KULeuven
DistriNet
Research Group

Dnmaloc: a more secure memory allocator

Yves Younan, Wouter Joosen, Frank Piessens and Hans Van den Eynden

DistriNet, Department of Computer Science

Katholieke Universiteit Leuven

Belgium

Yves.Younan@cs.kuleuven.ac.be

28 September 2005

Overview

- Introduction
- Attacks
- Doug Lea's malloc (ptmalloc)
- A Safer Allocator
- Related Work
- Conclusion

Introduction

- Many allocators ignore security
- Performance and waste limitation is more important
- Many allocators can be abused to perform code injection attacks
- More security is possible at a modest cost

Overview

- Introduction
- Attacks
  - Heap-based buffer overflow
  - Off by One/Off by Five
  - Dangling Pointer References
- Doug Lea's malloc (ptmalloc)
- A Safer Allocator
- Related Work
- Conclusion

Heap-based buffer overflows

- Heap memory is dynamically allocated at run-time
- Can also be overflowed but no return address is available
- Modify data pointers (IPO) or function pointers - not always available
- Modify the memory management information associated with heap memory

Off by one / Off by five

- Special case of buffer overflow: limited space needed
- Off by one: write one byte past the bounds
- Off by five: don't occur often but demonstrate low-space attacks
- Usually only exploitable on little endian machines

(LSB is stored before MSB)

Dangling pointer references

- Pointers to memory that is no longer allocated
- Dereferencing is unchecked in C
- Generally leads to crashes (SIGSEGV)
- Can be used for code injection attacks when deallocated twice (double free)
- A double free can be used to change memory management information allowing an overwrite of arbitrary memory locations

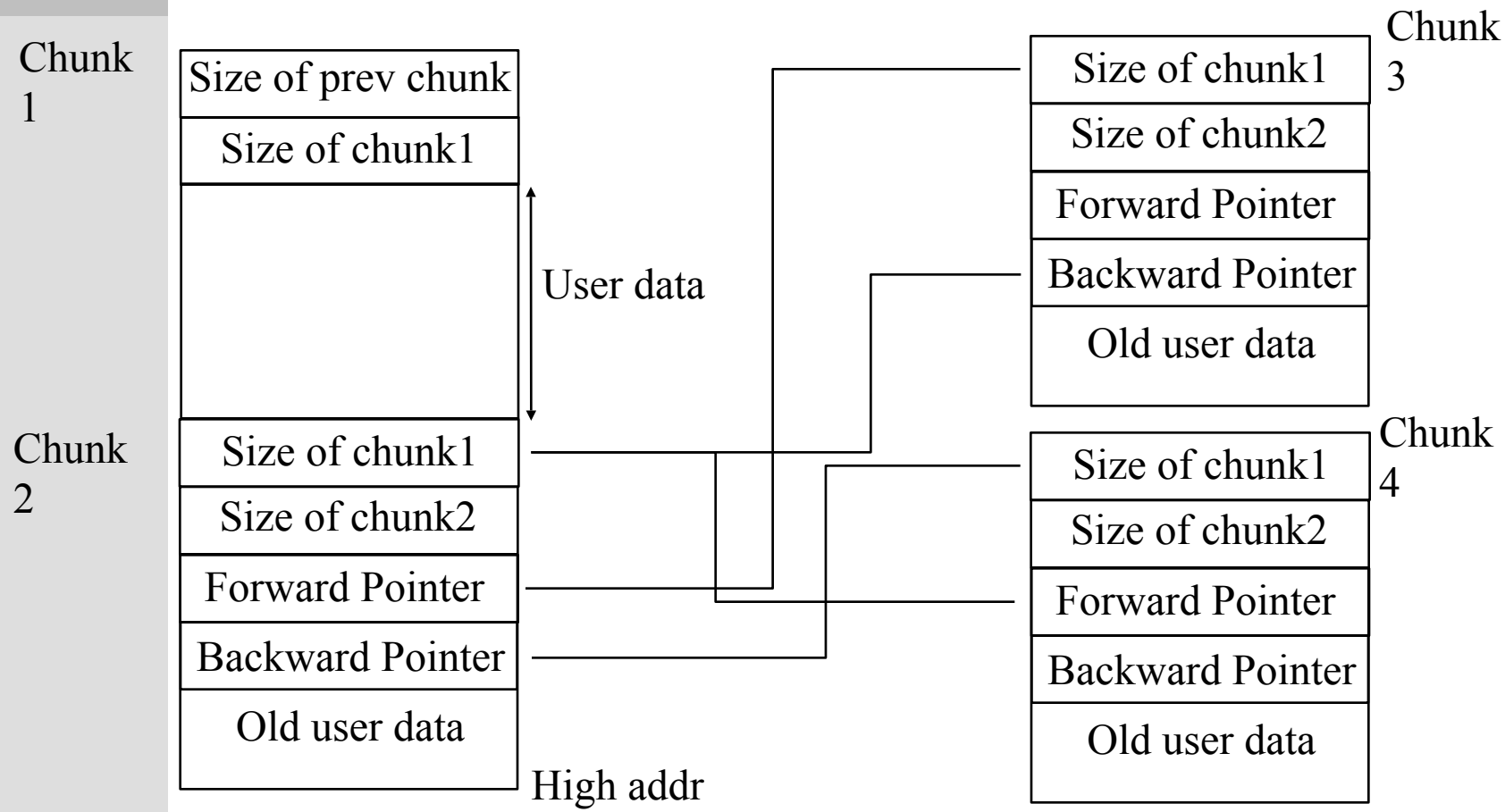
Overview

- Introduction
- Attacks
- Doug Lea's malloc (Linux)
- A Safer Allocator
- Related Work
- Conclusion

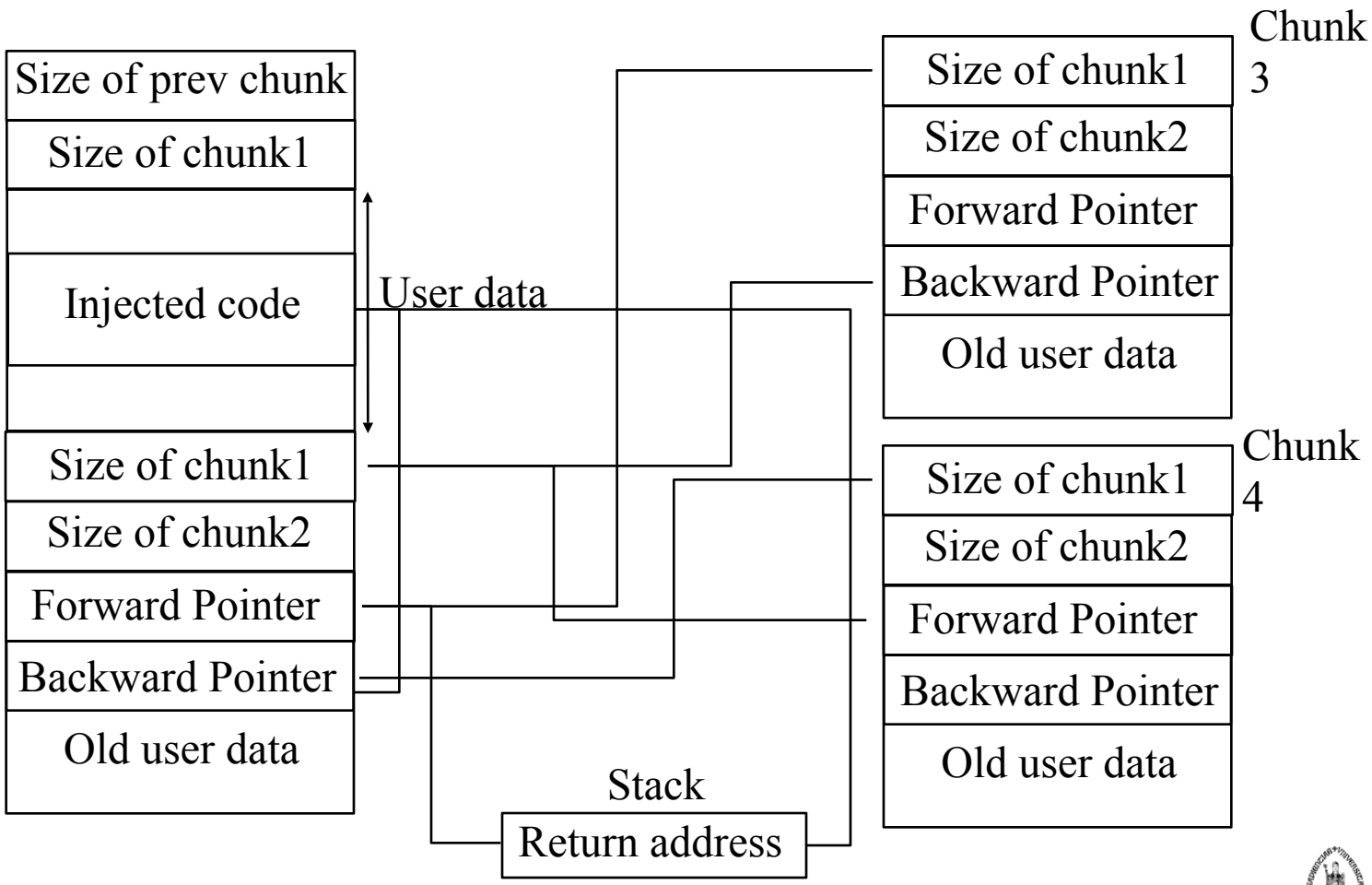
Doug Lea's malloc

- GNU lib C malloc (ptmalloc) is based on this malloc
- Every allocation is represented by a chunk
- Management information stored before the chunk
- Free chunks stored in doubly linked list of free chunks
- Two bordering free chunks are coalesced into a larger free chunk
- Description based on dlmalloc 2.7.2

Doug Lea's malloc



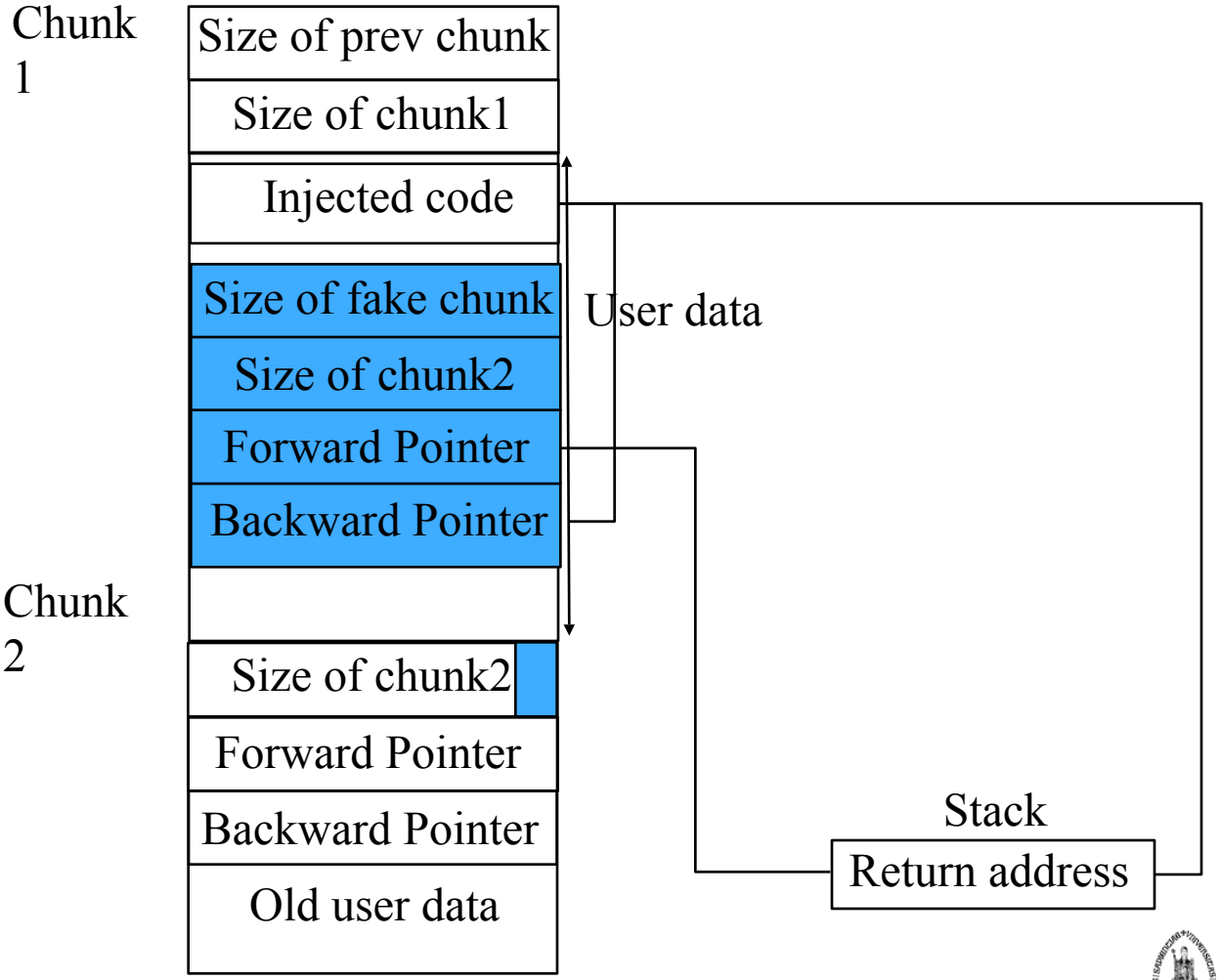
Heap Overflow (dlmalloc)



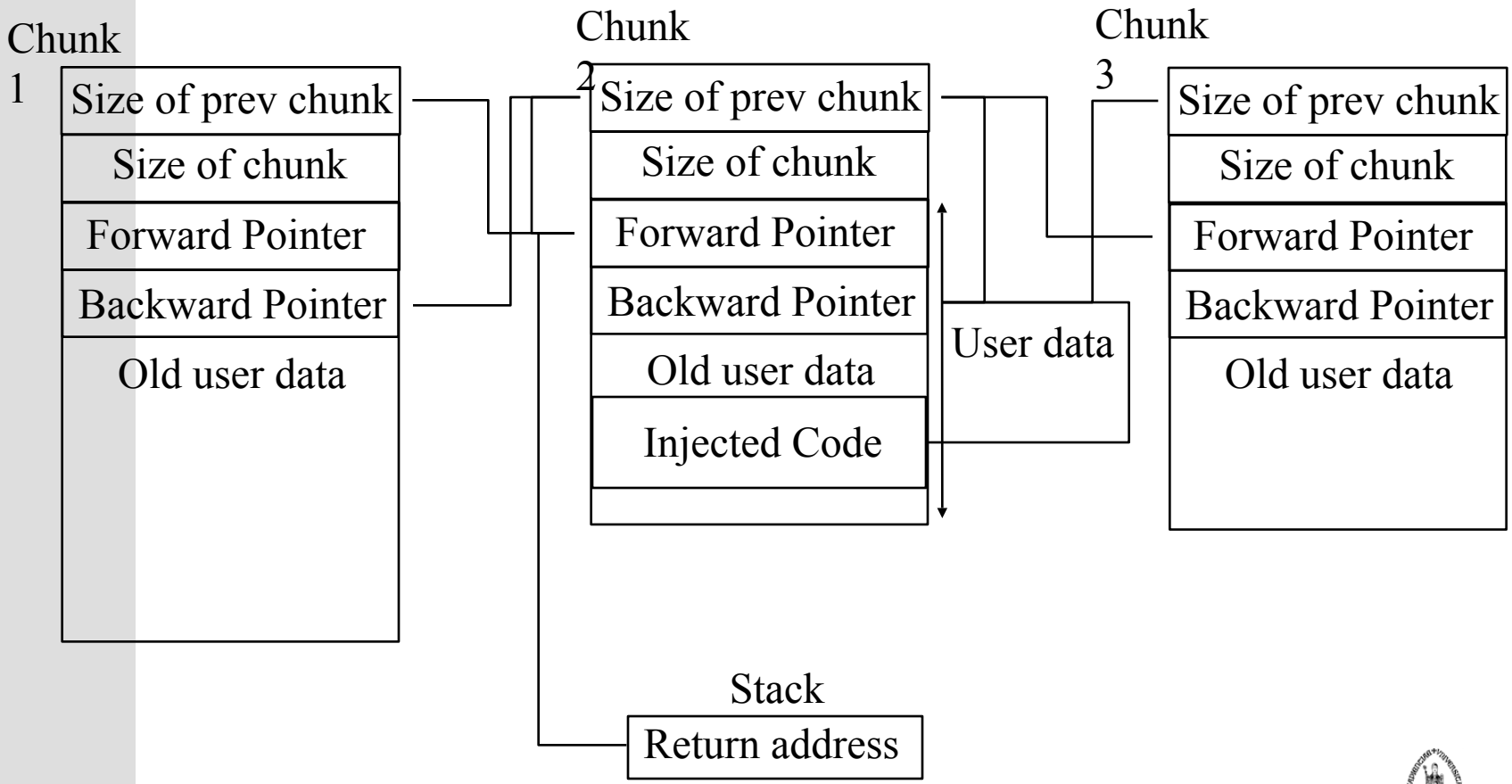
Off by one (dlmalloc)

- Chunk sizes are multiples of 8
- Size contains two flags mmapped and prev_inuse
- Two chunks must be next to each other (no padding) for off by one
- Prev_size of next will be used for data
- Overwrite 1 byte of the size and set prev_inuse to 0 and set a smaller size
- Make a fake chunk, containing modified pointers

Off by one (dlmalloc)



Dangling pointer references (dlmalloc)



Doug Lea's malloc conclusion

- Vulnerable to:
  - Heap overflow
  - Off by one/five
  - Double free
- Version 2.8.x contains some mitigation techniques, see related work

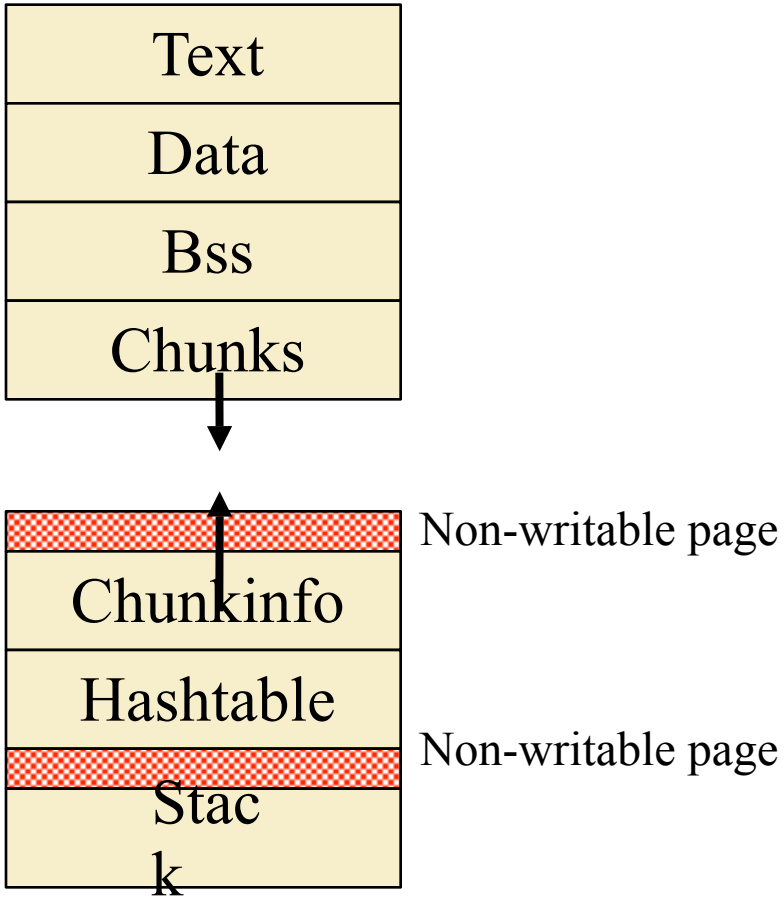
Overview

- Introduction
- Attacks
- Memory Allocators
- A Safer Allocator
- Related Work
- Conclusion

Design

- On most modern systems code and data are loaded into separate memory locations
- We apply the same to chunk information and chunks
- Chunk info is stored in separate contiguous memory
- This area is protected by guard pages
- A hashtable is used to associate chunks with chunkinfo
- The hashtable contains pointers to a linked list of chunk information accessed through the hashfunction
- Implemented in a prototype called dnmalloc (DistriNet malloc), based on dlmalloc

Modified memory layout

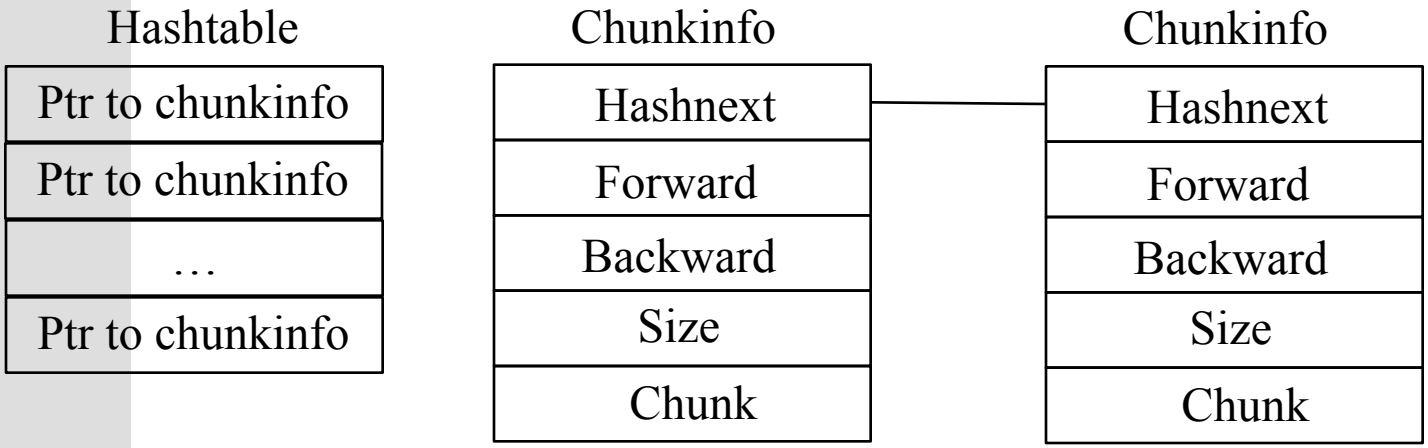


Dnmalloc: hashtable

- Hashtable is stored before the stack in a mmaped area big enough to hold it
- Each page is divided in 256 possible chunks (of 16 bytes, minimum chunk size)
- These chunks are separated into 32 groups of 8 chunks
- Each group has an entry in the hashtable, maximum 32 entries for every page
- One element of the group is accessed through a linked list of max. 8 elements

Dnmalloc: hashfunction

- To find a chunk's information from a chunk we do the following:
  - Substract the start of the heap from the chunk's address
  - Shift the result 7 bits to the right: gives us the entry in the hashtable
  - Go over the linked list till we have the correct chunk



Dnmalloc: Managing chunk information

- A fixed area is mapped below the hashtable for chunkinfos
- Free chunkinfos are stored in a linked list
- When a new chunkinfo is needed the first element in the free list is used
- If none are free a chunk is allocated from the map
- If the map is empty we map extra memory for it (and move the guard page)
- Chunk information is protected by guard pages

Dnmalloc performance overhead

Spec CPU2000 results for dlmalloc and dnmalloc
(13 runs on 8 identical pcs (P4 2.8ghz, 512mb) = 104 runs)

Program	Dlmalloc runtime	Dnmalloc runtime	Overhead percentage
gzip	253 +- 0	255.98 +- 0.01	1.18%
vpr	360.93 +- 0.16	360.55 +- 0.13	-0.11%
gcc	153.93 +- 0.05	154.76 +- 0.04	0.54%
mcf	287.19 +- 0.07	290.09 +- 0.07	1.01%
crafty	253 +- 0	254 +- 0	0.40%
parser	346.95 +- 0.02	346.61 +- 0.05	-0.10%
eon	771.05 +- 0.13	766.55 +- 0.11	-0.58%
perlbmk	243.20 +- 0.04	253.51 +- 0.05	4.24%
gap	184.07 +- 0.02	184 +- 0	-0.04%
vortex	250 +- 0	258.79 +- 0.04	3.52%
bzip2	361.64 +- 0.05	363.26 +- 0.07	0.45%
twolf	522.48 +- 0.43	513.27 +- 0.41	-1.76%

Dnmalloc memory usage overhead

- 2 guard pages -> 8192kb
- 20 bytes of chunk information / chunk
- 1 page for the hashtable per 32 pages of allocated memory
- Overhead depends on the amount of chunks and the size of the chunks
- Overhead for dlmalloc: 8 bytes per chunk

Overview

- Introduction
- Attacks
- Memory Allocators
- A Safer Allocator
- Related Work
  - Robertson et al. heap protector
  - Contrapolice
  - Glibc 2.3.5
- Conclusion

Robertson's heap protector

- Checksum stored in every chunk's header
- Checksum encrypted with a global read-only random value
- Checksum added when allocated, checked when freed
- Could be bypassed if memory leaks exist
- Dlmalloc 2.8.x implements a slightly modified version of this

Overview

- Introduction
- Attacks
- Memory Allocators
- A Safer Allocator
- Related Work
  - Robertson et al. heap protector
  - Contrapolice
  - Glibc 2.3.5
- Conclusion

Contrapolice

- Protects chunks by placing canaries (random) before and after the chunk
- Before exiting from a copy function, it checks if the canary before matches the canary after
- Could be bypassed if the canary value is leaked

Overview

- Introduction
- Attacks
- Memory Allocators
- A Safer Allocator
- Related Work
  - Robertson et al. heap protector
  - Contrapolice
  - Glibc 2.3.5
- Conclusion

Glibc 2.3.5

- Sanity checks before doing operations on chunks:
- Will not unlink a chunk if !(p->fd->bk == p->bk->fd == p)
- Checks if chunks are on the heap
- Checks if chunks are larger or equal to min size (16 bytes) and smaller than memory allocated up to now
- Checks if the first element on the list is the one being added or if it's not in use (prevents double free)
- Prevents current attack techniques
- Hopefully Daniel and I can solve that

Overview

- Introduction
- Attacks
- Memory Allocators
- A Safer Allocator
- Related Work
- Future work
- Conclusion

Future work

- Important limitation pointers stored on the heap are not protected (also holds for other countermeasures)
- Possible solution: store pointers in a pointer-only area
- Chunkinfo has 2 extra fields: pointerstart and pointersize
- Requires compiler modifications to ensure access of correct memory
- More analysis is needed: might break stuff

Overview

- Introduction
- Attacks
- Memory Allocators
- A Safer Allocator
- Related Work
- Future work
- Conclusion

Conclusion

- Many allocators ignore security issues
- Safer allocators are not necessarily much slower
- This work is part of larger research where other important areas in memory are also separated from normal data (currently working on a stack implementation with a visiting researcher Davide Pozza).
- Which is part of my real research: a more methodical approach to designing countermeasures
- Paper associated with this talk: Yves Younan, Wouter Joosen and Frank Piessens and Hans Van den Eynden. Security of Memory Allocators for C and C++. Technical Report CW419, Departement Computerwetenschappen, Katholieke Universiteit Leuven, July 2005
- See http://fort-knox.org (also has other papers: master thesis on vulnerabilities/some countermeasures, overview of all existing countermeasures, paper on the methodical approach)