Challenging Malicious Inputs with Fault Tolerance Techniques

Bruno Luiz
Agenda

• Threats
• Fault Tolerance
• Fault Injection for Fault Tolerance Assessment
• Basic and classic techniques
• Decision Mechanisms
• Implementation Methodology
Threats

- Fault is the identified or hypothesized cause of an error.
- An error is part of the system state that is liable to lead to a failure.
- A failure occurs when the service delivered by the system deviates from the specified service, otherwise termed an incorrect result.
The Classes of Faults

Development Faults
- Operational Faults
- Internal Faults
- External Faults
- Natural Faults
- Human-Made Faults
- Hardware Faults
- Software Faults
- Non-Malicious Faults
- Malicious Faults
- Non-Deliberate Faults
- Deliberate Faults
- Accidental Faults
- Incompetence Faults
- Permanent Faults
- Transient Faults

Physical Faults

Interaction Faults

Examples:
- Software Flaws
- Logic Bombs
- Hardware Errata
- Production Defects
- Physical Deterioration
- Physical Interference
- Intrusion Attempts
- Viruses & Worms
- Input Mistakes
Tree Representation of Faults

- Phase of creation or occurrence
  - Development
  - Operational
- System boundaries
  - Internal
  - External
- Phenomenological cause
  - Human-made
  - Natural
- Dimension
  - Software
  - Hardware
  - Hardw
- Objective
  - Non Malicious
  - Mal
- Intent
  - Non Del
  - Del
- Capacity
  - Acc
  - Inc
- Persistence
  - Per
  - Tr

Development Faults
Physical Faults
Interaction Faults

Mal: Malicious  Del: Deliberate  Acc: Accidental  Inc: Incompetence  Per: Permanent  Tr: Transient
Objective

• Malicious faults are introduced during either system development with the intent to cause harm to the system
  - They are grouped into two classes
    • Potentially harmful components
      - Trojan horses
      - Trapdoors
      - Logic or Timing bombs
    • Deliberately introduced software or hardware
      - Vulnerabilities or human-made faults
  • Non-malicious faults are introduced without malicious objectives
    - Vulnerabilities
Malicious Logic Faults

- That encompass development faults
  - Logic Bomb
  - Trojan horse
  - Trapdoor
- Operational faults
  - Virus
  - Worm
  - Zombie
Intrusion Attempts

• Malicious Inputs
  - To disrupt or halt service
  - To access confidential information
  - To improperly modify the system
Vulnerabilities

- Development or operational faults
- Common feature of interaction faults
- Malicious or non-malicious faults
- Can be external fault that exploit them
Fault Tolerance

“The goal of fault tolerance methods is to include safety features in the software design or Source Code to ensure that the software will respond correctly to input data errors and prevent output and control errors”

Software faults are what we commonly call "bugs"
Fault Tolerance

- Can, in principle, be applied at any level in a software system
  - Procedure
  - Process
  - Full application program
  - The whole system including the operating system
- Economical and effective means to increase the level of fault tolerance in application
  - Watchd
  - libft
  - REPL
Error Detection and Correction

- Verification tests capable of detection of the errors
  - Replication
  - Temporal
  - Consistency
  - Diagnosis
- Once the error has been detected, the next step will be your elimination
  - Backward Recovery
  - Forward Recovery
Backward Recovery

- Checkpoint
- Restore checkpoint
- Fault detected
- Rollback
- Recovery
  - Recovery point
  - Fault
  - Tolerance
Forward Recovery

Fault detection and handling

Fault tolerated

Recovery point
Redundancy

- Types of Redundancy for Software Fault Tolerance
  - Software Redundancy
  - Information or Data Redundancy
  - Temporal Redundancy
- The selection of which type of redundancy to use is dependent on the...
  - Application’s requirements
  - Resources
  - Techniques
Robust Software

• Defined as “the extent to which software can continue to operate correctly despite the introduction of invalid inputs”
  - Out of range inputs
  - Inputs of the wrong type
  - Inputs in the wrong format
• Self-checking software features
  - Testing the input data
  - Testing the control sequences
  - Testing the function of the process
Diversity

• Since redundancy alone is not sufficient to help detect and tolerate software design faults
• This diversity can be applied at several levels and in several forms
• Forms of diversity
  - Design diversity
  - Data diversity
  - Temporal diversity
Basic Design Diversity

Input

Variant 1

Variant 2

Variant 3

Decider

Incorrect

Correct
Data Diversity

• To avoid anomalous areas in the input data space that cause faults
• Use data re-expression algorithms (DRAs) to obtain their input data
• Depends on the performance of the re-expression algorithm used
  - Input Data Re-Expression
  - Input Re-Expression with Post-Execution Adjustment
  - Re-Expression via Decomposition and Recombination
Overview of Data Re-Expression

• A re-expression algorithm, R, transforms the original input x to produce the new input, \( y = R(x) \)

• The input y may either approximate x or contain x’s information in a different form
Data Re-Expression With Postexecution Adjustment

• A correction, A, is performed on P(y) to undo the distortion produced by the re-expression algorithm, R

• This approach allows major changes to the inputs
Data Re-Expression via Decomposition and Recombination

- An input $x$ is decomposed into a related set of inputs
- Results are then recombined

\[ x \rightarrow x_1, \ldots, x_n \]

\[ P(x_1) \]

\[ P(x_2) \]

\[ \ldots \]

\[ P(x_n) \]

\[ P(x_i) \]

\[ F(P(x_i)) \]
Fault Injection for Fault Tolerance Assessment

- Injecting faults enables a performance estimate for the fault tolerance mechanisms
  - Fuzzing
    - Latency (the time from fault occurrence to error manifestation at the observation point)
  - Exploit vulnerability
    - Coverage (faults handled properly)
Fault Injection for Fault Tolerance Assessment

• Advantages of Fault Injection using fuzzing
  - Accelerating the failure rate
  - Able to better understand the behavior of that mechanism

• Error propagation
• Output response characteristics
Fault Injection for Fault Tolerance Assessment

- Advantages of Fault Injection using exploration
  - Saving and restoring the execution context
  - Integrity of the data during execution
  - Test backward recovery
Programming Techniques

- Assertions
- Checkpointing
- Atomic actions
Assertions

• Are a fairly common means of program validation and error detection
• In essence, they check whether a current program state to determine if it is corrupt by testing for out-of-range variable values
• Simplest form

if not assertion then action
Assertions

• Several modern programming languages include an assertion statement
• When an error does occur it is detected immediately and directly, rather than later through its often obscure side-effects

```c
int *ptr = malloc(sizeof(int) * 10);
assert(ptr != NULL);
// use ptr
```
Assertions

• Simplify debugging
• Checked at runtime

```java
int total = countNumberOfUsers();
if (total % 2 == 0)
{
    // total is even
} else
{
    // total is odd
    assert(total % 2 == 1);
}
```
Checkpointing

- Is used in error recovery, which we recall restores a previously saved state of the system when a failure is detected.
- Saves a complete copy of the state when a recovery point is established.
- The information saved by checkpoints includes:
  - Values of variables in the process
  - Environment
  - Control information
  - Register values
Checkpointing

- Complex mechanism of restoring the stack and register state of the checkpointed process
- Save the state of data in memory, the processor context (register and instruction pointer) and the stack
  - User-level
  - Kernel-level
Checkpointing

- Methods
  - Internal
    - Only be used by the process being checkpointed
    - Insert some code into the process to be checkpointed
  - External
    - May be used by any process
    - Examine the information published by the kernel through the /proc
Checkpointing

• Types
  - Static
    • Gathering kernel state information
    • Information can be acquired more or less directly from the kernel
  - Dynamic
    • Track all operations by a process
    • Replace C library functions with wrappers

• Existing systems
  - libckpt
  - condor
  - hector
  - icee
  - EPCKPT
  - CHPOX
Atomic Actions

• Are used for error recovery
• An atomic action is an action that is
  - Indivisible
  - Serializable
  - Recoverable
Basic and Classic Techniques

- Recovery Blocks
- N-Version Programming
- Retry Blocks
- N-Copy Programming
Recovery Blocks

- Dynamic technique
- Uses an AT and backward recovery
- RcB scheme
  - Executive
  - Acceptance test
  - Primary and alternate blocks (variants)
  - Watchdog timer (WDT)
Recovery Block Operation

- General Syntax

```plaintext
ensure by Acceptance Test
by Primary Alternate
else by Alternate 2
else by Alternate 3
... Alternate n
else by
else failure exception
```
Recovery Block Operation

- **RcB entry**
  - Establish checkpoint
  - Execute alternate
  - Evaluate AT
  - Pass
    - Discard checkpoint
  - Fail
    - Restore checkpoint
  - Exception signals
- **New alternate exists and deadline not expired?**
  - Yes
    - Evaluate AT
  - No
    - Restore checkpoint
    - Failure exception
- **RcB exit**
N-Version Programming

• Static technique
• Use a decision mechanism (DM) and forward recovery
• NVP technique consists
  - Executive
  - n variants
  - DM
N-Version Programming Operation

- General Syntax

Run Version 1, Version 2, ..., Version n
if (Decision Mechanism (Result 1, Result 2, ..., Result n))
    return Result
else failure exception
N-Version Programming Operation

NVP entry -> Distribute inputs -> Version 1 -> Version 2 -> ... -> Version n -> Gather results -> DM

DM: Exception raised

Output selected -> NVP exit

Failure exception -> NVP exit
Retry Blocks

• RtB technique is the data diverse complement of the recovery block (RcB) scheme

• RtB technique consists
  - Executive
  - AT
  - DRA
  - WDT
  - Primary and backup algorithms
Retry Block Operation

Ensure Acceptance Test
by Primary Algorithm (Original Input)
else by Primary Algorithm (Re-expressed Input)
else by Primary Algorithm (Re-expressed Input)
...
...
[Deadline Expires]
else by Backup Algorithm (Original Input)
else failure exception
Retry Block Operation

RtB entry

Establish checkpoint

Execute algorithm

Evaluate AT

Exception signals

Pass

Discard checkpoint

RtB exit

Yes

New DRA exists and deadline not expired?

No

Invoke backup

Pass

Evaluate AT for backup

Fail

Failure exception

RtB exit

Yes

Restore checkpoint

Pass

Evaluate AT for backup

Fail
N-Copy Programming

- NCP is the data diverse complement of N-version programming (NVP)
- Copies execute in parallel using the re-expressed data as input
- NCP technique consists
  - Executive
  - 1 to n DRA
  - n copies of the program or function
  - DM
N-Copy Programming Operation

• General Syntax

```
run DRA 1, DRA 2, ..., DRA n
Run Copy 1(result of DRA 1),
    Copy 2(result of DRA 2), ..., 
    Copy n(result of DRA n)
if (Decision Mechanism (Result 1, Result 2, ..., 
    Result n))
    return Result
else failure exception
```
N-Copy Programming Operation

NCP entry

- Distribute inputs
  - DRA 1
    - Copy 1
  - DRA 2
    - Copy 2
  - ... (DRA n)
    - Copy n

- Gather results

- DM

- Exception raised
  - Output selected

- NVP exit
  - Failure exception
Decision Mechanisms

• Adjudicators determine if a “correct” result is produced by a technique
• Adjudicator would run its decision-making algorithm on the result
• Adjudicators generally come in two flavors
  - Voters
  - ATs
Adjudicator

• Acceptance Tests (ATs)
  - Reasonableness tests
  - Computer run-time tests
Acceptance Tests

- Basic approach to self-checking software

1. Variant input
2. Receive variant result
3. Apply AT
4. Set pass/fail indicator (TRUE/FALSE)
5. Return status
Reasonableness Tests

- Determine if the state of an object in the system is reasonable
  - Precomputed ranges
  - Expected sequences of program states
  - Other expected relationships
Range Bounds AT

• General Syntax

BoundsAT (input, Min, Max, Status)

Set Status = NIL
Receive algorithm result (input)
Retrieve bounds (Min < and < Max)

if input is within bounds (i.e., Min < input < Max)
then Set Status = TRUE
else Set Status = FALSE (Exception)
End

Return Status
Range Bounds AT Operation

Variant input

Set status = NIL

Receive variant result, $r$

Min < $r$ < Max ?

No

Set status = FALSE

Yes

Set status = TRUE

Return status

No

Set status = FALSE

Bounds AT
Computer Run-Time Tests

• Test only for anomalous states
• Detect anomalous states such as
  - Divide-by-zero
  - Overflow
  - Underflow
  - Undefined operation code
  - Write-protection violations
Recovering Exploration

• The recovering exploration technique uses RcB to accomplish fault tolerance.
• When a checkpoint is established the values of data in memory, the processor context (register and instruction pointer) and the stack are saved.
• Time-out via the watchdog occurs, resets the watchdog time, and restores the checkpoint.
Recovering Exploration

Malicious Input

RcB

Checkpoint

Vulnerability

Program

Malicious Code

WDT
Anti-Fuzzing

- Technique to prevent hacker discover zero day vulnerabilities in vendors
- The inputs are distributed for the modules and in case the results are distinct a error is detected.
- Use N-Version Programming in which each version is an module.
Anti-Fuzzing

Malicious Input

Distribute inputs

Version 1: negative

Version 2: range

Version 3: anomalies

Decision Mechanism

Program

Vulnerability
Implementation Methodology

1. It is defined an initial architecture and a technique for your implementation
2. They identify the classes of susceptible to flaws to happen, and that should be tolerated
3. They incorporate the mechanisms of detection of errors, necessary to the attendance of all the classes of important flaws
4. Recovery algorithms are defined that will be worked after the greeting of the originating from sign the detection mechanisms
Questions?

Bruno Luiz
brunolcr@yahoo.com.br