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
p = malloc(10);
CRED

Address Space

Block₁  Block₂  Block₃  Block₄

p = malloc(10);
q = p+15;

Object Table

<table>
<thead>
<tr>
<th>Base</th>
<th>Extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

OOB object

<table>
<thead>
<tr>
<th>Address</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
p = malloc(10);
q = p + 15;
r = q - 10;
```c
p = malloc(10);
q = p+15;
r = q - 10;
*q = 100; // Error
```
p = malloc(10);
q = p + 15;
r = q - 10;
*q = 100;
BMB Compiler

Address Space

\[
p = \text{malloc}(10);
q = p + 15;
r = q - 10;
*q = 100;
v += *q;
\]

Object Table

<table>
<thead>
<tr>
<th>Base</th>
<th>Extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>

Base | Offset | Value
---|--------|-------
15  | 100    |
Net Effect of Our Technique

Block 1  Block 2  Block 3  Block 4

sparse  dense  sparse

Block 1

Block 2

Block 3

Block 4
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

This is an exploit!! When this message is received, Pine crashes and won't start again until this message is manually removed from the mailbox file.
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

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

<table>
<thead>
<tr>
<th>Secure</th>
<th>Pine</th>
<th>Mutt</th>
<th>Apache</th>
<th>Sendmail</th>
<th>MC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

GCC | CRED | BMB |
## Results

<table>
<thead>
<tr>
<th></th>
<th>Secure</th>
<th>Continues Correctly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine</td>
<td>![X]</td>
<td>![X] ![X] [✓]</td>
</tr>
<tr>
<td>Mutt</td>
<td>![X]</td>
<td>![X] ![X] [✓]</td>
</tr>
<tr>
<td>Apache</td>
<td>![X]</td>
<td>![✓] ![✓] [✓]</td>
</tr>
<tr>
<td>Sendmail</td>
<td>![X]</td>
<td>![X] ![X] [✓]</td>
</tr>
<tr>
<td>MC</td>
<td>![X]</td>
<td>![X] ![X] [✓]</td>
</tr>
</tbody>
</table>

GCC  | CRED  | BMB  
---  | ---   | ---  
---  | ---   | ---  
---  | ---   | ---  
---  | ---   | ---  
---  | ---   | --- 
## Results

<table>
<thead>
<tr>
<th></th>
<th>Secure</th>
<th>Continues Correctly</th>
<th>Initializes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Mutt</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Apache</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Sendmail</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>MC</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

### Details:

- Pine: Secure ✗, Continues Correctly ✗, Initializes ✗
- Mutt: Secure ✗, Continues Correctly ✗, Initializes ✗
- Apache: Secure ✗, Continues Correctly ✗, Initializes ✗
- Sendmail: Secure ✗, Continues Correctly ✗, Initializes ✗
- MC: Secure ✗, Continues Correctly ✗, Initializes ✗
### Results

<table>
<thead>
<tr>
<th></th>
<th>Secure</th>
<th>Continues Correctly</th>
<th>Initializes</th>
<th>Correct For Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine</td>
<td>☒ ☑ ☑</td>
<td>☒ ☒ ☑</td>
<td>☑ ☑ ☑</td>
<td>☒ ☒ ☑</td>
</tr>
<tr>
<td>Mutt</td>
<td>☒ ☑ ☑</td>
<td>☒ ☒ ☑</td>
<td>☑ ☑ ☑</td>
<td>☒ ☒ ☑</td>
</tr>
<tr>
<td>Apache</td>
<td>☒ ☑ ☑</td>
<td>☑ ☑ ☑</td>
<td>☑ ☑ ☑</td>
<td>☒ ☒ ☑</td>
</tr>
<tr>
<td>Sendmail</td>
<td>☒ ☑ ☑</td>
<td>☒ ☒ ☑</td>
<td>☑ ☑ ☑</td>
<td>☒ ☒ ☑</td>
</tr>
<tr>
<td>MC</td>
<td>☒ ☑ ☑</td>
<td>☒ ☒ ☑</td>
<td>☑ ☑ ☑</td>
<td>☒ ☒ ☑</td>
</tr>
</tbody>
</table>
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

![Graph showing performance comparison between different applications]

- Pine
- Mutt
- Apache
- Sendmail
- MC

Performance metrics include read, move, small page, large page, receive small, receive large, send small, send large, copy, move, mkdir, and delete operations.
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