Enhancing Server Availability and Security Through Failure-Oblivious Computing

Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, and William S. Beebee, Jr.

presented by
Dmitriy Levchenkov and Tudor Marian
The problem

- The software that comes to the market is buggy
- Most of the errors are memory access ones
- They can result in
  - program crashes (segmentation fault, etc.)
  - infinite loops
  - security holes (buffer overrun exploitation, etc.)
Why does this problem exist?

- Unsafe languages (like C) are highly popular. They operate with arrays and pointers in sloppy ways with possibilities of out of bounds array accesses and invalid pointer accesses.
- Other languages exist which have built-in validity checks, but they lose in speed and flexibility.
Standard solution

- Use a safe modification of the compiler (e.g. safe-C). This modification has built-in checks for invalid accesses.
- If invalid access occurs the program throws an exception or restarts the program.

Downside:

In some cases it's imperative to continue execution, not stop the program. Restarting the program may be a slow process and may result in loss of data.
Solution: Failure-Oblivious Computing

- If invalid memory access has been identified, instead of throwing an exception try to “ignore the error and continue execution normally”.
- The authors call this strategy a failure-oblivious computation, “since it is oblivious to its failure to correctly access memory”.
Failure-Oblivious Computing: Details

- IF writing a value out of bounds DO nothing!
- IF reading a value out of bounds DO return some manufactured value

if we come up with a carefully manufactured value, the program will be able to continue execution in a normal way.
Goals

- Acceptable Continued Execution
  - eliminate security vulnerability
  - the server should continue to serve its customers

- Acceptable performance
  - Expected slowdown: 8-12 times
  - Not crucial for interactive servers/applications

- Authors claim that the technique can achieve these goals
When we can expect this to work

- Program has short error propagation distances
- Program has short control flow error propagation distances

- should work all right on servers
- bad idea for numerical computations
Advantages

- **Availability**
  
  “we never stop working for you! (sm)”

- **Security**

  no more buffer overruns!

- **Minimal Adoption Cost**

  just compile it!

- **Reduced Administration Overhead**

  To your admin: “U R fired!”
Disadvantages

- Unanticipated execution paths
  - producing bogus results without an error message
  - getting stuck in an infinite loop

- The bystander effect
  "ah, come on, it will fix the error itself..."
**Example of success:**

**Mutt mail client**

- Contains a function to convert UTF-8 string into UTF-7 string.
- Size of the string may increase. The bound on the output string size is calculated incorrectly by Mutt.
- Results in writing beyond the end of the allocated string array.
Example of success: Mutt mail client (page 2)

- **Standard Version:** fails with a segmentation fault
- **Version compiled by safe-C:** terminates and report an error during initialization
- **FO version:** returns a truncated UTF-7 string. The program later reports that the folder with such name cannot be found. Execution of the program continues.
Implementation

- Checking code fairly standard (like in safe-C); uses a table of objects with corresponding bounds
- Continuation code
  - ignore illegal writes
  - manufacture values for illegal reads
  - iterates through small integers (helps if the value affects loop bounds or loop termination conditions)
  - returns 0 and 1 more often than others
Experiments I

Versions
- Standard
- Bounds Check version compiled with CRED safe-C compiler
- Failure-Oblivious version

Behaviour
- Security and Resilience
- Performance
- Stability
Experiments II

- Programs
  - Pine
  - Apache
  - Sendmail
  - Midnight Commander
  - Mutt

- Hardware and OS
  - *Dell* workstation, 2 CPUs *P4* 2.8GHz, 2Gb RAM
  - *Red Hat* 8.0 *Linux*
The memory error
- While processing From field, inserts '\' before special symbols. Doesn't correct the string length appropriately.

Security and Resilience
- Standard version crashes
- Bounds Check version detects an error and terminates during initialization
- Failure Oblivious version truncates the string and that doesn't affect the visible part on the screen. Pine continues to work properly
Pine 4.44 continued

- **Performance**
  - **Slowdown for**
    - Read – 6.9 times
    - Compose – 8.1 times
    - Move – 1.34 times
  - *Not really noticeable by a user*

- **Stability**
  - FO version was used by authors on a regular basis. No unexpected behavior noticed.
Apache HTTP server 2.0.47

- The memory error
  - automatic redirection routine has only space for 10 substrings; if more present in the URL, the routine writes beyond allocated space
- Security and Resilience
  - Standard version corrupts the stack and may be remotely exploited
  - Bounds Check version detects an error and terminates the faulting process. Apache restarts the process automatically (takes time).
  - Failure Oblivious version blocks illegal writes, uses 10 substrings and continues execution normally
Apache continued

- **Performance**
  - **Slowdown**
    - 1.03-1.06 times
  - *Not noticeable by a user*

- **Stability**
  - *was running and serving*
    - www.flexc.csail.mit.edu
  - *for 9 months.*
  - *No complaints received from users*
Sendmail v8.11.6

- The memory error
  - while parsing a mail address a certain combination of 0xFF and `\' characters may trigger writing arbitrarily many characters into the output buffer
- Security and Resilience
  - Standard version corrupts the stack and may be remotely exploited
  - Bounds Check version terminates during the initialization (there are some other memory access errors).
  - Failure Oblivious version blocks illegal writes, the program later rejects the offending letter.
Sendmail continued

- **Performance**
  - **Slowdown**
    - 3.6-3.9 times
  - *Not noticeable at all – mail processing is not time-critical operation*

- **Stability**
  - *was used on a regular basis by the authors*
  - *rejected occasionally sent offending e-mails*
Midnight Commander v4.5.55

- The memory error
  - *while processing links in* `tgz` *files* MC *puts them in a stack-allocated buffer without checking if it has enough space*

- Security and Resilience
  - *Standard version* corruptions the stack and terminates with a segmentation fault
  - *Bounds Check version detects the error and terminates*
  - *Failure Oblivious version blocks illegal writes, the program later rejects incorrect links, reports that to the user and continues to execute.*
Performance
- Slowdown for procedures like Copy/Move/Del 1.4-1.8 times
- Not really noticeable by a user

Stability
- was used on a regular basis by the authors
- rejected opening offending *tgz* files
- turned out that MC has other memory access errors (e.g. when processing configuration files)
**Mutt (concluded)**

- Fails to allocate appropriate buffer for UTF-7 string. FO-code truncates the string. The truncated string is later rejected by the program.
- **Performance**
  - Slowdown for procedures like Read/Move 1.4-3.6 times
  - Not really noticeable by a user
- **Stability**
  - was used on a regular basis by the authors
  - rejected offending strings
Related work

- Using *boundless memory blocks* (if out of bound write occurs, extend the array). Eliminates size-calculation errors.
- Terminate a function in which an error has occurred and return default value.
- FO computing may be applied to safe languages (like Java).
- Compilers community developed only “unsound heuristics” to analyze the code directly for memory errors.
Related work (continued)

- Run-time detection of buffer overrun
- Rebooting
- Repairing data structures
- “Automatic Detection and Repair of Errors in Data Structures” by Brian Demsky and Martin Rinard (to be presented next time)
Conclusions

- Memory errors happen
- Sometimes it's better to continue execution
- Failure-Oblivious Computing might help
- FO computing upgrades a safe compiler
- It tries to ensure program's continuation by
  - discarding invalid writes
  - manufacturing values for invalid reads
- May be successfully applied for servers and other applications with short distances of error propagation.
Thank you all for coming!