An Experimental Study of File Permission Vulnerabilities Caused by Single-Bit Errors in the SELinux Kernel Policy File

Tom Brown and Michael Ihde

Final Presentation

ECE442 Spring '04

Quick Review of Project Goal

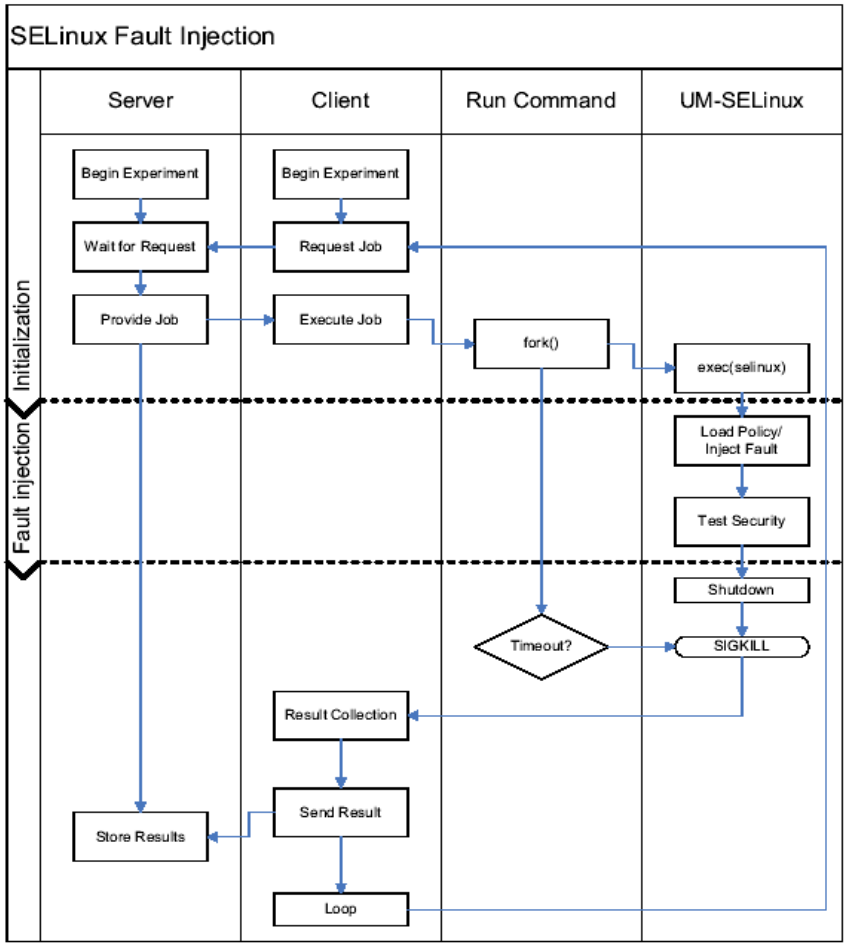
- Determine if SELinux is vulnerable to single-bit errors
  - First study focuses on errors in policy file
- 4Mbit DRAM suffers approx. 6000 FIT [1]
- A system with 1GB of standard ECC will still experience 3435 FIT [2]
  - Equivalent to 900 errors in 10000 machines over 3 years
  - Google uses at least 15,000 commodity machines
    - 1% fault vulnerability rate would create 13 vulnerabilites in 3 years

Quick Review of SELinux

- Provides Mandatory Access Controls to Linux through the Linux Security Module framework
  - Regular user/file permission is Discretionary Access Control
    - Owner of file/process can grant permissions to others
    - Root user is all powerful with Discretionary Access Control

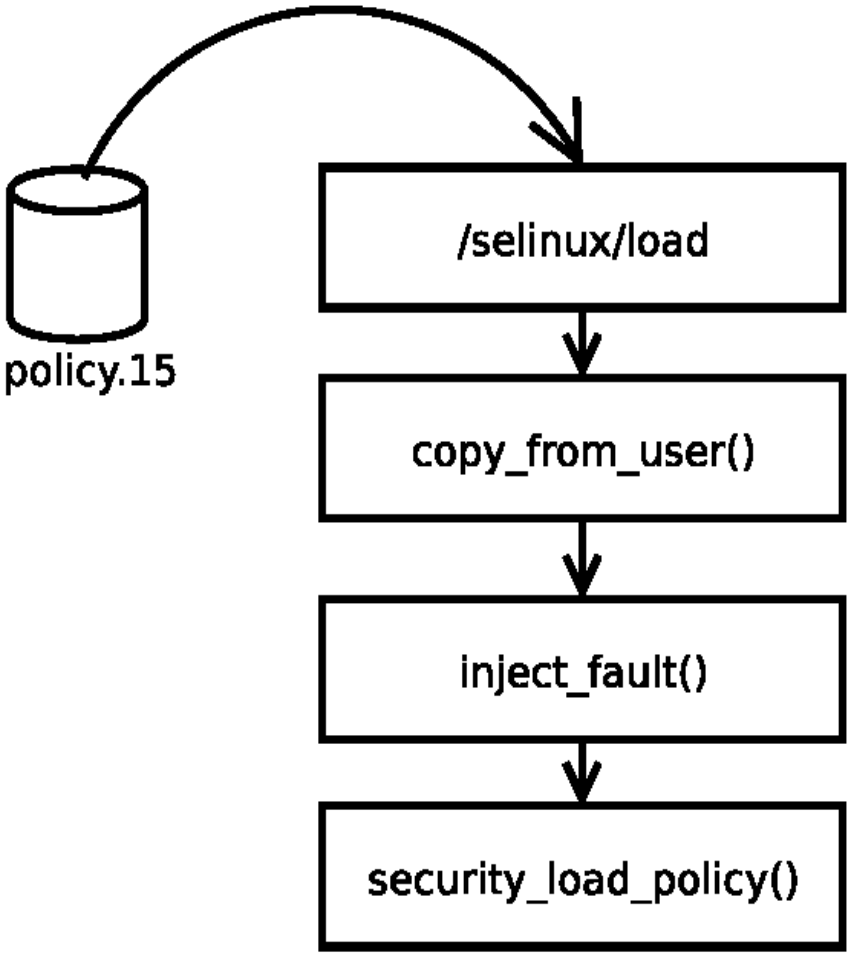
The New Fault Injector Framework

- Allows for distributed processing
- Simplified Injection Method
- Robust recovery from faults and restarts



Error Injector Location

- Implemented by added code to selinuxfs.c
- Injects error as the policy is being loaded
- Errors occur before SELinux processing
  - Most directly represents disk faults
  - If the policy loads in can represent many other faults



The Test Policy

- Designed to reduce policy size and facilitate easier testing
  - 1 domain (kernel_t)
  - 1 user (system) / compared to 3 (system, sysadm, user)
  - 301 rules / compared to 19741
  - Specifically allow permissions needed to run, deny everything else
  - File contexts not modified
- The million dollar question…does this represent a real policy?

The Short Answer

- Yes
  - Our security tests only concern the target file system, and thus a simple policy is representative of a portion of a complex policy
- and No…
  - A real policy would have far more rules with possible interactions
    - Errors may have a greater effect.

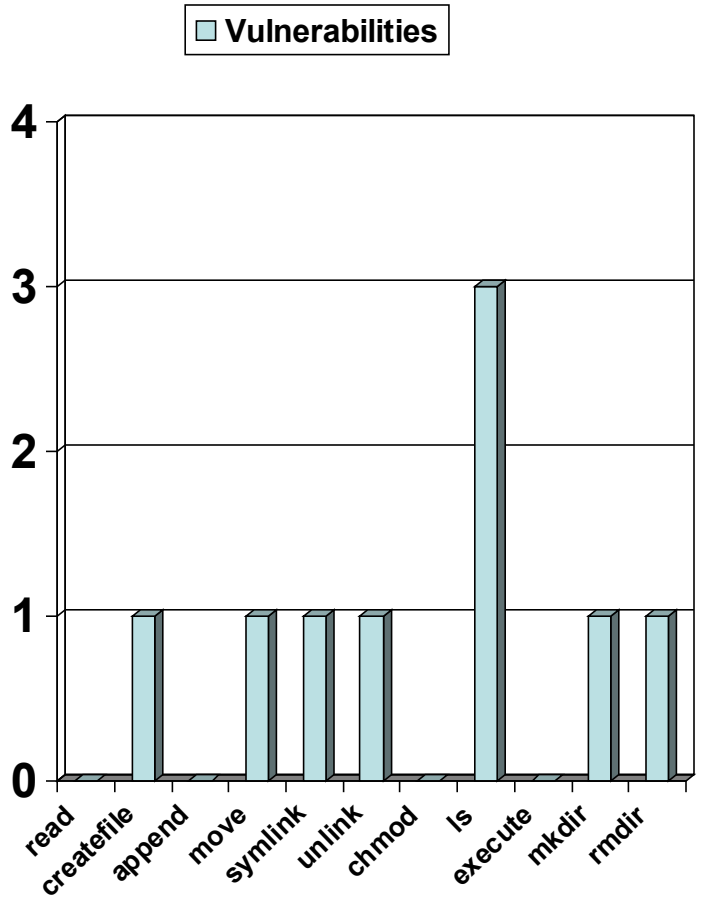
Results

- A majority of the errors had no measured security impact
- Errors that failed policy load may create vulnerabilities if injected after load
  - This is an artifact of our injection method

Errors That Cause Vulnerabilities	8
Errors That Had No Measured Security Impact	96770
Errors That Failed Policy Load	48040
Total Errors Injected	144818

Results (cont.)

- In most cases each vulnerability occurred only once
- Some of the operations actually require multiple permission
  - Our method masked these vulnerabilities
- Read and Execute were accidentally allowed in the fault free policy
  - Injections actually denied read in over 1.3% of the errors



Other Difficulties and Problems

- Every bit was injected with an error, but approx. 3.8% of the runs were lost due to possible UML crashes.
- Memory Leak in UML forced reboot of machine every three hours
- Enabling SELinux auditing would have allowed easier parsing, allowing for a more complete picture.

Future Work

- Correct errors in test policy and re-run experiments
- Trace back vulnerabilities to their cause in source code.
- Perform random error injection in a full policy
- Perform error injection into the Text or Data segments using the ptrace interface
- Independent inspection of source code to compare to our experimental study

References and Questions

- [1] J. Ziegler et al. (2003, May) IBM experiments in soft fails in computer electronics
- [2] T.J. Dell (1997, Nov) A white paper on the benefits of chipkill-correct ECC for pc server main memory