

# Towards Building Repairable Systems

*Prof. Tzi-cker Chiueh*

*Experimental Computer Systems Lab*

*Stony Brook University*

*chiueh@cs.sunysb.edu*

# Introduction

- There are no such things as unbreakable systems
- What is the next best thing?
- Reliability vs. Availability
  - ◆  $\text{Availability} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$
- Security vs. Intrusion Tolerance
  - ◆  $\text{MTTB} \rightarrow \text{infinity} \rightarrow \text{Unbreakable systems}$
  - ◆  $\text{MTTR} \rightarrow 0 \rightarrow \text{Repairable systems}$

# After an attack occurs, now what?

- Restore the compromised system back to normal order as quickly as possible → Fast reparability
- Make sure the same attack cannot happen again → Attack signature generation
- Fix the underlying problem → Patch creation
- Find out who does it → Digital forensics

# Reparability vs. Recoverability

- Repair is different from recovery

Example: An input error or intrusion took place 24 hours ago, how to selectively undo the effects of those and only those transactions affected?

- Recovery: restore the database to the state 24 hours ago → lose all good transactions in between
- Repair: automatically undo those and only those that are affected by the error/intrusion
- Back-up or mirroring won't help
- Key technical issue: how to minimize collateral damage for non-fail-stop failures

# Fast Reparability

- Key Challenges
  - ◆ Comprehensive versioning
  - ◆ Inter-transaction dependency tracking
  - ◆ Repair-time recovery
  - ◆ Transparency to existing IT infrastructure
- Examples: Both RDB and RFS work are funded by NSF
  - ◆ RDB can quickly repair a damaged DBMS server → works for Oracle and Sybase DBMS server, < 10% overhead
  - ◆ RFS can quickly repair a damaged NFS server → works for NFS server, negligible throughput degradation

# Project Idea 1

- How to build a repairable three-tier internet services and web services
  - ◆ Fast reparability is end-to-end property
  - ◆ Based on RFS and RDB
  - ◆ Require a dynamic information flow tracking mechanism
    - ★ GIFT is a general information flow tracking framework for C/C++ programs
    - ★ Provide both manageability as well as reparability

# Project Idea 2

- Portable and efficient continuous data protection (CDP) → every update is undoable
  - ◆ At the NFS/CIFS interface
  - ◆ At the SQL interface
  - ◆ At the file system call interface
  - ◆ At the iSCSI interface

# Automatic Attack Signature/Patch Generation (NSF)

- Goal: Automate or at least facilitate signature and patch generation process
- Once a single attack is detected, derive an accurate signature for the attack and its variants
  - ◆ Active honeypots that automatically learn new attack signatures and preemptively feed them to firewall
- Identify the nature and location of vulnerability and provide a fix
  - ◆ Need to exploit vulnerability type-specific knowledge

# Attack Signature Requirements

- Low fault positive/negative rate
  - ◆ Focus only on relevant bytes
  - ◆ Context-aware to accommodate multiple-packet attacks
  - ◆ Abstract constraint, e.g., length constraint
- Applicable to multi-process distributed application
- Regardless of traffic compression/encryption/encoding
- Usable by network-based IDS and/or firewall

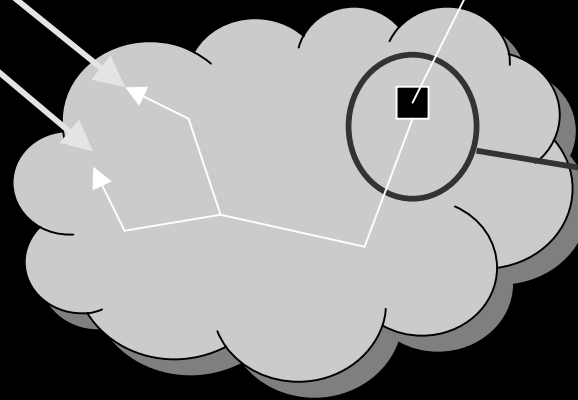
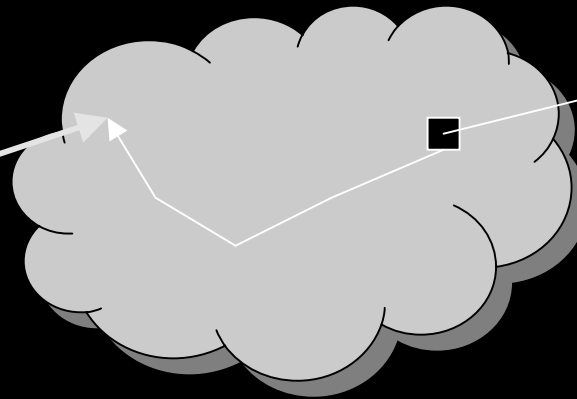
*Dynamic Data Flow Graph*

*Corrupted  
target address*



*Dynamic Control Flow Graph*

*Patch  
Generation*



# Project Idea 3

- How to generate attack signature from a software patch
- Two key challenges:
  - ◆ Identify vulnerability from original and patch
    - Graph-level diff
  - ◆ Compute the minimal portion of the original program that is related to the vulnerability
    - ★ Abstract slicing

# Project Idea 4

- Signature/patch generation for malicious mobile code on Windows
- Technical Challenges:
  - ◆ Comprehensive system event logging
  - ◆ Dependency/causality analysis
  - ◆ Deduce root cause (“downloaded Word document contains VB script”) and derive fix (“such Word instance should be sand-boxed”)