzer for verifying security vulnerabilities in Linux device drivers.

Resources:
- These papers give a good overview of the current state-of-the-art:
  - https://pdos.csail.mit.edu/papers/chen-kbugs.pdf
  - https://dl.acm.org/doi/10.1145/1321631.1321719
  - https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/Linux-Linux-Kernel.html