ALEXKIDD-FUZZER: Kernel Fuzzing Guided by Symbolic Information
(Working in Progress)
Kyunhtae Kim and Byoungyoung Lee
Department of Computer Science and CERIAS, Purdue University

PROBLEMS
• Unexplored paths and low code coverage due to low quality of inputs

OBJECTIVE
• Find more BUGS/CRASHES on various system software (i.e. OS kernel)
• Maximize kernel code coverage

PROPOSED SOLUTION
• We design ALEXKIDD-FUZZER, which overcomes limitation of fuzzing and symbolic execution.
• We first employ general fuzzing mechanism such that feasible execution paths are explored at a rapid pace.
• Furthermore, during fuzzing execution, we allow concolic engine to guide the fuzzer to make unreachable-code reachable.

MOTIVATION
• Black-box fuzzing and white-box fuzzing (i.e. symbolic execution) are both getting popular for software testing
• Black-box fuzzing has limitation of handling constant values whereas such a random testing is fast and efficient
• Symbolic execution suffers from state explosion and performance overhead of constraint solver although it can generate high quality inputs which lead to all feasible paths

PROJECT OVERVIEW
• Fuzzing logic generates and mutates input programs depending on various sources and transfer particular inputs that need to be further analyzed by agent logic
• Concolic logic records path constraints and solve them during concrete execution of input programs
• Agent logic glues between fuzzing and concolic logics by symbolizing input programs and verifying constraints

EXPECTED CODE COVERAGE

IMPLEMENTATION
• All implementation is on Ubuntu-14.04 LTS
• Main fuzzing logic is built upon Google syzkaller
• Concolic logic leverages S2E symbolic execution framework
• Agent logic is written in python 2.7

CHALLENGES
• How to handle nondeterministic behaviors caused by global state
• How to generate valid sequences of system calls in user programs

FUTURE WORK
• Measure code coverage and performance overhead
• Find real-world bugs/crashes and analyze them.