Software Fault Injection for Survivability

Jeffrey M. Voas & Anup K. Ghosh

Presented by Alison Teoh
Goals of Software Testing

• Correctness
• Reliability
• Usability
• Robustness
• Performance
Goals of Software Testing

- Correctness
- Reliability
- Usability
- Robustness
- Performance
Goals of Software Testing

- Correctness
- Reliability
- Usability
- Robustness
- Performance
Outline

• Basic definitions and Testing Technique Overview
• Algorithm for Fault Injection Analysis
• Fault Injection Security Tool (FIST)
• Interface Propagation Analysis (IPA)
• Conclusions
Some Basic Definitions

Information Survivability: “The ability of a system to continue to operate in the presence of faults, anomalous system behaviour, or malicious attack.”

Fault Injection: “The process of perturbing program behaviour by corrupting a program state during program execution.”
Three Primary Threats to Survivability:

- Software Flaws
- Malicious Attacks
- Anomalous Behaviour of Third Party Software
Three Primary Threats to Survivability:

• **Software Flaws**
  – We don’t know where the actual errors are
  – Simulate random flaws

• **Malicious Attacks**
  – Subject software to well-known attacks

• **Anomalous Behaviour of Third Party Software**
  – Libraries and COTS components may be flawed
  – Simulate component failure
Algorithm

\[ P = \text{Program under analysis} \]
\[ S = \text{State of the system} \]
\[ x = \text{Input value} \]
\[ l = \text{Location in } P \]
\[ PRED = \text{Security violation predicate (assertion)} \]
\[ \text{for } P \text{ and } S \]
Algorithm

1 – Execute P on selected input x
2 – Instrument code to determine each l in P that is exercised by x.
3 – Determine the outcome of an unperturbed run of P
4 – Alter some variable at location l (inject a fault)
5 – If security predicate (assertion) was violated, record location l
6 – Repeat steps 1-5 until coverage goals met
7 – Use recorded locations in code as basis of further analysis (code inspection, verification, etc)
FIST (Fault Injection Security Tool)

• Implementation of fault injection analysis algorithm
• C/C++
• Allows developer to:
  – Randomly perturb program states
  – Append or truncate strings
  – Attempt Buffer Overflows
  – Perform other fault injection functions
FIST

Adaptive Vulnerability Analysis

Fault Injection Engine
- buffer overflow
- data corruption
- string manipulation
- fault composition

Program Inputs
- strings and other variables
- server commands
- configuration files
- network traffic

Instrumented

Security Policy Assertion

Statistical Collection

Relative Security Metrics

System State

Vulnerability Knowledge
FIST

• Miscellaneous Reasons FIST is effective:
  – Always attempts to overflow buffers
    • Most tools only target specific, vulnerable functions
    • StackGuard, Fuzz
  – Allows users to specify “security violations” for individual applications under analysis
    • Choose from predefined assertions
    • Create your own assertions based on any C expression
  – Capable of external assertion monitoring
FIST

• FIST Analysis was performed over a variety of network service daemons
• Several potentially exploitable locations were identified
• Security violation identified in WU-FTPD was later independently discovered and reported by CERT-CC
IPA (Interface Propagation Analysis)

- Simulates component/subsystem failures
- Start from worst case assumptions, observe system-wide effects
- Unit performance is unimportant unless it affects the integrity of the entire system
IPA

IPA uses two fault injection algorithms:

• Propagation From

• Propagation Across
IPA

Propagation From

• Corrupts data exiting a component to observe the types of system failures that ensue.

• Provides information regarding semantic interactions between components as a measure of tolerance
IPA

Propagation Across

• Corrupts data entering a component
• Simulates input failure to gauge component’s robustness
• Mimic human operator errors, hardware failures, or failures from other subsystems
Conclusions

• Fault Injection Analysis can be used in an unconventional way to test survivability in several different scenarios:
  – Software flaws in program source code
  – Malicious attacks
  – Anomalous behaviour from third party software

• By identifying problem components and functions automatically, drastically reduce areas that require manual analysis
Questions?