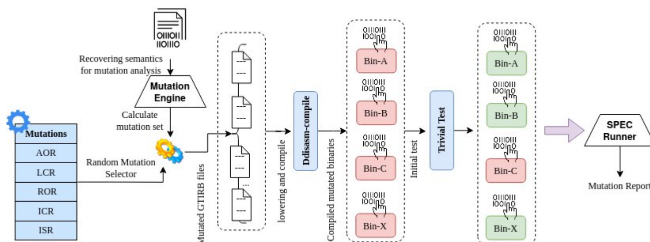
SN4KE: A lightweight and scalable framework for binary mutation testing

8 March 2021, by Ingrid Fadelli



Credit: Ahmadi, Kiaei & Emamdoost.

When developers deliver software to their clients, they often also provide what is known as a 'test suite.' A test suite is a tool that allows users to test software, unveil any bugs it might have and give developers a chance to fix these bugs or other potential issues.

In addition to evaluating software, therefore, developers also need to ascertain the efficacy of a test suite in identifying bugs and errors. One way to run test suite evaluations is via mutation testing, a technique that generates several 'mutants' of a program by slightly modifying its original code. While mutation testing tools have proved to be incredibly helpful, most of them cannot be applied to software that is only available in binary code (a way of representing texts or instructions for computers using two symbols, generally '0' and '1').

Researchers at Arizona State University, Worcester Polytechnic Institute and the University of Minnesota have recently developed SN4KE, a framework that can be used to carry out mutation analyses at a binary level. This framework, presented at the Binary Analysis Research (BAR) NDSS symposium '21 in February, is a new tool to efficiently test suites for software based on binary codes.

"Our work stems from a similar concept in the software testing domain," Mohsen Ahmadi, one of the researchers who carried out the study, told TechXplore. "In our study, we applied source-level mutation operators on closed-source programs using two novel binary rewriting techniques."

Researchers apply so-called 'mutation operators' to generate different variations of an original software program. The ultimate goal of mutation testing methods is to evaluate how well test suits distinguish an original binary code from its variations. When this analysis is complete, a test suite destroys each mutant and generates a 'mutation score," which is essentially the total number of mutants it killed over the total amount of mutants it generated.

"One involved factor in achieving a higher mutation score is related to the reachability of mutated instruction(s), causing an exception that propagates the error to a noticeable change in the program output," Ahmadi said. "The more sections of the code a test suite covers, the higher the odds are for the test suite to detect the mutants."

Ahmadi and his colleagues created a lightweight and scalable binary mutation framework with a rich set of mutation strategies inspired from source-level mutation engines. The main challenge when trying to apply mutations at a binary level is to recover the semantics lost when mutations are compiled.

"In our selection of the right set of rewriting tools, we considered the following factors: 1) architecture-independence, 2) runtime performance, 3) semantic recovery accuracy," Ahmadi said. "Another advantage of our research is that we compare two rewriting schemes; one is based on reassemble-able disassembly, and the other works on top of full-translation. Given our selection criteria, we opted for Ddisasm (a renowned

disassembler) as a candidate that relies on recovering relocatable assembly code and Rev.ng (a tool for binary analysis) for the full-translation."

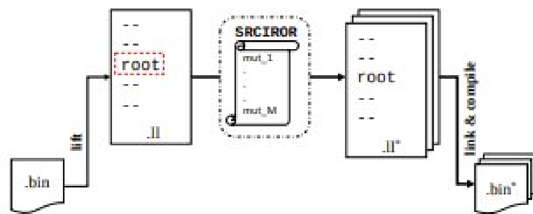


Fig. 2: Integration of Rev.ng and SRCIROR to generate mutant binaries

Credit: Ahmadi, Kiaei & Emamdoost.

In contrast with previously developed mutation testing methods, the framework created by the researchers produces a larger number of mutants, as it has a diverse set of mutation operators. In their experiments, Ahmadi and his colleagues realized that techniques like Rev.ng, which recompile the translated binary code into an intermediate representation, are not suitable for conducting mutation analyses.

"The size of the binaries rewritten by Rev.ng increased up to 70x compared to the baseline," Ahmadi explained. "The reason for this is the inclusion of QEMU's callbacks, used for chaining the translated blocks into resulting binaries. We found that the mutation score was directly related with the number of killed mutants and generally observed a higher mutation score from Ddisasm results compared to Rev.ng and previous works."

So far, the framework for binary mutation testing created by this team of researchers has achieved highly promising results. In the future, it could allow developers and researchers worldwide to evaluate test suites for software programs based on binary codes.

"In our recent paper, we addressed the limitations of binary mutation by employing more robust binary rewriting approaches and adopting a comprehensive set of mutation operations," Ahmadi said. "This work could be extended for proof-testing the patches when there is no access to the source code. One way to approach it is to map the mutation operators to the possible vulnerabilities in a binary. For example, an incorrect replacement of code during a software patch might cause a double-fetch vulnerability due to ambiguity introduced at memory read/write patterns."

More information: SN4KE: Practical mutation testing at binary level. arXiv:2102.05709 [cs.SE]. arxiv.org/abs/2102.05709

Github repository project: github.com/pwnslinger/sn4ke/

www.ndss-symposium.org/ndss-program/bar-2021/

© 2021 Science X Network

APA citation: SN4KE: A lightweight and scalable framework for binary mutation testing (2021, March 8) retrieved 22 October 2021 from https://techxplore.com/news/2021-03-sn4ke-lightweight-scalable-framework-binary.html

This document is subject to copyright. Apart from any fair dealing for the purpose of private study or research, no part may be reproduced without the written permission. The content is provided for information purposes only.