

Run-time Support for Detection of Memory Access Violations to Prevent Buffer Overflow Exploits

Pramod Ramarao, Akhilesh Tyagi & Gyungho Lee
Department of Electrical & Computer Engineering,
Iowa State University, USA.

6th Information Security Conference Oct 1-3, 2003.

Talk Outline

- ❑ Memory Access Violations
- ❑ Overview of Buffer Overflow Exploits
- ❑ Proposed Solution
- ❑ Some results
- ❑ Conclusion

Memory Access Violations

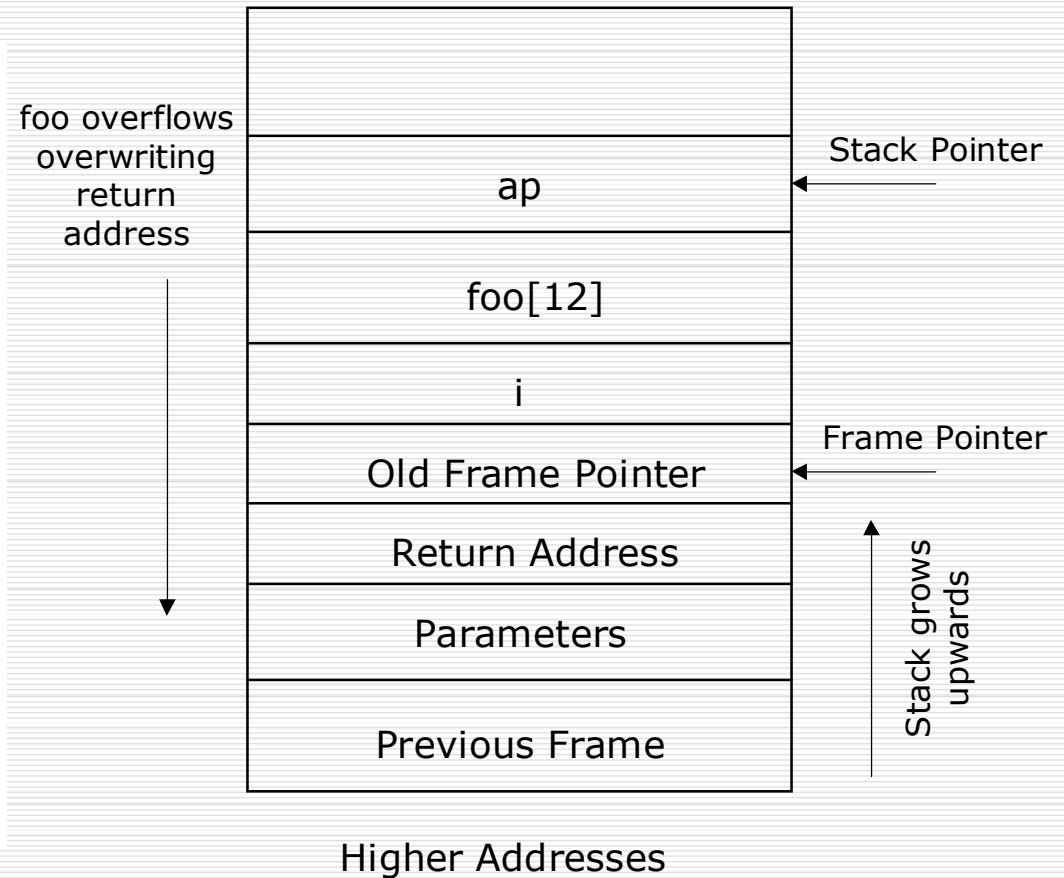
- What are memory access violations ?
 - Reading or writing past the end of array storage.
 - Pointer-based accesses to memory beyond allocated space.
- Leads to vulnerabilities in software.
 - Security exploits like Buffer Overflows.

Memory Access Violations (cont...)

- Checking for violations is expensive.
 - Need to track every pointer and maintain pointer-object mapping.
 - Pointers in C can be cast in different contexts within the same expression.
- Existing solutions.
 - Overhead is too high (e.g. 30x).
 - Coverage is not complete.

Overview of Buffer Overflows

```
char sc[]=
"\x31\xc0\x50\x68//sh\x68/bin\x89
\xe3\x50\x53\x89\xe1\x99\xb0\x0b
\xcd\x80";
char large_str[50];
void main()
{
int i;
char foo[12];
int *ap = (int *)large_str;
for (i = 0; i < 50; i += 4)
}
*ap++ = sc[i];
strcpy(foo, large_str);
```

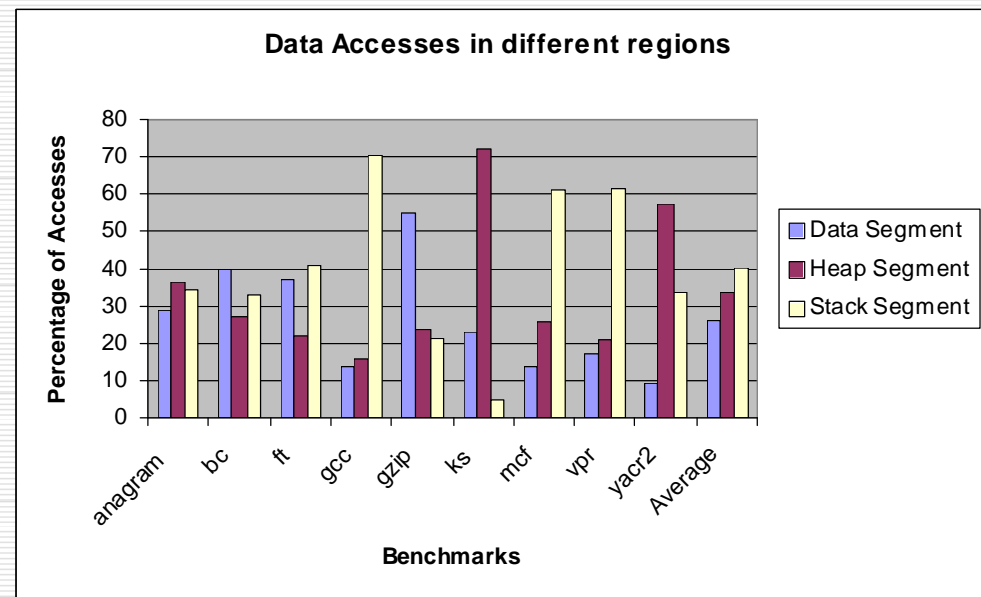


Proposed Solution

- Two-pronged approach.
 - Hardware checking for stack addresses.
 - Software checking for static/heap.
- Hardware checking can be done at almost no overhead.
 - Reduces overhead considerably.
- Checking by compiler for all static and heap addresses.
 - Average 2x overhead.

Proposed Solution

- Data accesses to various regions in benchmarks.



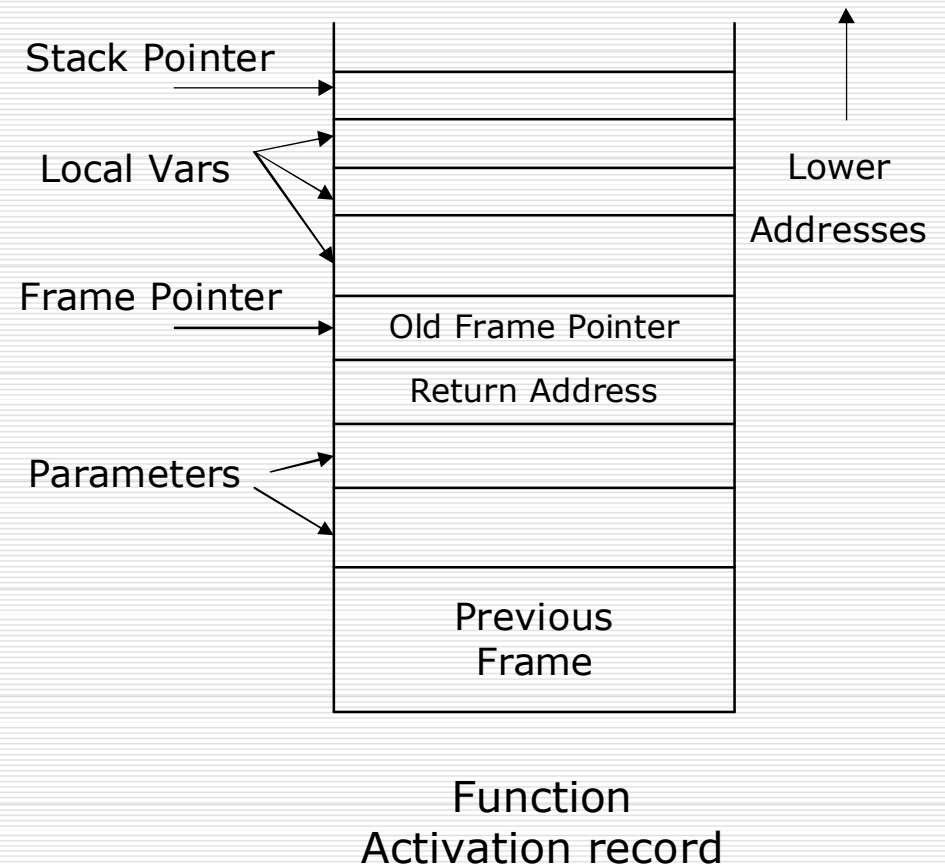
Hardware Approach

- Percentage of aggregates allocated to stack in benchmarks.
- Since 40% of accesses are stack based, complexity of checking drops by 60%. (refer paper)

Benchmark	% Allocated
anagram	25%
bc	29%
ft	0%
ks	8%
gcc	54%
gzip	28%
mcf	25%
vpr	89%
Average	32%

Hardware Approach (cont...)

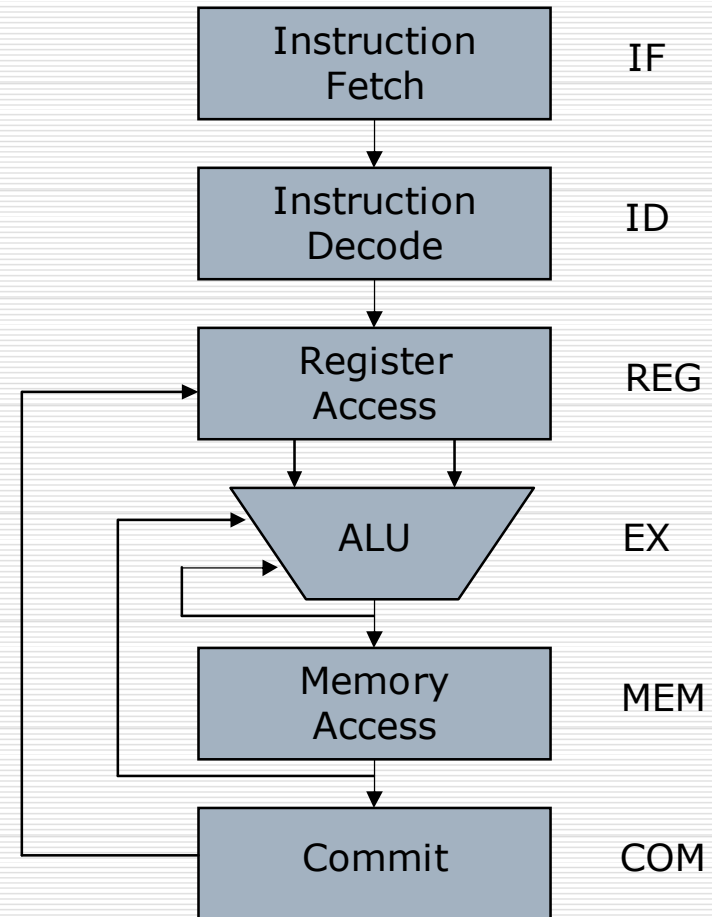
- For all stack addresses, bounds available in the stack pointer and frame pointer.
- Use a checking condition during EA calculation.



Hardware Approach (cont...)

- To allow access to parameters, store size of the parameters passed to the function on the stack.

```
if (EA is stack address)
  if ($sp < EA ) or
    ($fp < EA)
    if (EA > $fp+size)
      raise an exception
    endif
  endif
endif
```



Hardware Approach (cont...)

- EA can be determined to be a stack address easily within the micro-architecture.
- Advantages
 - No source code modification required.
 - Since checking is overlapped with other activities such as memory access itself, almost zero overhead is incurred.

Software Approach

- ❑ 30% of data accesses are into the data segment region.
- ❑ Compiler checks pointer-based accesses to data and heap segments.
 - Currently checking only pointer-based accesses to data segment has been implemented.
- ❑ The GNU C Compiler is augmented with providing checking functionality.

Software Approach (cont...)

- Two tables are maintained dynamically.
 - A data object bounds table which includes the characteristics of the data structure being tracked.
 - A pointer-object map table which maintains the mapping between a pointer and object during program execution.

Software Approach (cont...)

- Use a space and time efficient hashing technique to manage tables. (refer paper for details)
- Advantage of the hashing technique
 - Facilitates only pointer dereferences to be checked. There is no need to check every pointer arithmetic operation.

Software Approach (cont...)

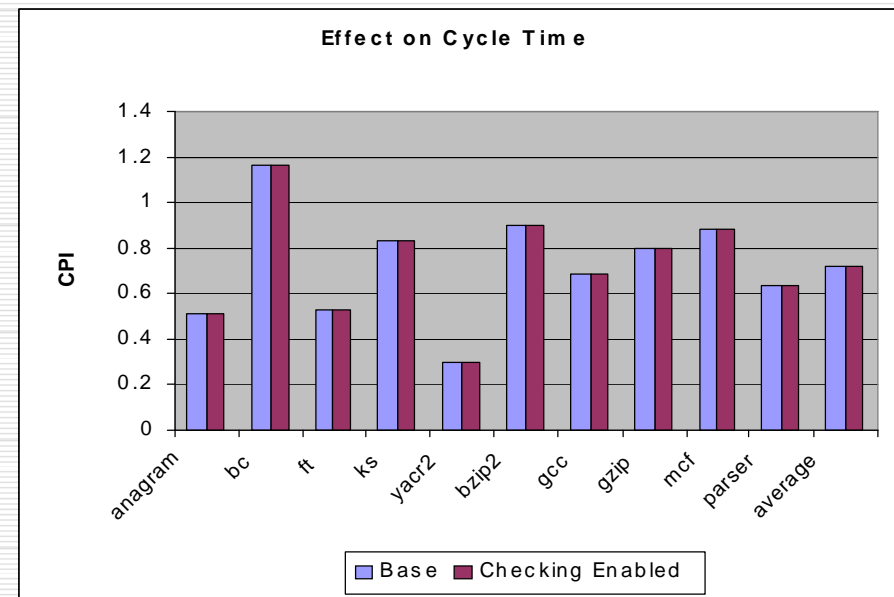
- Advantages of the software approach
 - Works on the IR of the compiler, so this technique is not bound to a particular language implementation.
 - Resulting programs are compatible with existing libraries.

Results

- Hardware Approach.
 - Tested against the set of buffer overflow exploits available in the Libsafe package.
 - Combination of pointer-intensive benchmarks and benchmarks from the SPEC2000 suite.
- Able to detect all exploits in the Libsafe package and raised a machine exception.

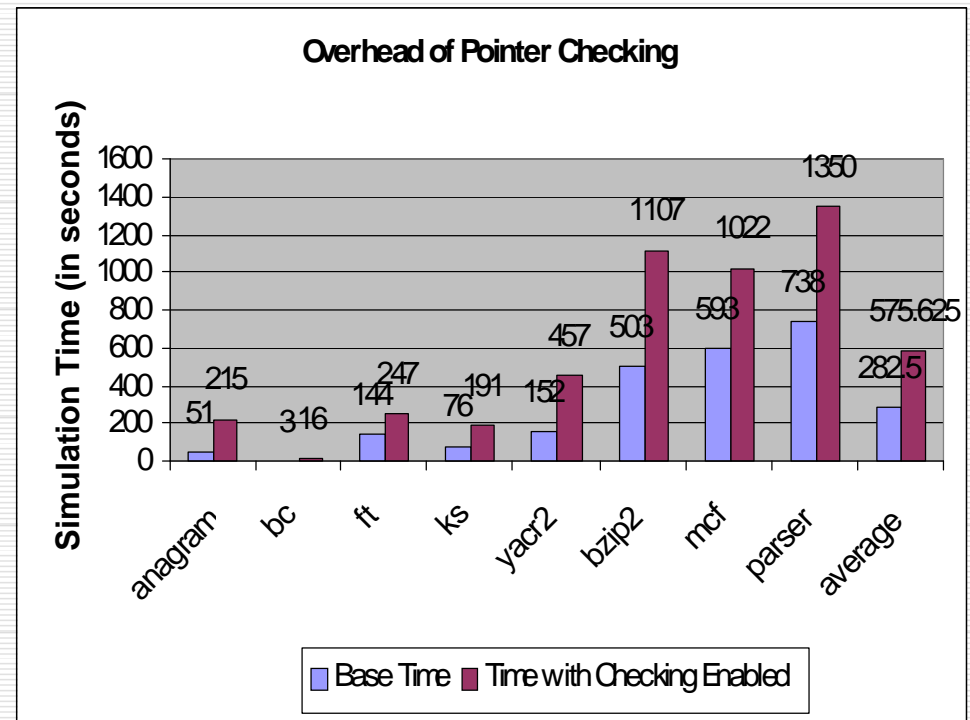
Results (cont...)

- Effect on the cycle time. (increase in cycle time leads to performance degradation).
- Technique has almost no overhead.



Results (cont...)

- Software Approach.
 - All the benchmarks are compiled with the modified compiler and benchmarks are simulated on the modified architecture with checking.



Conclusion

- ❑ A new approach combining both hardware and software techniques to prevent buffer overflow exploits.
- ❑ The hardware approach has almost no overhead.
- ❑ The software approach has an overhead of 2x on the average.