PEKO:基於軟體度量之模糊測試能量規劃
PEKO:Software-metric-basedPowerScheduleforFuzzing

國立陽明交通大學資電亥客與安全碩士學位學程

研究生　　：陳廷宇
指導教授：黃俊穎 教授

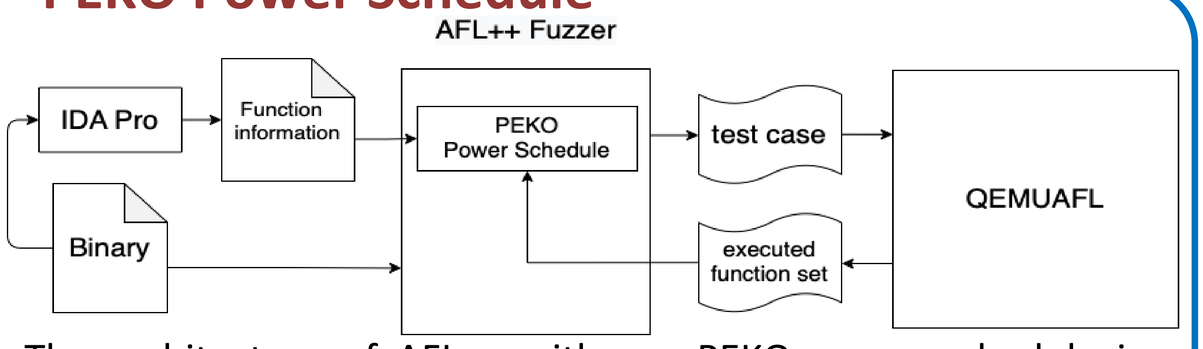
Abstract

Fuzz testing has become an integral part of software testing and vulnerability mining. Especially in vulnerability mining, human experts usually leverage a fuzzer to find a crash rapidly while conducting code audit or reverse engineering. Although current fuzz testing methods are efficient, they still unavoidably waste energy on trivial states when a program under test (PUT) is complex. We believe the waste of energy is mainly because a fuzzer lacks knowledge about a program. While it is challenging for a human expert to intervene in fuzzing processes, properly guiding a fuzzer would be the key to improving fuzzing performance. This study introduces a power schedule based on a new software metric called Procedure Error Key Omen (PEKO) to guide fuzzing processes. It drives the fuzzer to focus on possibly essential functions in a PUT by calculating the estimated PEKO value of the executed functions. PEKO values can be further fine-tuned by either human experts or external software analysis tools. Based on the feedback from humans and alternative external sources, the fuzzer can focus more on interesting parts of a PUT without wasting fuzzing resources. We integrated our PEKO power schedule into the AFL++ fuzzer, which can reach vulnerabilities in our experiments more quickly than other power schedule and it found 1 CVE in nm-new.

Motivation

The efficiency and performance of a fuzzer highly depend on the quality of test cases fed to a PUT. Good test cases lead fuzzers to reveal hidden vulnerabilities in a PUT. In contrast, poor test cases waste resources on running useless test loops. A straightforward idea to improve the performance of a power scheduler is to classify codes implemented in a program as interesting or uninteresting and spending more time exploring interesting parts. Uninteresting functions are generally well-tested implementations, such as standard implementation of hash, encoding or encryption. On the other hand, a program also has functions which are more likely to contain faults. While most of the current power schedulers only use runtime states to perform test case scheduling and are agnostic about the Program Under Test (PUT), causing them to give the same weight on the edges from uninteresting functions and dangerous functions. We believe that power scheduling with program structure awareness is beneficial to fuzzing performance.

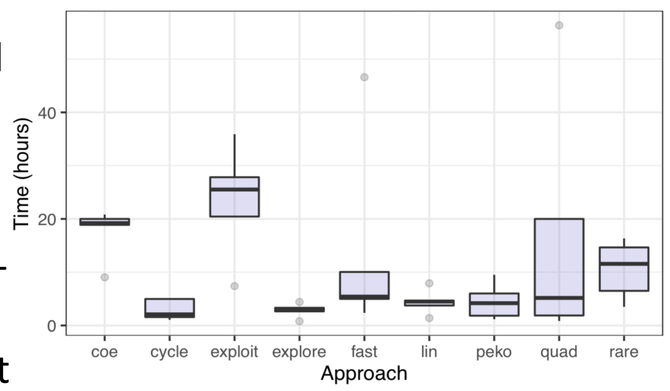
PEKO Power Schedule

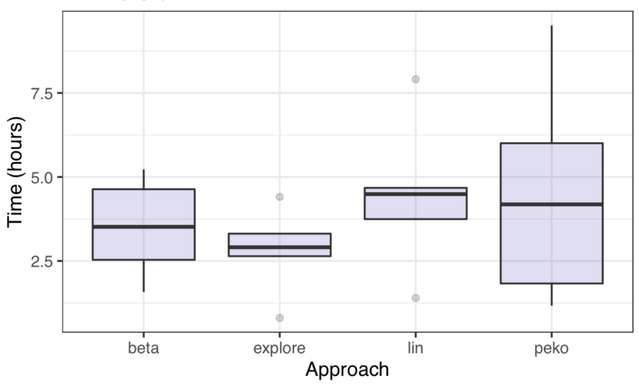


The architecture of AFL++ with our PEKO power schedule is as the figture above. Before the fuzzer start fuzzing PUT, we use the PEKO Metric Analyzer to leverage IDA pro to calculate the PEKO value of each function of a binary executable file, then feed the generated function list, which contains the address of functions and their corresponding PEKO value, to the fuzzer for PEKO power schedule to plan the distribution of energy.

Evaluation: Woff2 issue 609042

The issue 609042 of Woff2 is a Heap-based buffered overflow vulNerability. we can see that the PEKO power schedule performs better than most of the strategies from AFLFast and have similar average time with lin power scheduler from AFLFast.

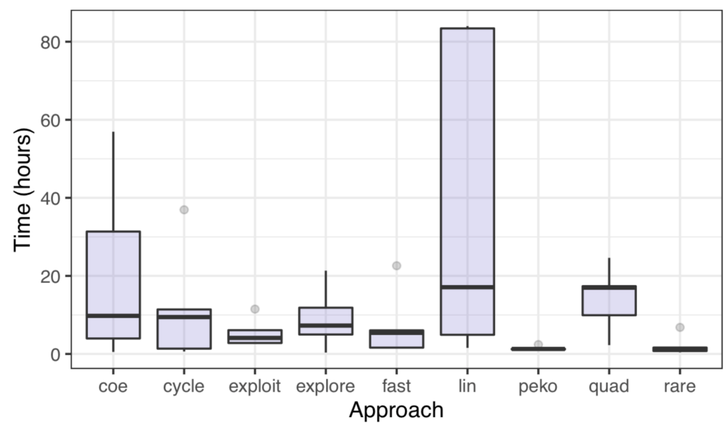




The Woff2 is a static link binary and contains lots of functions, which can be considered as well-tested. Because of that, we use the issue 609042 of Woff2 to show the efficiency of a fuzzer with PEKO power scheduler after fine-tune. it shows that the beta, which is the POKO power schedule with function labeling, has a better average time compare with lin and PEKO without labeling.

Evaluation: CVE-2021-364

The CVE-2021-3648 was found during fuzzing the nm-new of the binutils, the bug is that nm-new stuck in a infinite loop while demangling a malform rust symbol, which cause the nm-new crash due to resource exhausted. We use the minimal time a fuzzer instance needs to get the first hang and the first crash to indicate when a fuzzer instanse triggered the CVE-2021-3648. In the figure we can see that the PEKO power schedule can trigger the bug more earlier than other power schedules in average time.



Summary

This thesis propose a new power schedule base on our new software metric, Procedure Error Key-Omen, aka PEKO which can be intergrated into a fuzzer as a metric for power schedule easily to aid power schedule. And we also implement a PEKO power schedule which leverage the PEKO software metric to distribute the energy among seeds more objectively. Compare with the power schedule, which would try to spent more energy on those path which are rarely explored, that cause it to withdraw the energy from path those are frequently visited. Without taking the potential of a path to find a bug or explore more states into account, that would cause a fuzzer to stop fuzzing on some valuable path thus extending the time to find a bug. What's more, because of the nature of the PEKO power scheduler, a human expert can affect the energy allocation of a fuzzer by just correcting the size of an array in IDA Pro or labeling some function as secure or dangerous, that make the collaboration between a human expert and the PEKO power schedule being continuous, any trivial information that can change the PEKO value is adequate, the fuzzer would find a way to increase the code coverage by those scattered knowledge.