POLITECNICO DI TORINO
Repository ISTITUZIONALE

FAUST: fault-injection script-based tool

Original
FAUST: fault-injection script-based tool / Benso, Alfredo; DI CARLO, Stefano; DI NATALE, Giorgio; Prinetto, Paolo Ernesto; Solcia, Ivano; Tagliaferri, Luca. - STAMPA. - (2003), pp. 160-160. (Intervento presentato al convegno IEEE 9th On-Line Testing Symposium (IOLTS) tenutosi a Kos, GR nel 7-9 July 2003) [10.1109/OLT.2003.1214386].

Availability:
This version is available at: 11583/1416278 since:

Publisher:
IEEE Computer Society

Published
DOI:10.1109/OLT.2003.1214386

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

24 April 2024

FAUST: FAUlt-injection Script-based Tool

A. BENSO, S. DI CARLO, G. DI NATALE, P. PRINETTO, I. SOLCIA, L. TAGLIAFERRI
Politecnico di Torino
Dipartimento di Automatica e Informatica Torino, Italy
{benso, dicarlo, dinatale, prinetto, solcia, tagliaferri}@polito.it

Abstract

The tool described in this paper aims at evaluating the effectiveness of software-implemented fault-tolerant techniques used in safety-critical systems. The target application is stressed with the injection of transient or permanent faults. The user can therefore observe the real behaviour of the application in presence of a fault, and, if necessary, take the appropriate countermeasures. The accent is put on the extreme easiness of use and the portability on all UNIX platforms.

1. Introduction

Dependable software used in human-critical applications (automotive, medical and spatial) should be able to detect whether an error appears, locate its position within the code, and recover from possible consequences [1]. To ensure that these skills are achieved by a fault-tolerant system, several techniques have been proposed during the last years and they are mainly based on analytical models, experimental techniques and fault-injection. The main advantage of fault-injection, with respect to the previous alternatives, is that a realistic test environment can be set up, putting the software under test in a condition that is very close to the normal context of execution [2]. FAUST project lies in this context: this tool has been designed in order to test fault tolerant software, without the need of external hardware or cost effective software. The injector is not related to any specific language or operating system so that a wide number of COTS applications (commercial off-the-shelf) are targeted and can be injected. The portability of our injector is achieved thanks to the use of widely diffused programmer tools, like *GDB* (gnu debugger) [3] and *objdump*, and many bash-shell scripts, whose main purpose is to drive automatically the execution and to generate the appropriate faults. The chosen implementation of FAUST employs two different types of injections: transient errors, modeled as a bit-flip in memory, permanent faults, deputed to emulate hardware stuck-at.

2. FAUST Architecture

The tool deals with applications written in every language supported by GDB and does not require any modification of the target application. From the user point of view, the injector appears as a command line tool whose parameters are:

- injection location (code segment, data segment, stack segment, CPU registers, variables);
- injection type (transient or permanent);
- injection time (random or user chosen);
- number of injections.

Faust exploits the previous parameters to:

- generate the sequence of instructions to load and run the application within the debugger;
- inject the fault;
- collect the results.

For each injection experiments the tool categorizes the program results by comparing them with the correct one previously gathered. The expected result can be of three different types: *no errors* if the injection doesn't affect program execution, *crash* if the program halts and *wrong output* when program provides results different from the expected ones. Then, for each injection, the results are collected in one log-file.

The most important aspects of our approach deal with the high portability of the tool which, thanks to its implementation, can be exported to all the UNIX based systems and to the reduced time overhead employed to perform the injections.

3. References

[1] P.P. Shirvani, N. Oh, E.J. McCluskey, D.L. Wood, M.N. Lovellette, K.S. Wood, *"Software-Implemented Hardware Fault Tolerance Experiments: COTS in Space"* International Conference on Dependable Systems and Networks (FTCS-30 and DCCA-8), New York (NY), 2000, Page(s) B56-57.

[2] F. Faccio, C. Detcheverry, M. Huhtinen CERN, Geneva, Switzerland,*"First evaluation of the Single Event Upset (SEU) risk for electronics in the CMS Experiment"*, CMS NOTE 1998/054 CERN, Geneva, Switzerland.

[3] R. Stallman, R. Pesch, S. Shebs et al., *Debugging with GDB*, June 2002. http://www.gnu.org/software/gdb

Proceedings of the 9th IEEE International On-Line Testing Symposium (IOLTS'03)
0-7695-1968-7/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY