usman@vt.edu
u.nadeem.vt@gmail.com

MUHAMMAD USMAN NADEEM

/uNadeem
/UsmanNadeem

EDUCATION

MS COMPUTER SCIENCE – CGPA 3.8 – Dec 2019 (Expected) — VIRGINIA TECH
- Relevant Courses: Advanced Compilers, Multiprocessor Programming, Linux Kernel Programming, Operating Systems, System and Software Security, LLVM Programming, Software Engineering

BS COMPUTER SCIENCE – Major CGPA 3.7 — LUMS, PAKISTAN
- Relevant Courses: Distributed Computing, Computer Architecture, Network Security, Networks

EXPERIENCE

GRADUATE RESEARCH ASSISTANT — VIRGINIA TECH
- Work included research and software development based project on Symbolic Execution

GRADUATE TEACHING ASSISTANT — VIRGINIA TECH
- Computer Organization (x86 assembly, C, Linux), Software Design and Data Structures (Java)

UNDERGRAD TEACHING ASSISTANT — LUMS
- CS 200 - Intro to Programming (C++), CS 202 - Data Structures (C++)

PROJECTS

Tools/Languages/ Keywords

LLVM Code Equivalence Verification (Research Assistant – Virginia Tech**)**
- I implemented a program equivalence verification tool as a fork of KLEE, an open source symbolic execution engine
- My tool is being used to verify that instrumentation for memory safety and code obfuscation through LLVM compiler does not result in any undefined behavior or side effects

KLEE, LLVM, C++, Compilers, Software Security, Symbolic execution

Test Case Coverage Analysis in Open Source Software (Course Project – Software Engineering**)**
- I built a tool using Cobertura Framework that analyzed changes to the code coverage over multiple versions of programs written in Java; Top projects from Git Hub were analyzed
- Showed that 11% of refactored test cases had a negative influence on test-level code coverage. This effect is hidden since developers only look at the high-level coverage numbers

Software Testing, Java, Code Coverage, GitHub

Feedback-Directed Optimizations (FDO) and Branch Patterns (Course Project – Compilers**)**
- I used Intel Processor Trace and Linux Perf tool to analyze Branch patterns in the SPEC 2006 benchmark and uncovered some interesting program control-flow patterns that a compiler could utilize to generate more performance optimized code
- Showed that the *training inputs* for Feedback Directed Optimization yielded different program behavior than the *reference inputs*, e.g. SPEC2006.Omnetpp only had a *16% path overlap*

Linux Perf, Intel PT, Python, Program Optimization

Optimizing *IFRit*; A data race detector (Course Project – Multiprocessor Programming**)**
- Optimized an existing research-prototype Data Race detector and achieved 2-3x speedup for PARSEC benchmark over original tool
- Implemented *fine-grained synchronization* to improve parallelism
- I redesigned the data structures used by the tool and reduced the amount of *shared* data needed per memory location to fit in a *128-bit array*, as opposed to a nested hash table per location
- I re-implemented the data access mechanism to use Intel MPX bounds table as a key-value store

Intel MPX, Multicore Programming, C++, Performance Optimization

Hardening Android Applications to Enhance Data Privacy (Senior Year Project – Undergrad**)**
- Implemented static taint analysis for smali byte code in Java to analyze Android apps
- Tracked sensitive data from Android API calls to data sinks; Replaced it with scrambled data to prevent data leakage and maintain user privacy

Java, Smali Android bytecode, Security, Compiler Static Analysis

Operating Systems/Compilers Projects
- Implemented a Log Structured File System using FUSE for Linux
- Implemented a Linux Driver for Flash based storage devices
- Implemented a Stochastic priority based task Scheduler in Linux Kernel
- Implemented various middle/back-end compiler passes in LLVM e.g. Strength Reduction, Constant Propagation, Available Expressions, Loop-invariant Code Motion, Register Allocation

Linux Kernel, File Systems, LLVM