Buffer Overflow

KAMI VANIEA

JANUARY 28TH 2016

Some slides copied from Myrto Arapinis' talk last year

First, the news…

Heartbleed

- https://xkcd.com/1354/
- http://heartbleed.com/
- https://www.us-cert.gov/ncas/alerts/TA14-098A

Data != Code

Poor programing decisions can lead to the computer executing the contents of data

Secure programming is really all about those bad programming practices we told you not to do back in your first programming class.

- Check for divide by 0
- Integers have a maximum size, don't go over it
- If you allocate an array of 128 things don't put 129 things in it
- If you are a reading from an array of 128 things, don't try and read the 129th
- Users put bad stuff into the input, always check the input
- Other people write poor code, if you get a value back from a library, check it

Simple (complex) example

- The following is a worked example of how memory accesses of the stack are supposed to work
- I want you to understand three facts from this example:
  1. Return addresses locations are not special, any assembly code can write to them
  2. At assembly level, code looks like a pile of GoTo statements
  3. Memory boundaries between variables are not strongly enforced

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

Original code snippets (left) compile into the assembly code (bottom).

Compiled Code

Caller

```
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   mov1 %ecx, %edx  // unrelated b=a code
```

…

Callee

```
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

Note that this code is intended to provide a clear example and makes somewhat liberal use of pseudocode.

Compiled Code

Caller

```
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   mov1 %ecx, %edx  // unrelated b=a code
```

…

Callee

```
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

The stack (right) grows downwards, starting with a high memory address and progressing towards a smaller memory address.

Compiled Code

Caller

```
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
```

…

Callee

```
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	
…	
64	
60	
56	
52	
48	

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

Our example uses four registry values (bottom right).

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	
…	
64	
60	
56	
52	
48	

Register Values

%ebp	
%esp	
%eip	
%eax	

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

1. Push arguments onto the stack (in reverse)

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	
56	
52	
48	

Register Values

%ebp	128
%esp	64
%eip	
%eax	?

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

1. Push arguments onto the stack (in reverse)

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   mov1 %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	
52	
48	

Register Values

%ebp	128
%esp	60
%eip	
%eax	?

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
    f(arg1, arg2);
    b = a;
Callee
    f(arg1, arg2){
        return arg1 + arg2;
    }
```

2. Push the return address. i.e. the address of the instruction to run after control returns

Compiled Code

```
Caller
    pushl arg2
    pushl arg1
    call f
        // push address of mov instruction
        // %eip <- address of f()
    movl %ecx, %edx  // unrelated b=a code
…
Callee
    pushl %ebp  // save caller base pointer
    movl %esp, %ebp  // %ebp <- %esp
    movl 12(%ebp), %eax  // store arg2 in %eax
    addl 8(%ebp), %eax  // add arg1 to %eax
    popl %ebp
        // %ebp <- (%esp)
        // %esp <- %esp + 4
    ret
        // %eip <- (%esp)
        // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	
48	

Register Values

%ebp	128
%esp	60
%eip	
%eax	?

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

3. Jump to the function's address

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	
48	

Register Values

%ebp	128
%esp	60
%eip	
%eax	?

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

4. Push the old fame pointer to the stack

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	128
%esp	52
%eip	
%eax	?

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

5. Set frame pointer to where the end of the stack is right now

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	52
%esp	52
%eip	
%eax	?

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

6. Do the local computation (addition in this example)

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	52
%esp	52
%eip	
%eax	arg2

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

6. Do the local computation (addition in this example)

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	52
%esp	52
%eip	
%eax	arg2+arg1

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

7. Reset the previous stack frame

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	128
%esp	52
%eip	
%eax	arg2+arg1

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

7. Reset the previous stack frame

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	128
%esp	56
%eip	
%eax	arg2+arg1

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

8. Jump back to the return address

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	128
%esp	60
%eip	
%eax	arg2+arg1

Base

Stack

Instruction

Returned val

Original Code Snippets

```
Caller
   f(arg1, arg2);
   b = a;
Callee
   f(arg1, arg2){
      return arg1 + arg2;
   }
```

9. Start executing line after the original call

Compiled Code

```
Caller
   pushl arg2
   pushl arg1
   call f
      // push address of mov instruction
      // %eip <- address of f()
   movl %ecx, %edx  // unrelated b=a code
…
Callee
   pushl %ebp  // save caller base pointer
   movl %esp, %ebp  // %ebp <- %esp
   movl 12(%ebp), %eax  // store arg2 in %eax
   addl 8(%ebp), %eax  // add arg1 to %eax
   popl %ebp
      // %ebp <- (%esp)
      // %esp <- %esp + 4
   ret
      // %eip <- (%esp)
      // %esp <- %esp + 4
```

Stack

Address	Value
128	return address
…	
64	arg2
60	arg1
56	Return address
52	Caller %ebp //128
48	

Register Values

%ebp	128
%esp	60
%eip	
%eax	arg2+arg1

Base

Stack

Instruction

Returned val