NFTAPE Fault Injector for Win32 Applications

Sandeep Yenugula
Final Year Under Graduate,
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur,
India-721302
ysandeep@iitkgp.ac.in

Cuong Pham
Software Engineer,
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL- 61801
phammanhcuong@gmail.com

Daniel Chen
Research Scholar,
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL- 61801
dchen8@illinois.edu

Zbigniew Kalbarczyk
Principal Research Scientist,
Center for Reliable and High-Performance Computing
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL- 61801
kalbarcz@illinois.edu

Ravi K.Iyer
Director,
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL-61801
rkiyer@illinois.edu

ABSTRACT

Fault injection techniques are widely used for assessing the reliability of applications. NFTAPE is a software tool developed for performing automated fault injections to evaluate the reliability of applications. This poster demonstrates the design and implementation of a fault injector which can be integrated with the NFTAPE framework to support dependability assessment of Windows applications.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]:Fault Tolerance, Performance attributes, Reliability, availability, and serviceability.

General Terms

Performance, Reliability

Keywords

Automated fault injection, Fault Injector, Reliability evaluation.

1. INTRODUCTION

An application which can handle the data corruption and perform efficiently even in the unseen situations is considered reliable. Fault injection techniques are widely used for evaluating the reliability of applications. Networked Fault Tolerance and Performance Evaluator (NFTAPE), developed at the University of Illinois, is a tool that performs automated fault injections and characterizes the application's sensitivity to errors. The key component in the NFTAPE framework [1] that performs the actual injections is called the *fault injector*. A fault injector injects faults in an application to assess application sensitivity to errors and evaluate the efficiency of the error detection and recovery mechanisms within the application. This assists the designer in enhancing the reliability support.

The wide use of the Windows based systems [2], the robust operation of Windows application becomes of paramount importance. In this context, development of a fault injector for Windows applications becomes essential in enabling rapid and accurate evaluation of reliability of applications executing on top of Windows operating system. The fault injector presented in this poster focuses on conducting injections in the user space of the target application.

2. APPROACH

Debugger-based injection method is used to design the injector. The approach is as follows:

- Design and implement a basic application debugger using the APIs [4] provided by Windows operating system.

- Extend the developed simple debugger to support automated fault injection to text, data, and stack memory of an application.
- Make the injector compatible with the NFTAPE architecture and existing injectors for Linux and Solaris systems [3].

3. IMPLEMENTATION

The programming language used for implementation of the fault injector is C++ and it is done in "Visual C++ 2008 Express edition" Environment.

The fault injector can conduct injections in the user space of Windows-based applications. The fault injector allows the user to create breakpoints at a specific address location of the target application and perform the injections when the breakpoint is triggered. The format of the input arguments to the injector is shown in Table 1.

Table 1: Input Format

Process Name	Breakpoint Address	Destination Address	Injection Type	Mask Function	Mask Size	Mask Data	Wait Time (milli sec)
Stack.exe	0x401291	0x404010	Text/Data/ Stack	OverWrite/ Add/BitFlip	1byte- 4byte	Value	Value (>1000)

Argument 1: Name of the Target Process.
Argument 2: Address at which a breakpoint is to be set.
Argument 3: Address at which injection is to be done.
Argument 4: Type of injection (Text/Stack/Data).
Argument 5: How the injection should be done.
1: Data Overwrite
2: Add a value
3: Bit Flip
Argument 6: size of the data to be modified(1–4bytes).
Argument 7: Data to be injected.
Argument 8: The wait-time for triggering the breakpoint.

The fault injector waits for certain time (specified by the user as argument 8) for the breakpoint to get triggered. The breakpoint is deleted if it is not triggered within the specified time. Note that since an exception cannot be caught within the time less than a second, the value passed as argument 8 should not be less than a second. The working of the Windows fault injector is shown in Figure 1.

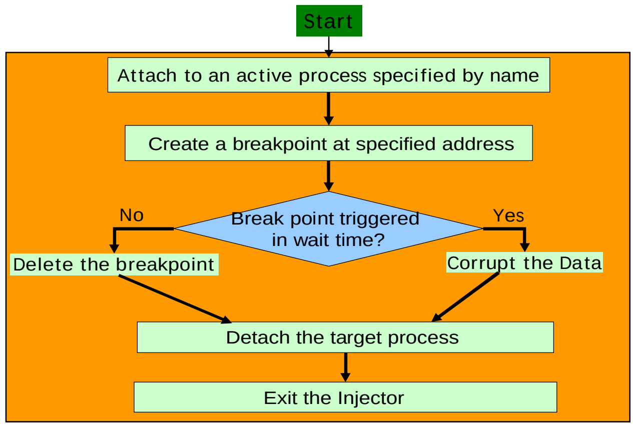

Start
Attach to an active process specified by name
Create a breakpoint at specified address
No
Break point triggered in wait time?
Yes
Delete the breakpoint
Corrupt the Data
Detach the target process
Exit the Injector

Figure 1: Operation of the Fault Injector for Windows Applications

The injections are performed assuming that the user provides the address location within the text/data/stack memory segment of an instruction or data to be corrupted.

Automation of the injection process:
In order to automate the injection process, a software utility is implemented to: (i) read the input from a file which contains the fault injection targets (i.e., the memory locations to be corrupted), (ii) invokes the fault injector, (iii) waits till the injector terminates, (iv) fetch (from the target file) the next target location.

4. RESULTS

The developed fault injector was tested on a benchmark application.

Description of Test Application:
The test application uses three local variables *i*, *j* and *sum* and a global variable *gv*. The *main()* function calls a local function *add()*, which adds the values of *i* and *j* and stores the result in the variable *sum*. The application increments *i* by 5, *j* by 25 and *gv* by 1 during each cycle and then prints the values of *i*, *j*, *sum* and *gv*.

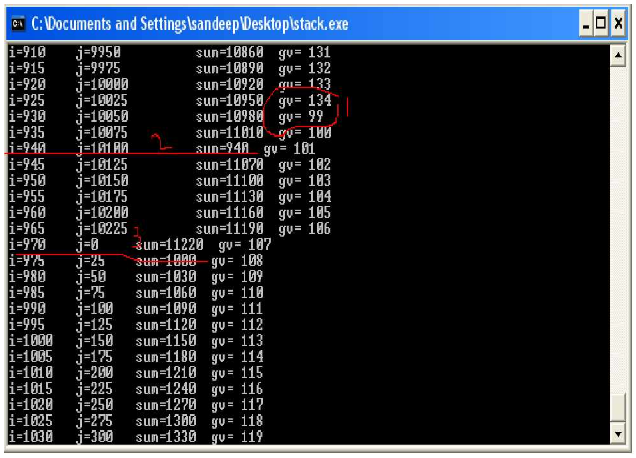
The above described application is tested with the following test cases (see Table 2).

Table 2: Test cases for the injector

Process Name	Breakpoint Address	Destination Address	Injection Type	Mask Function	Mask Size	Mask Data	Wait Time (milli sec)
Stack.exe	0x401291	0x404010	1	1	4	99	1001
Stack.exe	0x401366	0x38	2	1	4	0	3002
Stack.exe	0x40131e	0x34	2	1	4	0	2000

0x404010 is the address of the global variable *gv*.
0x38 is the offset of the local variable *i* of the local function add.
0x34 is the offset of the local variable *j* declared in the main function.

The results obtained by using these test cases on the test application are shown in Figure 2.


C:\Documents and Settings\sandeep\Desktop\stack.exe

```
i=910      j=9950          sum=10860  gv= 131
i=915      j=9975          sum=10890  gv= 132
i=920      j=10000         sum=10920  gv= 133
i=925      j=10025         sum=10950  gv= 134
i=930      j=10050         sum=10980  gv= 99
i=935      j=10075         sum=11010  gv= 100
i=940      j=10100         sum=940  gv= 101
i=945      j=10125         sum=11070  gv= 102
i=950      j=10150         sum=11100  gv= 103
i=955      j=10175         sum=11130  gv= 104
i=960      j=10200         sum=11160  gv= 105
i=965      j=10225         sum=11190  gv= 106
i=970      j=0     sum=11220  gv= 107
i=975      j=25    sum=1000  gv= 108
i=980      j=50    sum=1030  gv= 109
i=985      j=75    sum=1060  gv= 110
i=990      j=100   sum=1090  gv= 111
i=995      j=125   sum=1120  gv= 112
i=1000     j=150   sum=1150  gv= 113
i=1005     j=175   sum=1180  gv= 114
i=1010     j=200   sum=1210  gv= 115
i=1015     j=225   sum=1240  gv= 116
i=1020     j=250   sum=1270  gv= 117
i=1025     j=275   sum=1300  gv= 118
i=1030     j=300   sum=1330  gv= 119
```

Figure 2 : Results

- With the first injection (test case 1: Table 2) the value of *gv* is changed to 99 from 134.
- With the second injection (test case 2: Table 2) the value of variable *j* is changed locally in the function *add()* to *0* and hence, the variable *sum* prints the value of *i*. As the data changed is local to function *add(),* it does not affect the next addition.
- With the third injection (test case 3: Table 2) the value of variable *j* is changed in the *main()* function from 10225 to 0 and as the change is made in main, it is persistent.

5. CONCLUSION

This poster demonstrated the implementation of a fault injector that can be used for assessing the reliability of Windows applications. The fault injector can perform injections to user space of an application. The prototype is tested on a benchmark application. The target application can be hanged by corrupting with an invalid instruction. This injector can be used on any Window application for assessing the reliability, but the user needs to ensure that the breakpoint address and the destination address specified are valid. This research can be extended to enhance the fault injector to perform injections in the kernel space of Windows systems.

6. REFERENCES

[1] D. Stott, B. Floering, D. Burke, Z. Kalbarczyk and R.K. Iyer, “NFTAPE: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors,” in Proc. of 4th Int. Computer Performance and Dependability Symposium, IPDS’00, pp.91-100, 2000.

[2] Percentage of usage of operating systems: http://commons.wikimedia.org/wiki/File:Operating_system_usage_share.svg.

[3] D. Chen, G. Jacques-Silva, Z. Kalbarczyk, R. K. Iyer, B. Mealey: “Error Behavior Comparison of Multiple Computing Systems: A Case Study Using Linux on Pentium, Solaris on SPARC, and AIX on POWER”, in Proceedings of 14th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 339-346, 2008.

[4] MSDN Library: Debugging Functions: http://msdn.microsoft.com/en-us/library/ms679303(VS.85).aspx