Detecting and Eliminating Memory Leaks Using Cyclic Memory Allocation

Huu Hai Nguyen and Martin Rinard
Massachusetts Institute of Technology

presented by: Hannes Payer

Computational Systems Group, University of Salzburg

June 18, 2008

Table of Contents

What is a Memory Leak?

Memory Leak (explicit memory management)

A program that uses explicit memory allocation and deallocation has a memory leak when it fails to free objects that it will no longer access in the future.

Memory Leak (garbage collection)

A program that uses garbage collection has a memory leak when it retains references to objects that it will no longer access in the future.

What we want?

Problem:

Memory leaks are especially problematic for server programs that must execute for long (in principle unbounded) periods of time.

Goal: detect and eliminate memory leaks

Definitions

Allocation Site

An allocation site is a location in the program that allocates memory (e.g.: *malloc* call).

m-Bounded Access Property

An allocation site is *m*-bounded if, at any time during the execution of the program, the program accesses at most the last *m* objects allocated at that site.

Memory Leaks Elimination Procedure

1. Identification
   (find m-bounds empirically)

2. Elimination
   (use specific memory management)

3. Evaluation
   (check correctness)

Instrumentation

The following values are maintained for each allocation site:

- number of objects allocated at that site so far in the computation
- number of objects allocated at that site that have been deallocated so far in the computation
- an observed bound m

Valgrind addrcheck tool is used to obtain the sequence of addresses that the program accesses as it executes

How to find fitting m-bounds?

Procedure

- run instrumented version of a program on a sequence of training inputs of increasing size
- compare the observed bounds m for each allocation site
- if all of these bounds are the same for all of the inputs $\Rightarrow$ the site is m-bounded with bound m

How to find memory leaks?

Procedure

if $number\_allocations - number\_deallocations > m$,
there is a memory leak if the difference either

1) increases during a single run or

2) increases as the size of the input increases

Cyclic Memory Management

- supports programs written in C that explicitly allocate and deallocate objects
- each m-bounded allocation site is given a cyclic buffer with enough space for m objects (preallocated)
- allocation: use the last allocated object slot of cyclic buffer
- deallocation: perform no-op
- application changes are not required

Key issue: distinguish references to objects allocated in cyclic buffers from references to objects allocated via the normal allocation and deallocation mechanism

Variable-Sized Allocation Sites

Some allocation sites allocate objects of different size at different times

- find maximum size of object allocated at each allocation site empirically
- set initial size $= m \times max\_size$
- allocate additional memory at runtime, if buffer runs out of memory

Failure-Oblivious Computing

Program Transformation

sound versus unsound transformation

Unsound Transformation Strategy

enables programs to execute through anomalies to continue to deliver acceptable service

Advantages:

- eliminates security weaknesses
- enables programs to execute successfully through buffer-overflows (e.g.: server attacks)

Experiments

Open-source programs:

- Squid (web proxy cache): 104,573 LOC
- Freeciv (multi-player game): 342,542 LOC
- Pine (email client): 366,358 LOC
- Xinetd (security tool): 23,470 LOC

Evaluation:

- ability to eliminate memory leaks
- potential impact of an incorrect estimation of the bounds m at different allocation sites

Experiments

Procedure:

- training runs: find suitable m-bounded allocation sites
- validation runs: test accuracy of the estimated bounds from the training runs and effect of overlaying live objects
- conflict runs: for each m-bounded allocation site ($m > 1$) test $\lceil m/2 \rceil$ bound (overlaying)
- long-term usage: several months lasting experiments

Results

Program	% m-bounded	% memory	%invalidated
Squid	55.7	86.0	2.9
Freeciv	48.3	84.9	0.0
Pine	48.3	15.0	1.5
Xinetd	58.8	89.8	0.0

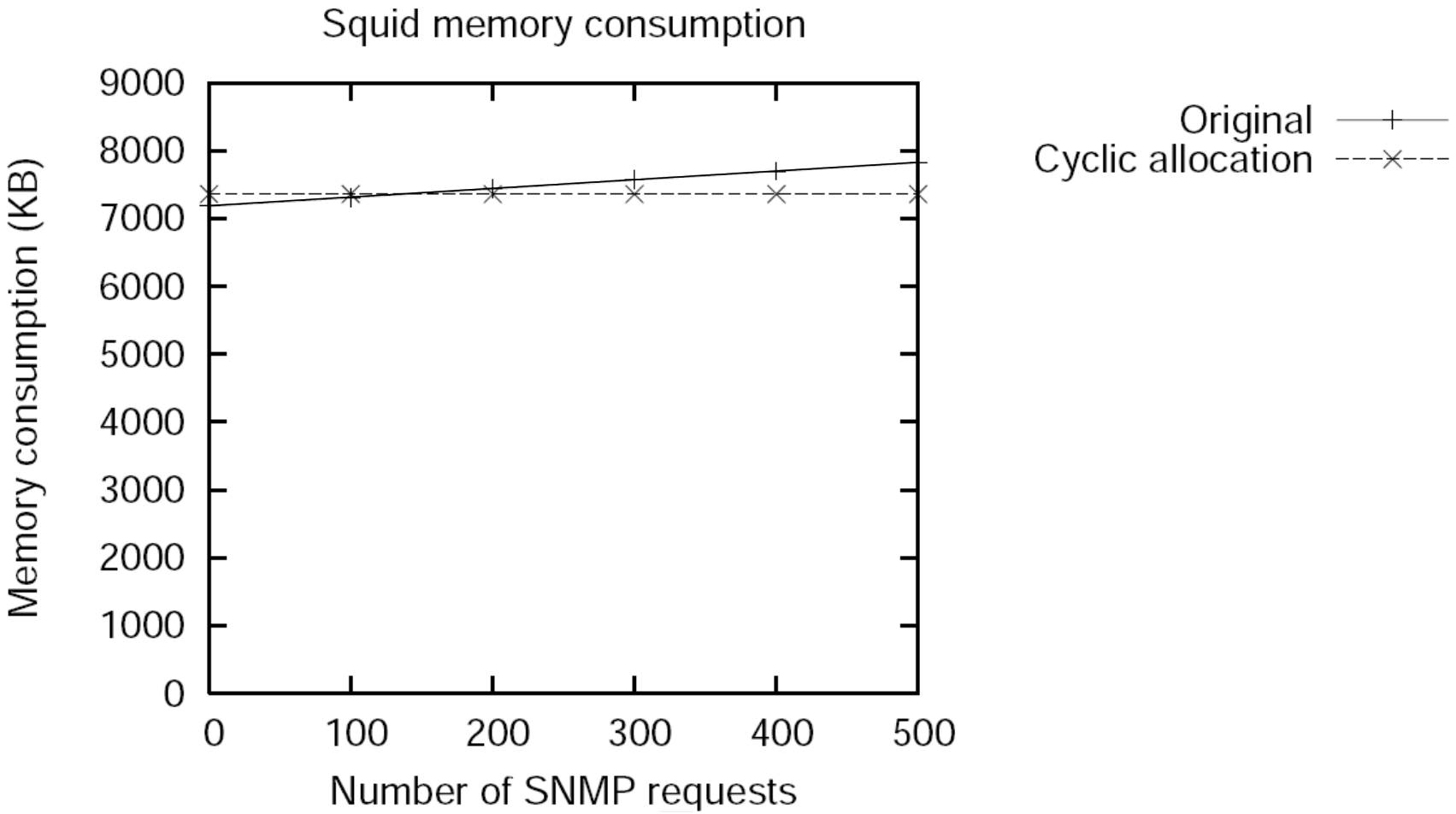
Squid

- training inputs: different HTTP, FTP, SNMP queries with different attributes
- training and validation runs:

m	1	2	3	14
# sites	30	2	1	1

- overlaying live objects: problem with tree data structure
- memory leaks: memory leak found in the SNMP module
- conflict runs: able to process other requests
- long-term usage: no errors

Squid


Squid memory consumption
Memory consumption (KB)
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
100
200
300
400
500
Number of SNMP requests
Original
Cyclic allocation

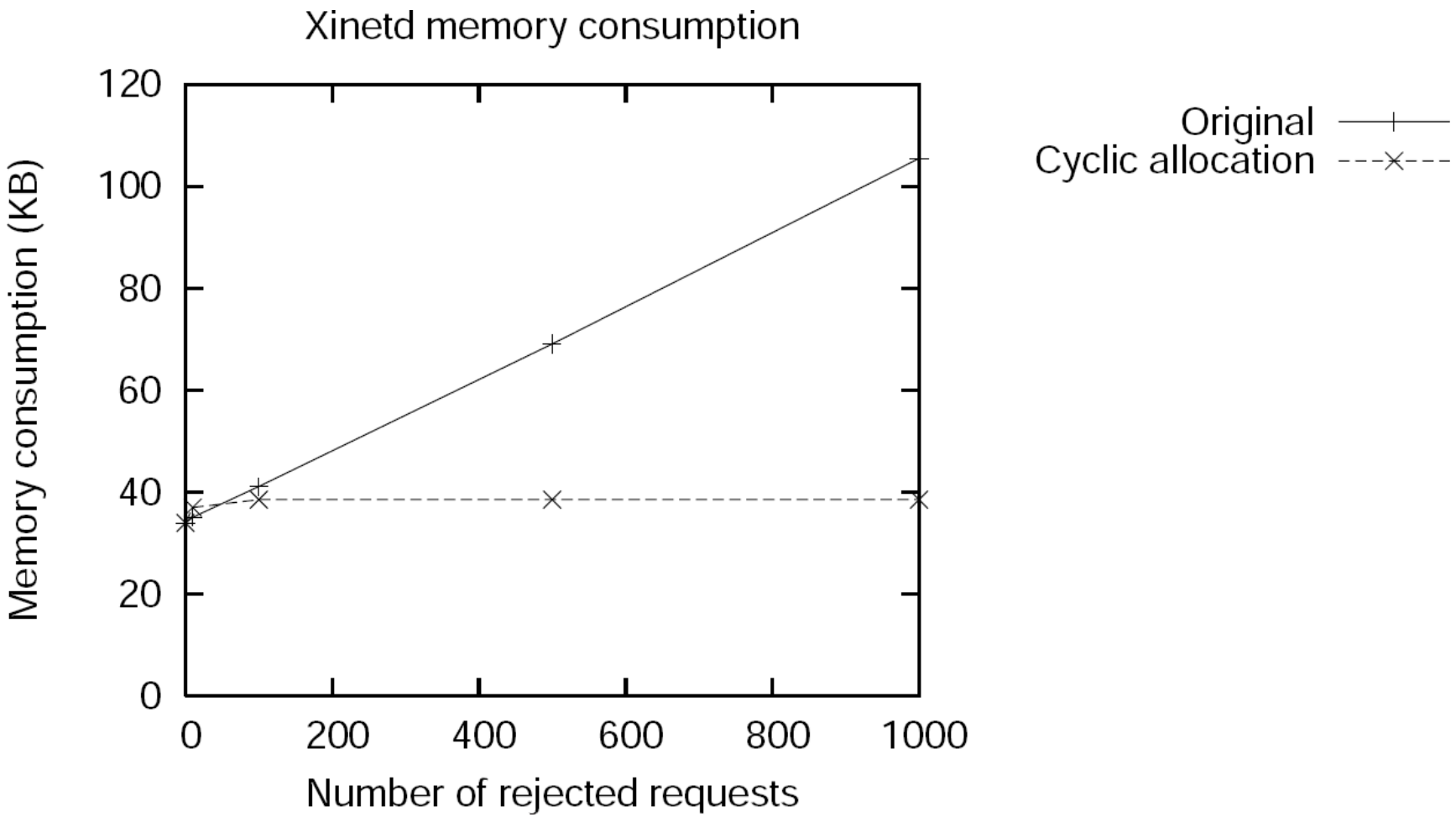
Xinetd

- training inputs: 10-200 requests
- training and validation runs:

m	1
# sites	10

- memory leaks: in the connection-handling code
- no overlaying and conflict experiments

Xinetd


Xinetd memory consumption
Memory consumption (KB)
120
100
80
60
40
20
0
0
200
400
600
800
1000
Number of rejected requests
Original
Cyclic allocation

Conclusion

Contribution:

- detection and
- avoidance of memory leaks without changing the program source code

Advantages of cyclic memory management:

- eliminates any memory leaks at allocation sites
- simple to implement and does not require the development of heavyweight static analysis

Future work:

- support other resource leaks (e.g.: file descriptors)
- use static analysis to find correct m-bounds