RUGRAT: Runtime test case generation using dynamic compilers

Amanda Crowell
CMSC737
November 10, 2009

Outline

- Background
- Related Work
- RUGRAT Description
- Targeted Code
- Research Goals
- Experiment/Case Studies
- Results
- Summary

The Goal

Automated methods that test resource error handling & dynamic security mechanisms

- Memory allocation errors
- File I/O problems
- Execution protection mechanisms
- Other types of error handling (EH)

Other testing methods not adequate

The Problem

Resource errors:

- Require notification from OS
- Hard to imitate
    - How do you eat up all of the Memory?
    - How do you deny access of a program to its files and resources?

Dynamic Security mechanisms:

- Require carefully crafted input
- Require vulnerable test programs

The Competition

Compiler-guided exception raising

- Attempt to force execution of EH code
- Compiler inserts extra (source) code

Probabilistic approaches

- Randomly change values in memory
- May not trigger all EH code

The Competition cont.

Additional techniques

- Replace library functions with stub functions
- Modify library memory image to return different values
- Aspect Oriented Programming to wrap code and cause execution of EH code

Problems:

- Require tester to supply more code
- Lacks fine-grained control

The RUGRAT solution

- Automated approach
- Generates tests for EH code
- Takes binary code as input
- Independent of source language
- Uses dynamic compiler to alter executing instructions
  - Doesn’t require changes to source
  - Dynamic compiler only required during testing
  - Can work with any dynamic compiler
- Requires (simple) test specification

The Pieces

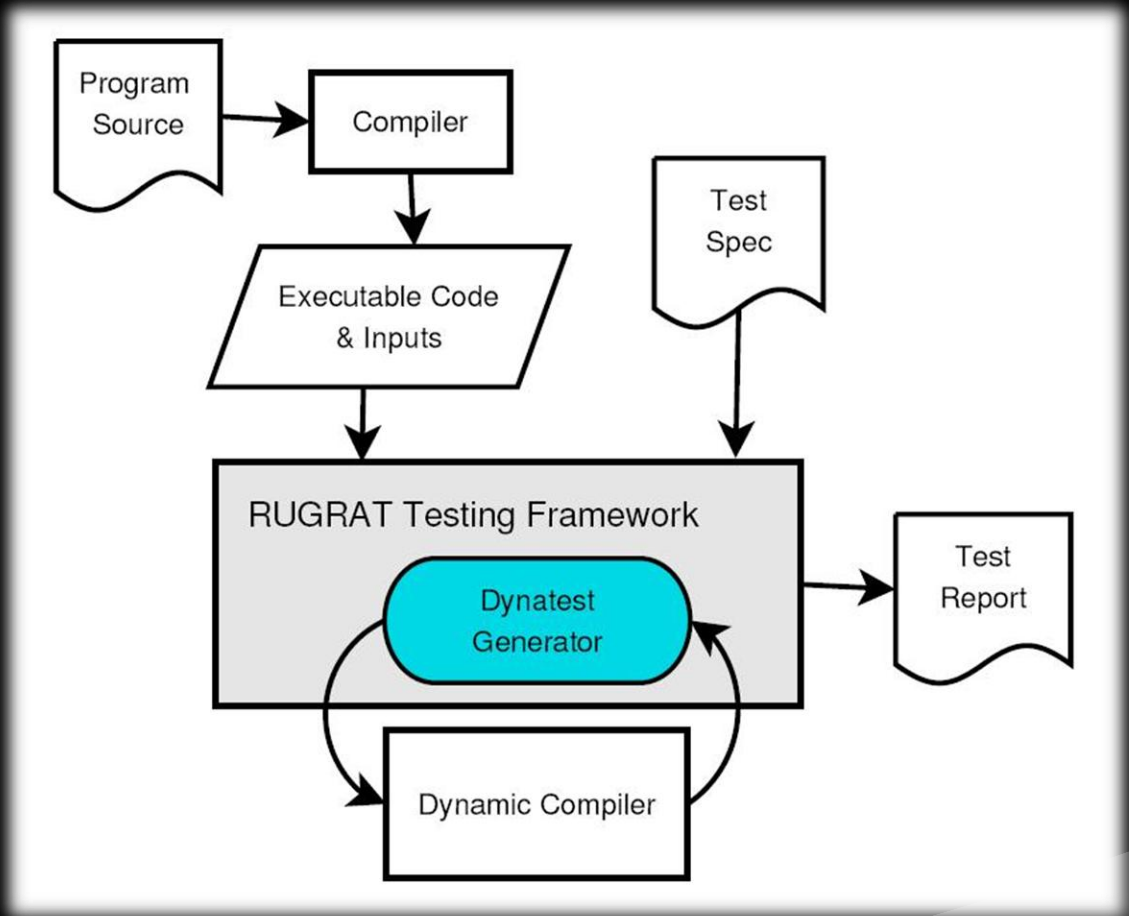
Dynatests

- Runtime tests generated by RUGRAT
- Consists of executable instructions

Test Specification

- Types of dynatests to create
- Oracles for the tests
- Points of program to target
  - “All call sites”
  - “Call to X in function Y”

The RUGRAT Framework



The Targets

Resource Error Handling code

```
if ((sptr = malloc(size+1)) == NULL) {
    findmem();
    if ((sptr = malloc(size+1)) == NULL)
      xlfail("insufficient string space");
}
```

- Goal: want to test findmem()
- Problem: how to make malloc() return null

The Targets cont.

Call	Success	Error	errno vals	Environmental condition
malloc	non-NULL pointer	NULL	ENOMEM	out of memory
socket	int > 0	-1	EACCESS, and others	no socket available
fork	int ≥ 0	-1	EAGAIN ENOMEM	process copy failed
read write	int ≥ 0	-1	EIO and others	IO error
open	int > 0	-1	EACCESS and others	file opening failed
fopen	file pointer	NULL	EACCESS and others	file opening failed

Table 1: Example targeted system calls

The Targets cont.

Function Pointer Protection (FPP) Mechanisms

```
work = unzip;
// stmts that may change work
for (;;) {
   if ((*work)(ifd, ofd) != OK){
// other non-pointer code removed
```

(a) C code with indirect function call

```
movl   $unzip, work
// ...
movl    work, %ecx
// setup arguments
call     *%ecx
```

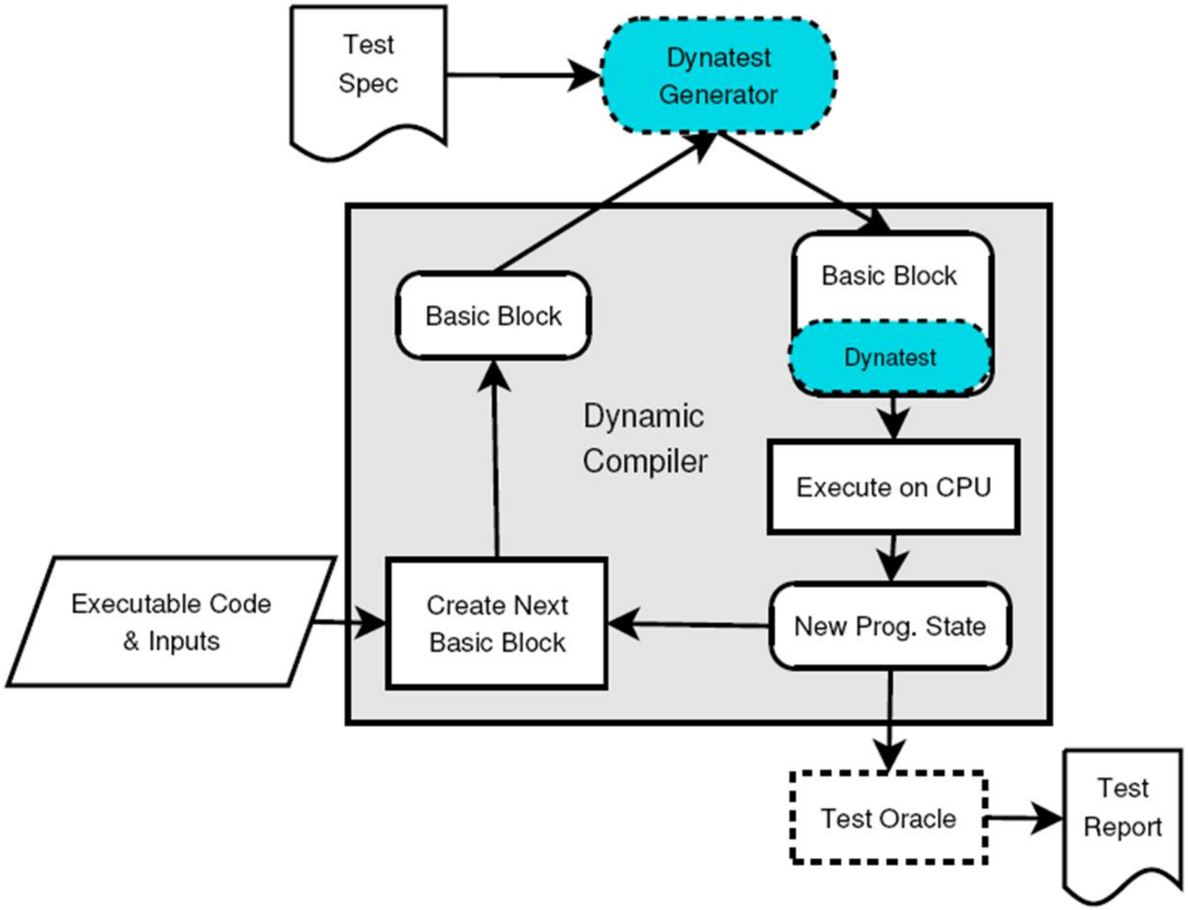
(b) Unprotected assembly code

```
movl    $unzip, %edx
xorl    _mech_key, %edx
movl    %edx, work
// ...
movl    work, %ecx
xorl    _mech_key, %ecx
// setup arguments
call    *%ecx
```

(c) Assembly code with security mechanism

- Goal: test if encrypting the address is effective
- Problem: difficulty in testing

The Architecture



The Research Questions

Testing Error Handling Code

Question	Answer
How much EH code actually requires specific external environment conditions to trigger execution?	Lines of code (LOC) that handles a particular error condition
How effective is RUGRAT in generating tests to cover EH code?	Measure of EH code coverage when using RUGRAT
How much of the EH code covered by RUGRAT is covered without RUGRAT?	Comparison of coverage with RUGRAT to coverage without
How effective is RUGRAT in detecting failures in EH code execution?	Case study of the RUGRAT failure detection capability

The Research Questions cont.

Testing function pointer protection mechanisms

Question	Answer
How effective is RUGRAT at systematically generating test cases to cover the targeted FPP mechanism?	Measures: (1) how well RUGRAT recognizes the FPP test points (2) how many test cases for the test points can be generated automatically
How effective is RUGRAT in exposing faults in the FPP mechanisms?	Case study of RUGRAT’s capability to expose fault in FPP mechanisms

The Subjects

- Criteria
  - programs that use the most function pointers

    and/or
  - contain the most EH code
- Selections
  - SPEC
  - MiBench
  - `space` application

The Validity Threats

- Choice of Dynamic Compiler (DynamoRIO)
  - Dependent on how it analyzes and programs it accepts
  - Solution – use PIN (another dynamic compiler) to verify analysis
- Choice of FPP Mechanism
  - Developed their own, based on PointGuard
- Application size
  - Limited to small-to-medium sized applications
  - Affects generalization

The EH Results

Program/function	EH call sites	Total EH stmts	# rugrat covrd	# w/o rugrat covrd	% incr covrd
		RQ1	RQ2	RQ3	
space/malloc	13/14	45	44	13	69
space/fopen	1	3	3	1	67
130.li/malloc	3	81	68	52	20
164.gzip/write	1	25	17	1	64
omission: boxed-sim/malloc	71	0	–	–	–
lout/malloc†	27	87	56	19	43
lout/OpenFile	4	42	22	6	38

† lout has 8 `malloc` callsites with no EH code associated.

C2 vs C3 :

- DynamoRIO seg fault
- EH options not covered by RUGRAT testcases
- Dead code

Omission discovery:

- Call sites that had no associated EH code

The FPP Results

- All possible function pointer call sites identified
- Each test passed
  - detected the attack
- Did not require vulnerable program
  - found all places protection mechanism *should* be applied
- Problem
  - Targets particular call site, but not different invocations
  - No contextual information provided by DynamoRIO

Program	# unique fptr callsites = # callsites tested
132.ijpeg	126
008.espresso	10
130.li	3
147.vortex	7
164.gzip	2
Total	148

The EH Case Study

- Subject – `space` application
- 100 randomly generated test cases
- Found 20 “interesting” EH points
  - The IF statement was covered, body was not
  - EH code attempts to handle error, not print and quit
- Verified RUGRAT forced EH code to execute – saved as expected results

The EH Case Study cont.

- 34 faults seeded by graduate students in the IF bodies
- Obtained estimated # of random tests necessary to cover 20 EH code points
  - Kept choosing 100 random test cases until covered
  - Estimated 1600 needed
- Compared expected to actual results
- RUGRAT detected 15 out of 34 faults

The EH Case Study cont.

- Reasons for 19 unexposed faults
  - 8 – `space` corrected for the fault by repeating proper assignments
  - 6 – affected return value & callers only checked for non-zero returns
  - 2 – too little memory allocated & `space` didn't notice
  - 1 – caused program to quit
  - 2 – unknown

The FPP Case Study

- Seeded a fault in their FPP mechanism
  - 5% of the time, generated key = identity key
- Applied to `132.ijpeg`
- Run for all 126 call sites
- Identity key generated 3 times
- RUGRAT detected all attacks

The Good

- Effective at generating test cases for under tested EH code
  - Easier to create resource errors
  - No change to source code required
  - Test specifications easy to create
- Effective at testing FPP mechanisms
  - Hard to test otherwise
  - Doesn’t require vulnerable code
- Mostly automated process

The Not-so-Good

- RUGRAT
  - New instantiations of RUGRAT needed for different kinds of targets
- Paper/Experiment
  - Limited timing data given (Runtime with testing vs runtime without)
  - No information given on time needed to develop test specifications
  - What does a test spec look like?
  - Required test case estimation method

The End!