

Making The Kernel Responsible: A New Approach To Detecting & Preventing Buffer Overflows

William R Speirs

The Advanced Technology Research Center,
The SYTEX Group Inc.

Presentation Overview

- Problem Description
- The Solution Idea
- The `ptrbounds` System Call
- Limitations & Improvements
- Questions

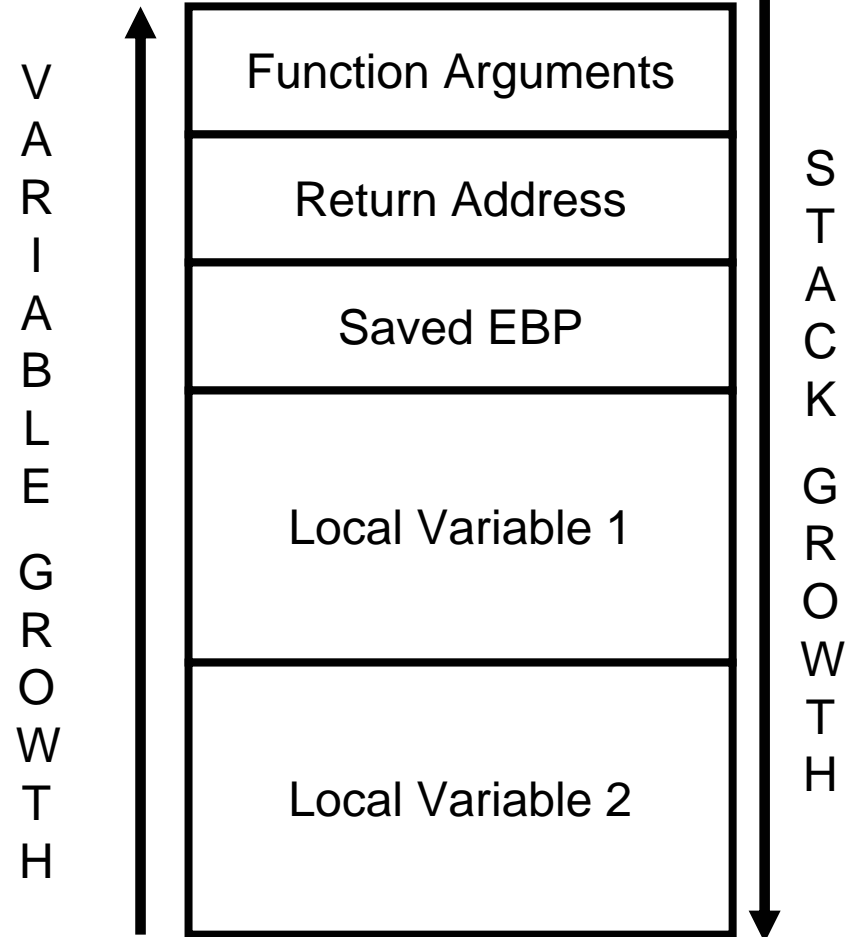
Problem Description

Where And How Do Buffer Overflows Occur?

1. Stack based buffer overflows
 - Local (automatic) variables allocated by the compiler for use inside a function
2. Heap based buffer overflows
 - Variables allocated at runtime by memory allocation functions (`malloc`, `new`, etc)
3. Data segment based buffer overflows
 - Global or static variables allocated by the OS when a program is loaded into memory

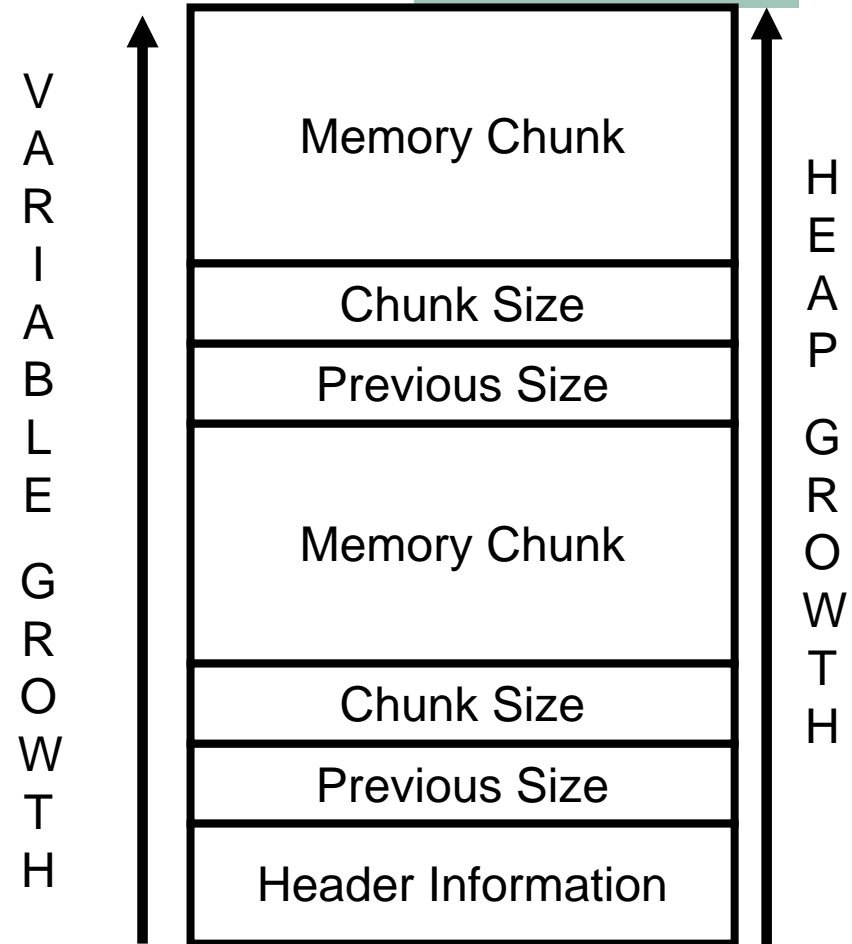
Stack Based Overflows

1. A local variable (usually char buffer) is allocated on the stack by the compiler during a function call
2. More data than space available is written to the variable's memory location
3. Any data following the variable is overwritten



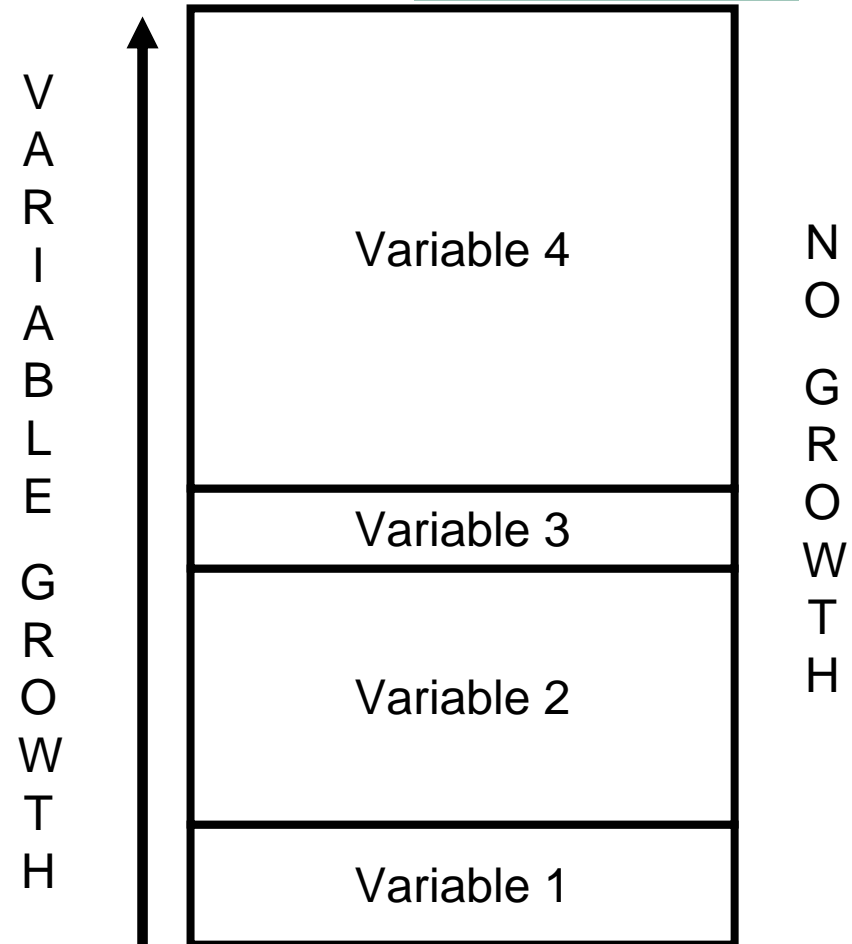
Heap Based Overflows

1. A chunk of memory is allocated on the heap by the programmer during runtime
2. More data than space available is written to the allocated memory
3. The chunk information following the chunk is overwritten



Data Segment Based Overflows

1. A section of memory is allocated by the OS when the program is loaded
2. More data than space available is written to the allocated memory
3. Other variables allocated in this region are overwritten



The Common Thread

- More data is written than space allocated
- Other pieces of memory, allocated to other variables, are overwritten
- In almost all cases, these overflows occur during runtime by user inputted data
- The programmer is usually trying to prevent these overflows from occurring

Overall Problem

- It is impossible to account for all possible situations that might arise when writing data to memory
- Currently no method is provided to the programmer to check the bounds of allocated memory
- Kernel does not track the bounds of variables, only segments of memory

The Solution Idea

Make The Kernel Responsible

- The kernel should be responsible for the proper memory management of all processes
- Have the kernel check the bounds of all pointers passed to system calls before writing data
- Enable the kernel to provide information about variables to the granularity of a byte, not a segment

Have The Compiler Aid The Kernel

- Provide stack variable information to the kernel that is normally lost during compilation
- Automatically add instructions to function calls that update the stack variable information
- Provide static and global variable information to the granularity of a byte during compilation

Provide Information To Programmers

- The programmer generally does not want overflows
- Provide programmers with bound information for variables
- Provide the bound information on a per-pointer basis so it can be used by library, kernel, and application programmers alike
- Created a new system call, `ptrbounds`

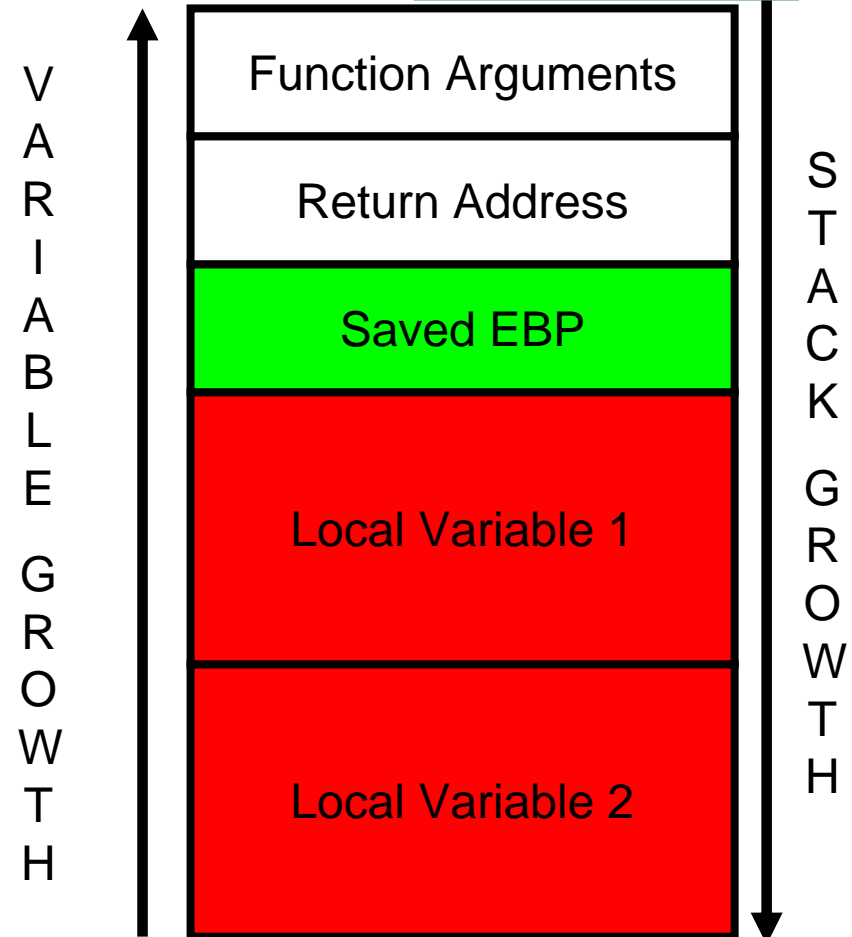
The ptrbounds System Call

ptrbounds Overview

- Provide upper and lower bound of any pointer of memory
- Provide the safest information available to the kernel
- **Prototype:** `int ptrbounds(void *ptr, void **lower, void **upper, void *seg);`
 - `*ptr` - The pointer in question
 - `**lower` - The lower bound of the ptr
 - `**upper` - The upper bound of ptr
 - `*seg` - The segment the pointer points to

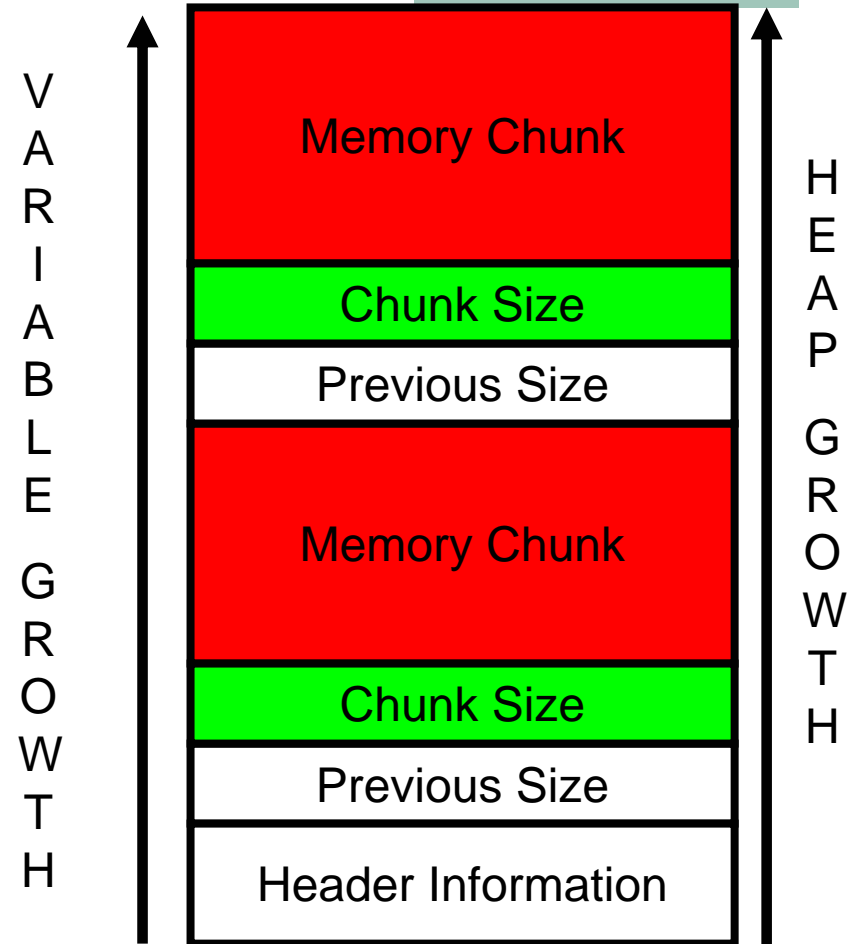
Stack Variables

- Returns the upper and lower bound of all allocated local variables for that function
- Saved frame pointers are used to walk back through the stack



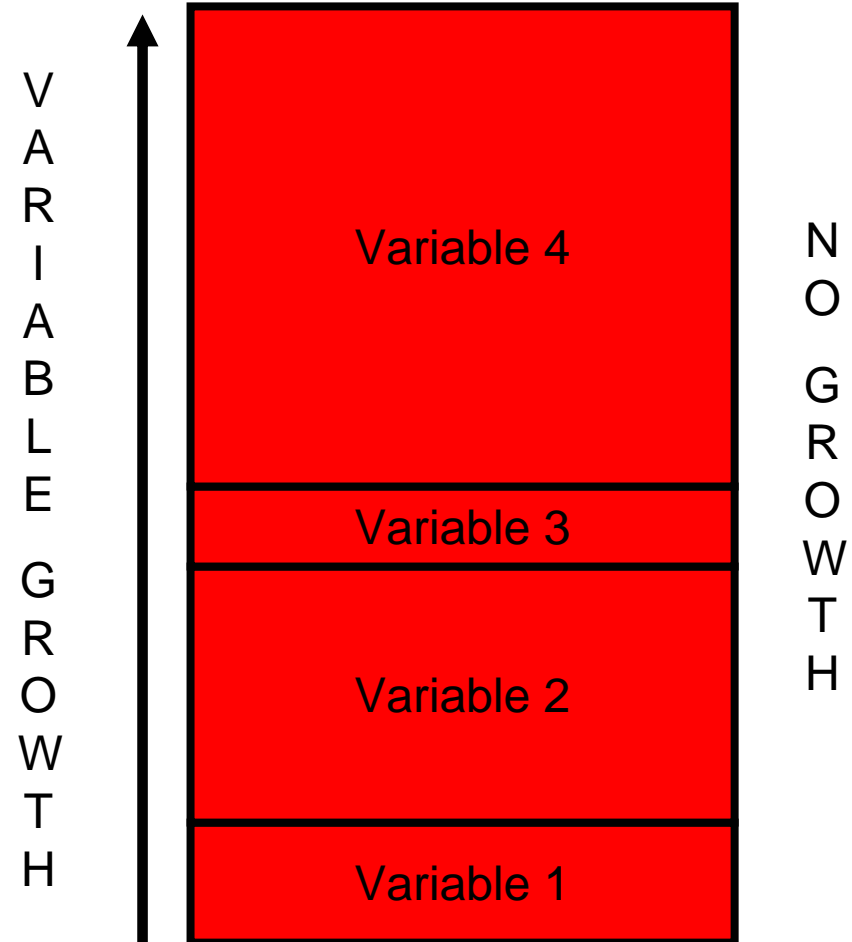
Heap Variables

- Returns the upper and lower bound of memory allocated to the variable pointed to
- The size of the variable is provided by the `size` information in the chunk headers



Data Segment Variables

- Returns the upper and lower bound of the entire data segment
- Information on the bounds of the data segment are provided by kernel data structures



Segment Values

- Shared libraries
- Stack structure
- Heap structure
- Code segment
- Kernel memory
- Unallocated memory
- Heap memory
- Stack memory
- Data segment



Limitations & Improvements

Limitations

- Stack Variables
 - Cannot obtain byte level granularity
- Heap Variables
 - Granularity is rounded to nearest 32 bit quantity, but always safe
- Data Variables
 - Bounds are for the whole data segment, not for each variable

Improvements

- Stack Variables
 - Have the compiler provide a table that contains the bounds of each locally allocated variable
- Heap Variables
 - Make (`malloc`, `new`, etc) a system call so the kernel knows about all allocated memory
- Data Segment Variables
 - Have the kernel parse the executable creating a table of the bounds of data segment variables as it is loaded into memory

Questions ?