Boundless Memory Blocks

Cristian Cadar
Massachusetts Institute of Technology
(now Stanford University)

M. Rinard, D. Dumitran
D. Roy, T. Leu
Massachusetts Institute of Technology

Annual Computer Security Applications Conference (ACSAC 2004)

Motivation

- Buffer overflows – the most common cause of security vulnerabilities
  - Majority of CERT reports are related to buffer overflows
  - Costs estimated in the billions of dollars

Memory Errors

- Buffer overflow attacks due to memory errors:
  - Usually on the call stack
  - But also on the heap

Safe C compilers

- Instrument the program with dynamic checks to detect illegal memory accesses
- When a buffer overflow is detected, program terminates with an error message

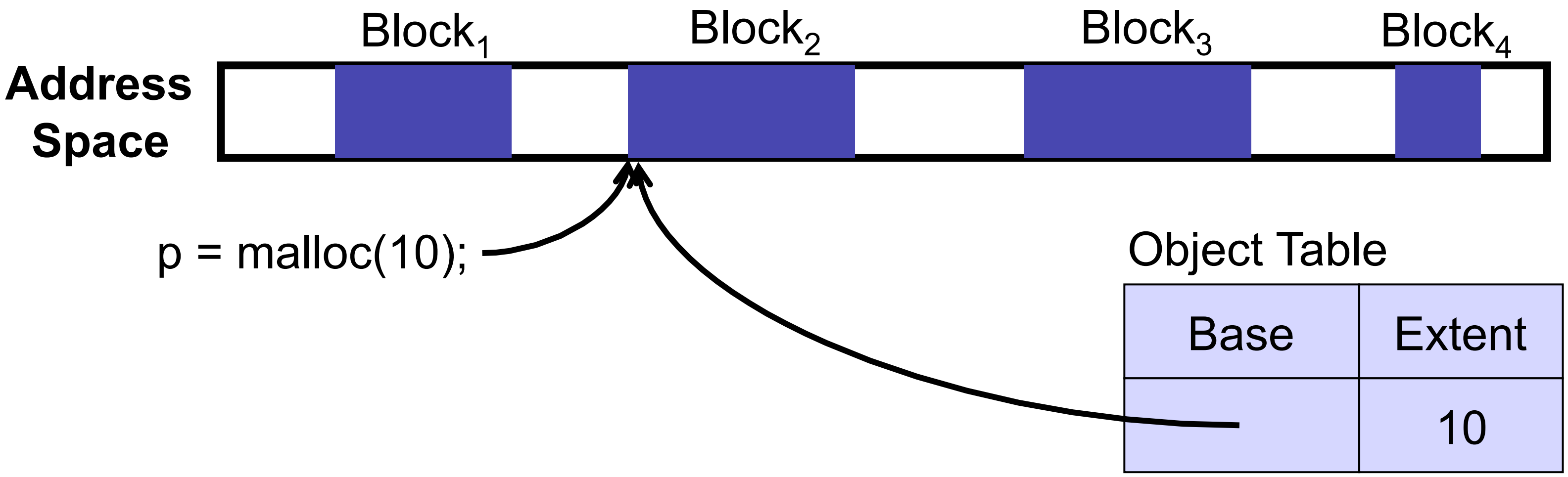
Continued Execution

- Detection critical, sometimes not the whole story
  - Terminating the program can be disruptive
  - Doesn’t address denial of service attacks
- **Focus on continued execution**
- **Through memory errors**

Our Technique

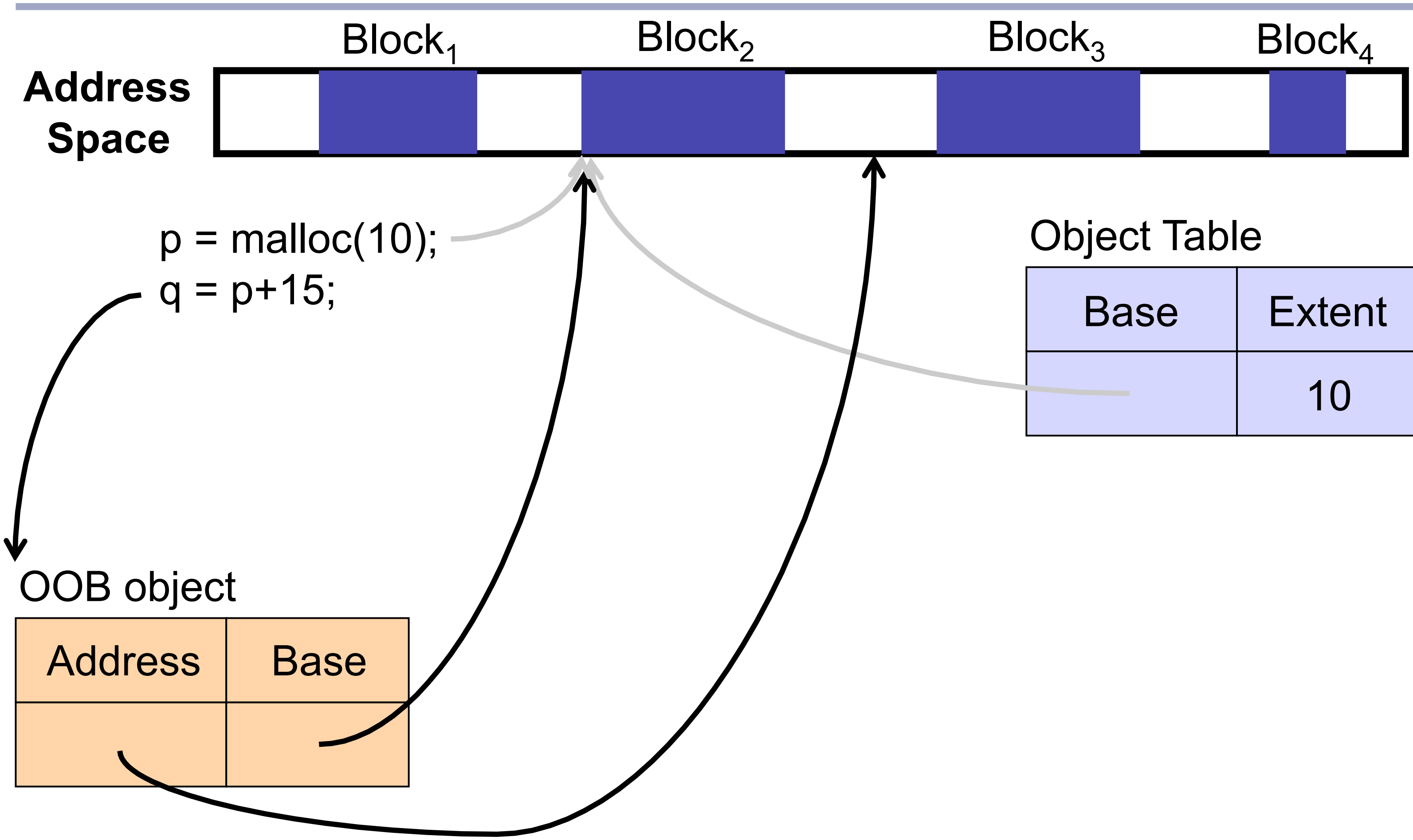
- Detect out of bounds writes
- Store values in a hash table
- Return values for corresponding reads

CRED



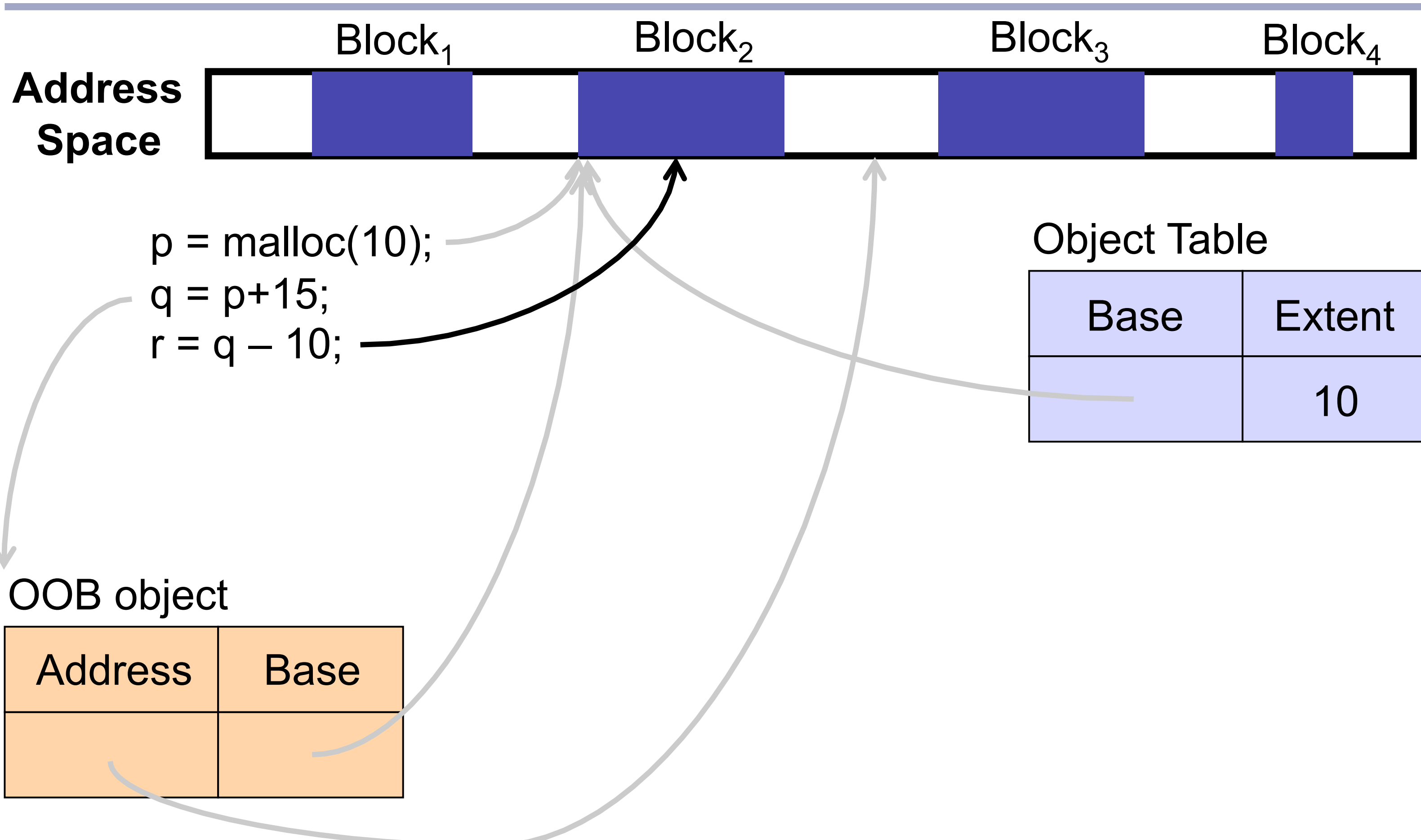
Block1
Block2
Block3
Block4
Address Space
p = malloc(10);
Object Table
Base
Extent
10

CRED

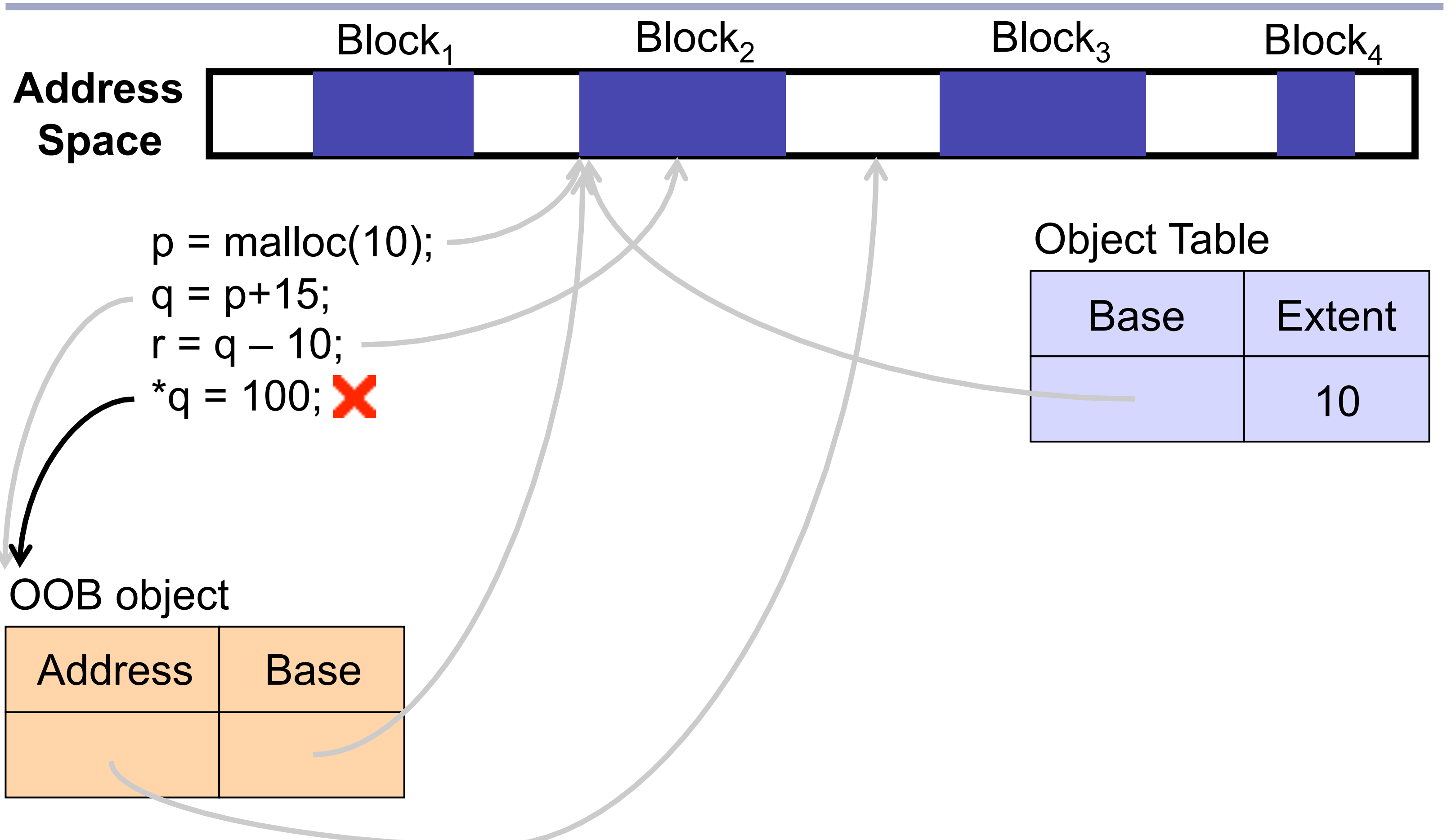


Block1
Block2
Block3
Block4
Address Space
p = malloc(10);
q = p+15;
Object Table
Base
Extent
10
OOB object
Address
Base

CRED

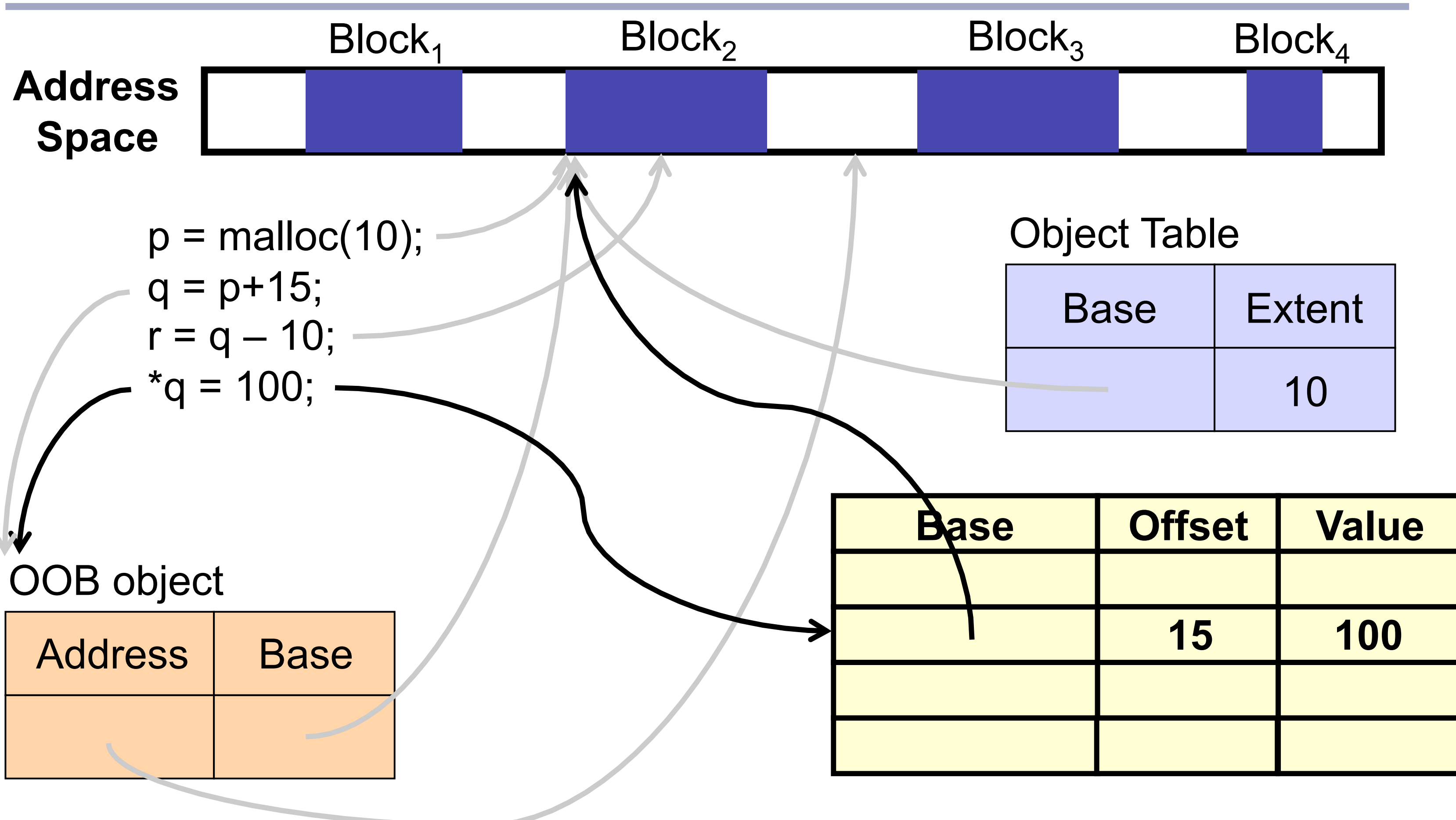

Block$_1$
Block$_2$
Block$_3$
Block$_4$
Address Space
p = malloc(10);
q = p+15;
r = q – 10;
Object Table
Base
Extent
10
OOB object
Address
Base

CRED

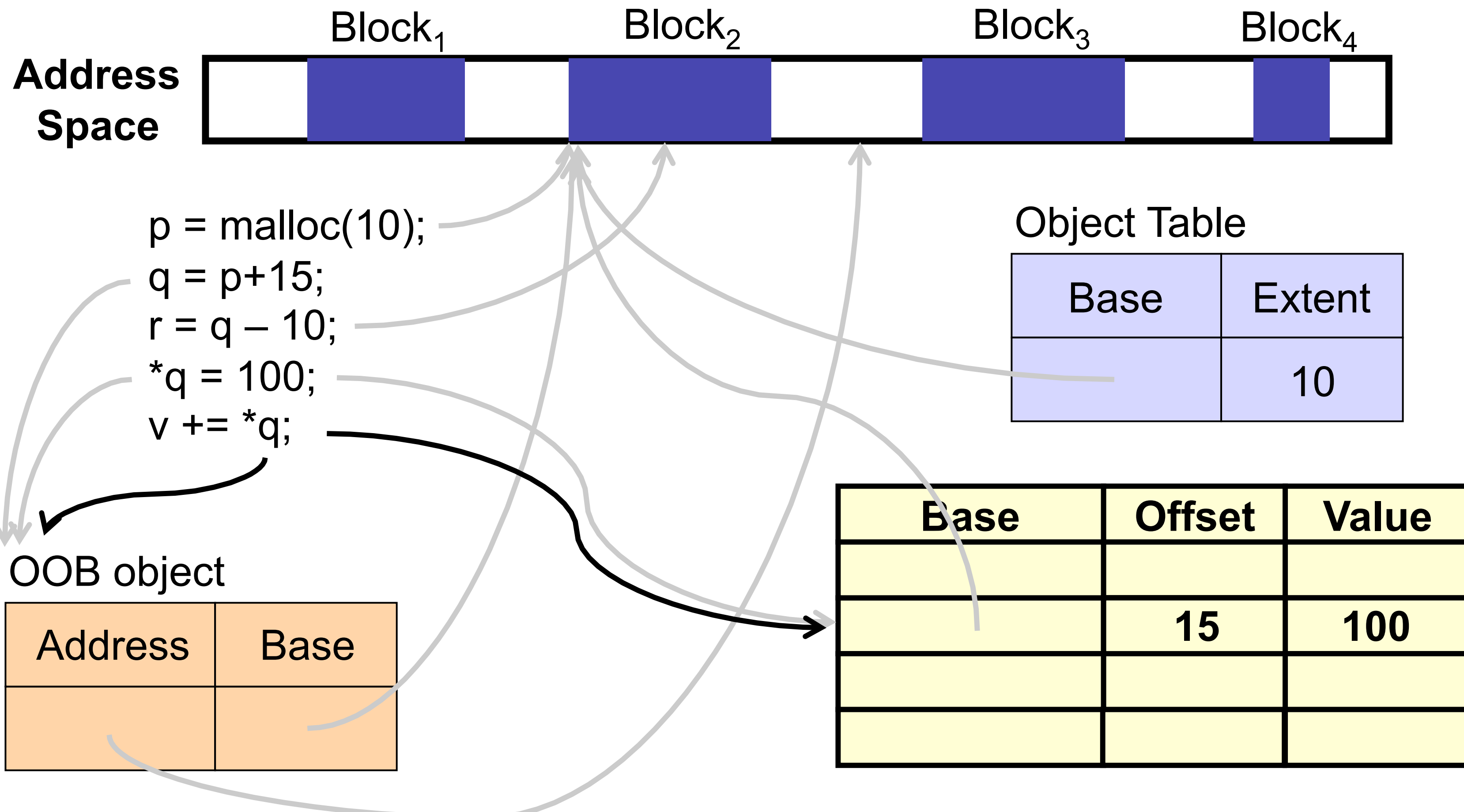


Block1
Block2
Block3
Block4
Address Space
p = malloc(10);
q = p+15;
r = q – 10;
*q = 100;
Object Table
Base
Extent
10
OOB object
Address
Base

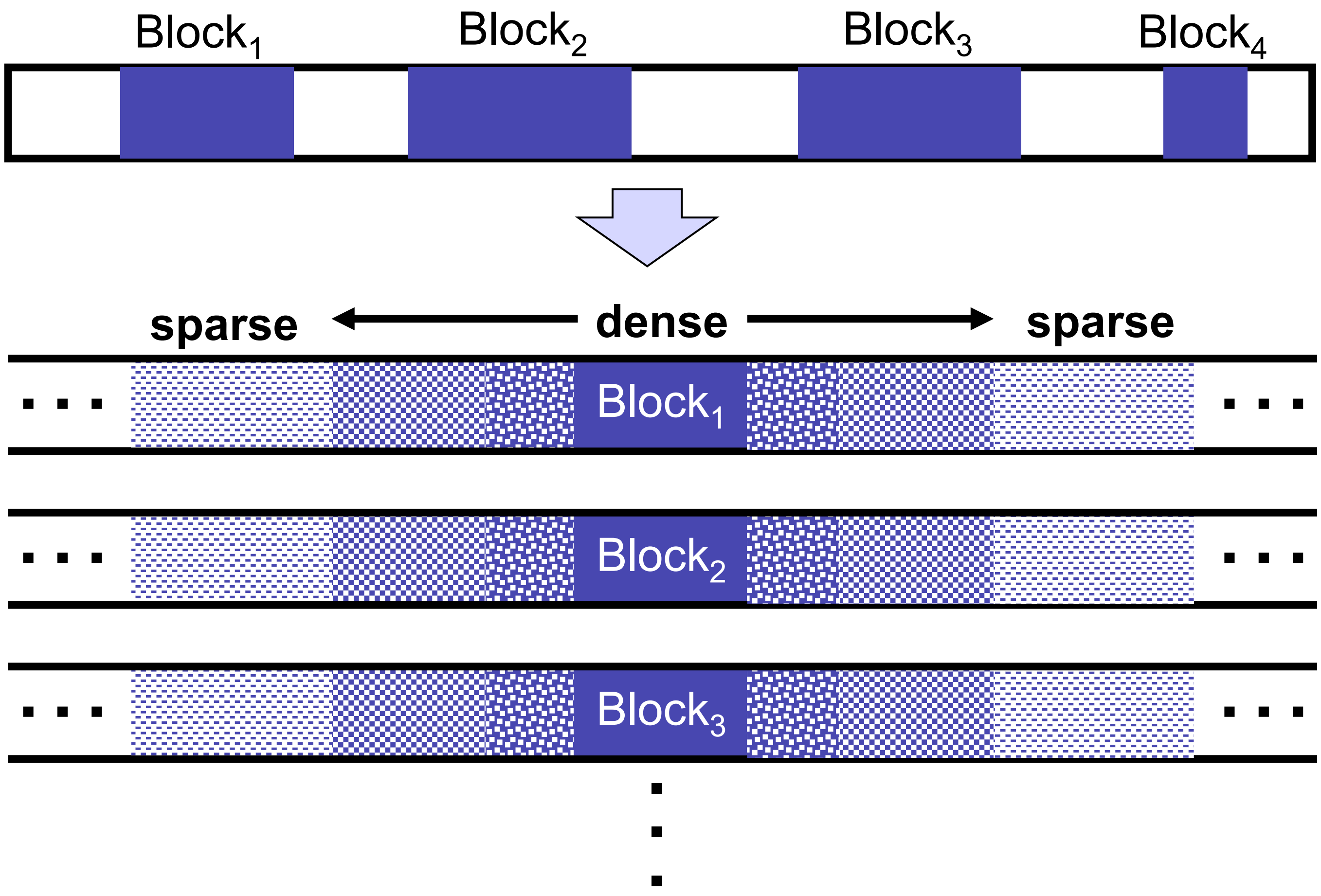
BMB Compiler



BMB Compiler



Net Effect of Our Technique



Possible Problems

- New DOS attack
  - Craft an input which will cause a large number of writes
  - Solution: treat the hash table as a fixed-size cache using the LRU replacement policy

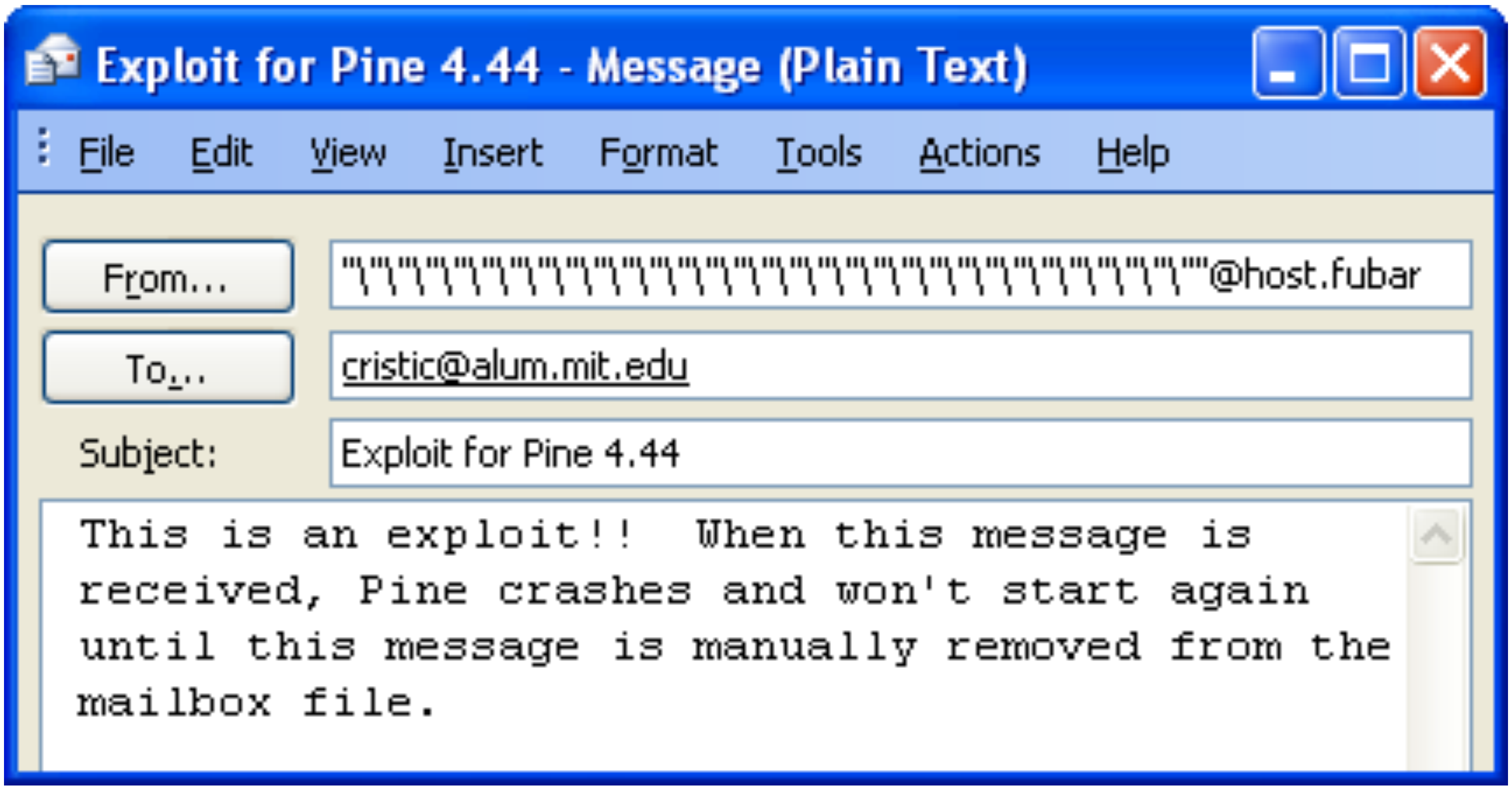
Possible Problems (cont.)

- Cache Misses
  - Bounded number of OOB writes?
  - Haven’t triggered cache misses in our benchmarks
  - But may be a serious problem
- Uninitialized reads
  - Found in Midnight Commander
  - Automatic zero-initialization

Evaluation

- Tested several open source programs
  - Servers: Apache, Sendmail
  - Mailers: Pine, Mutt
  - Utilities: Midnight Commander
- On publicized buffer overflow security vulnerabilities
  - SecuriTeam, Security Focus

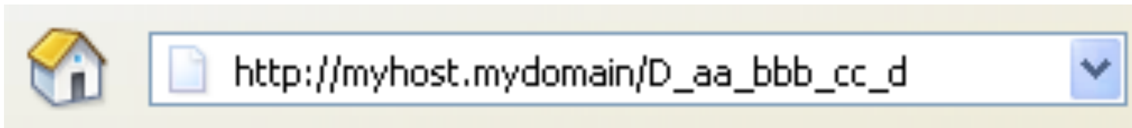
Vulnerabilities – Pine 4.44



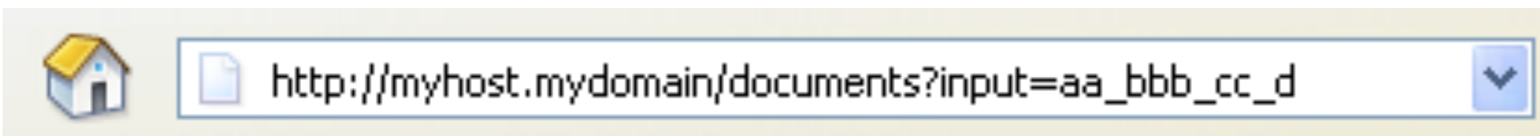
Vulnerabilities – Apache 2.0.47

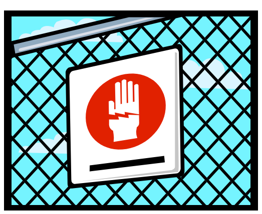
- Apache can redirect some URLs, which are specified by regular expressions
- Example: redirect URLs of the form *http://myhost.mydomain/D_(a*)_(b*)_(c*)_(d*)* to URLs of the form *http://myhost.mydomain/documents?input=$1_$2_$3_$4*

Vulnerabilities – Apache 2.0.47



D_(a*)_(b*)_(c*)_(d*) →
documents?input=$1_$2_$3_$4





Static buffer contains space for only 10 parenthesized captures!

Evaluation (cont.)

- Three versions per benchmark
  - GCC (Standard Compilation)
  - CRED (Bounds Check Compilation)
  - BMB (Boundless Memory Blocks Compilation)
- Tested each versions on the acquired vulnerabilities

Results

	Secure		
	GCC	CRED	BMB
Pine	✗	✓	✓
Mutt	✗	✓	✓
Apache	✗	✓	✓
Sendmail	✗	✓	✓
MC	✗	✓	✓

Results

	Secure			Continues Correctly		
	GCC	CRED	BMB	GCC	CRED	BMB
Pine	✗	✓	✓	✗	✗	✓
Mutt	✗	✓	✓	✗	✗	✓
Apache	✗	✓	✓	✓ (yellow)	✓ (yellow)	✓
Sendmail	✗	✓	✓	✗	✗	✓
MC	✗	✓	✓	✗	✗	✓

Results

	Secure			Continues Correctly			Initializes		
	GCC	CRED	BMB	GCC	CRED	BMB	GCC	CRED	BMB
Pine	✗	✓	✓	✗	✗	✓	✓ (yellow)	✓ (yellow)	✓
Mutt	✗	✓	✓	✗	✗	✓	✓ (yellow)	✓ (yellow)	✓
Apache	✗	✓	✓	✓ (yellow)	✓ (yellow)	✓	✓	✓	✓
Sendmail	✗	✓	✓	✗	✗	✓	✓	✗	✓
MC	✗	✓	✓	✗	✗	✓	✓	✓ (yellow)	✓

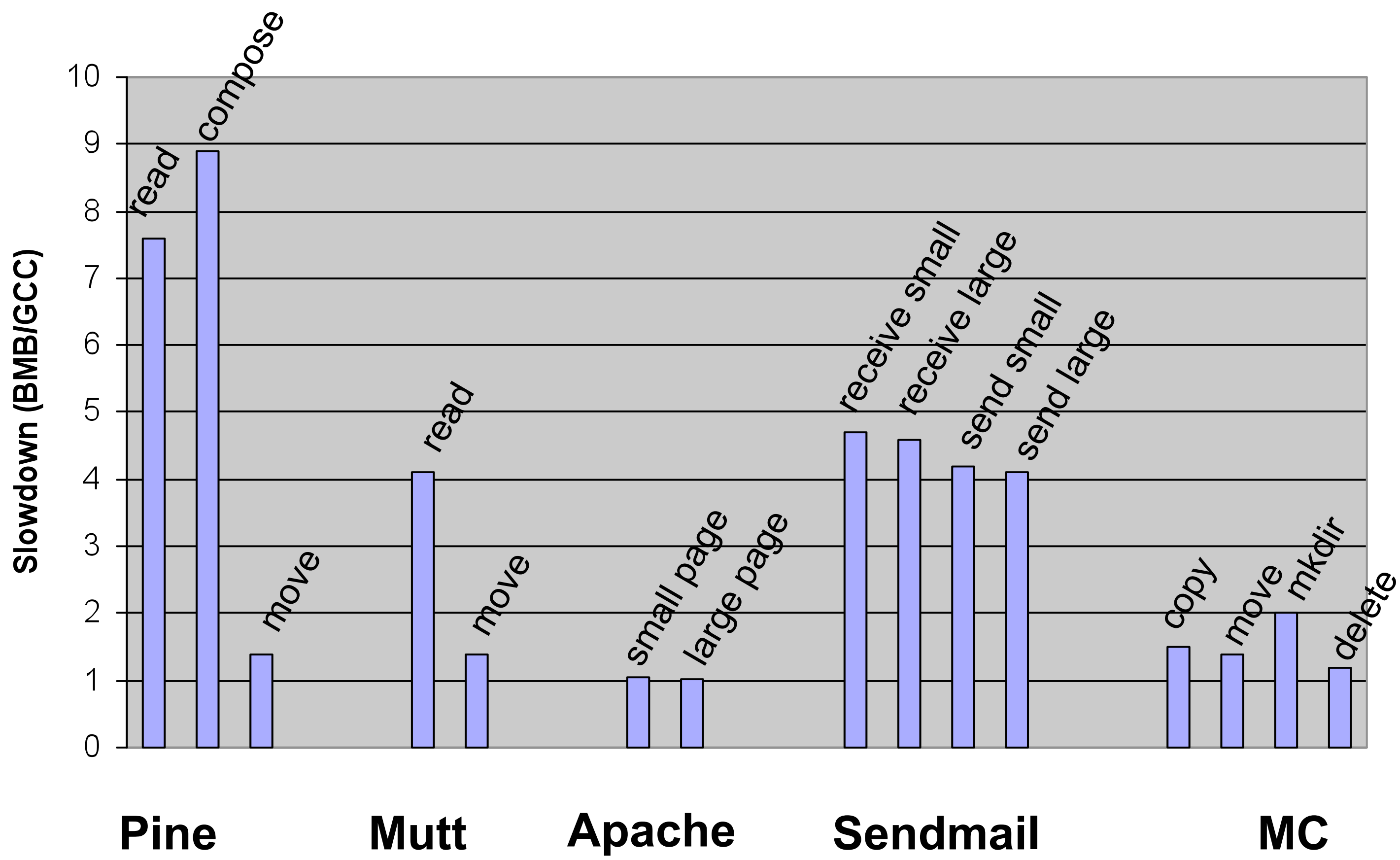
Results

	Secure			Continues Correctly			Initializes			Correct For Attack		
	GCC	CRED	BMB	GCC	CRED	BMB	GCC	CRED	BMB	GCC	CRED	BMB
Pine	✗	✓	✓	✗	✗	✓	✓ (yellow)	✓ (yellow)	✓	✗	✗	✓
Mutt	✗	✓	✓	✗	✗	✓	✓ (yellow)	✓ (yellow)	✓	✗	✗	✓
Apache	✗	✓	✓	✓ (yellow)	✓ (yellow)	✓	✓	✓	✓	✗	✗	✓
Sendmail	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗	✗	✓ (yellow)
MC	✗	✓	✓	✗	✗	✓	✓	✓ (yellow)	✓	✗	✗	✓ (yellow)

Decoupled Errors

- Developers may incorrectly calculate the size of a buffer
  - Hard to reason about the worst case, which is usually exploited by security attacks
- But the rest of code is correct
  - Although the programmer failed to allocate enough space, the program usually correct when provided with (conceptually) unbounded memory blocks.

Performance



Related Work – Continued Execution

- Failure Oblivious Computing [Rinard et al, OSDI 2004]
- Execution Transactions [Sidiroglou et al, Columbia Univ. TR 2004]
- BMB compiler generates anticipated and correct executions, but is less general

Related Work – Safe C Compilers

- Jones and Kelly [AADEBUG 1997], enhanced by Ruwase and Lam [NDSS 2004]
- Austin et. al [PLDI 1994]
- Yong and Horwitz [FSE 2003]
- Necula et al [POPL 2002]
- Jim, Morrisett et al [USENIX 2002]

Buffer Overflow Detection Tools

- StackGuard [Cowan et al, USENIX 1998]
- StackShield [http://www.angelfire.com/sk/stackshield/]
- Purify [Hastings and Joyce, USENIX 1992]
- Program shepherding [Kiriansky, Bruening, Amarasinghe, USENIX 2002]
- Rebooting, checkpointing, manual error detection and repair etc.

Extensible Arrays

- Many languages provide some form of extensible arrays – e.g. Java
- BMB
  - Preservation of the address space from the original implementation
  - Efficiency – allocates only elements which are actually accessed
  - Avoids denial of service attacks

Conclusion

- Boundless Memory Blocks
  - Eliminates security vulnerabilities and data structure corruption
  - Enhances availability
- Implementation
  - Store out of bounds writes in a hash table
  - Retrieve value from the hash table for out of bounds reads
- Net Effect
  - Give each data block its own address space
  - Address spaces dense in the middle, sparse everywhere else

Questions