Offensive Computer Security: Summary 2

David De Lille

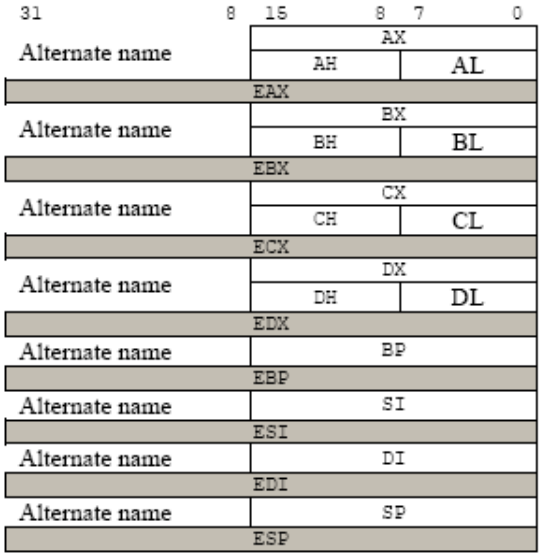
March 27, 2015

1 Intro to CPU and registers

1.1 Registers

64-bit register	lower 32-bit	lower 16 bit	lower 8 bit
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

- eax: accumulator (return value)
- ebx: base
- ecx: count
- edx: data
- ebp: base pointer
- esi: source index
- edi: destination index
- esp: stack pointer
- eip: instruction pointer



2 Why learn C?

- one of the most widely used programming languages
- other popular languages borrow from it
- used in everything (OS, embedded systems, drivers, libraries, other prog. languages)

3 Strings

3.1 String types

A *string* is an array of characters up to and including the terminating null character. *Length* = the number of characters including the null character. *Count* = number of elements in the array. *Size* = the number of bytes allocated to the array. Size of a string depends on the size of the characters and can be larger than the length.

Character types:

- [signed/unsigned] char
- [signed/unsigned] wchar_t (not meant to be signed/unsigned)

wchar_t is an integer type that can represent the largest character set among the supported locales. On Windows, it uses UTF-16 (2 bytes per character). On Linux, it uses UTF-32 (4 bytes per character).

3.2 String functions

Length/Size functions:

- strlen (run time)
- sizeof (compile time)
- wcslen (for wide characters)

Copying:

- memcpy: Copy block of memory
- memmove: Move block of memory
- strcpy: Copy string (unbounded)
- strncpy: Copy characters from string
- strcpy_s: (A windows function, not C99/C11)
- POSIX function (not C99/C11): strdup
- Windows functions (not C99/C11): wcscpy, wcscpy_s, _mbscpy, _mbscpy_s

Concatenation:

- strcat: Concatenate strings
- strncat: Append characters from string
- sprintf: Format strings (also copying)
- snprintf: Format strings (also copying)

3.3 Common errors/vulnerabilities

3.4 Improperly bounded string copies

The functions gets (cannot be used safely) and strcpy (can be used safely) are deprecated. However, misuse of their replacements can also be unsafe: strncpy, memmove, memcpy, sprintf, snprintf. Example on slide 24-27.

3.5 Off-by-one errors

This involves reading/writing outside the bounds on an array. Example on slide 29.

3.6 String truncation

Data is lost when a too large strings is put (safely) into too small of a destination. This can lead to errors in the application logic.

3.7 Null termination errors

When a string isn't properly null terminated. The functions strncpy and strncat don't null terminate.

3.8 Preventing errors/Mitigations

- use encoding best practices
- use safe function correctly
- use a unified model for handling strings (see slide 70)
- use stack cookies (more details in further lectures)

4 Pointers

4.1 Asterisk operator: *

When used for declaring, it instantiates a variable pointing to an object of the given type.

When used an a unary operator, it denotes indirection. If this operator doesn't point to an object or function, the behaviour is undefined. Dereferencing NULL is a vulnerability if it is a memory-mappable address.

4.2 Ampersand operator: &

This operator shows the actual data stored in a pointer. It is used to get the address of an object.

4.3 Member-of operator: ->

This deferences a structure and accesses a member of that structure.

4.4 Array indexing: x[y]

This calculates the address based on the address of x and the value of y.

4.5 Function pointers

This type of pointer points to executable code in memory. This can be very dangerous if it can be made to point to malicious code. Example on slide 42. Notice the use of register al on slide 45, line 14 in the Assembly code.

4.6 Global Offset Table (GOT)

This is the section of an ELF-file (Executable and Linking Format) that holds absolute addresses for all accessed global variables, and is essential for dynamically linked libraries. This is used to make the code position-independent. Executables on Windows use a similar technique for using libraries, but only the Linux version is exploitable.

Before a library function is used, the GOT points to the Run Time Linker (RTL). When the function is called, the RTL finds the actual address and writes it into the GOT. Any subsequent calls use the function's address directly, without involving the RTL.

The GOT is located at a fixed address in every ELF, and it is not write-protected because the RTL needs to be able to change it. If the GOT can be overwritten by an attacker, it can be used to redirect an existing function to malicious shellcode.

4.7 .ctors and .dtors section

This is a section added by the GCC compiler, that contains the destructor function pointers. These function pointers will be executed after main exits. The function pointers in the related .ctors section are executed before main, and are therefore never targeted.

4.8 Find-the-bug exercises

Slide 43-45: The code on the left allocates memory and truncates the address by storing it in a char variable, which is only 1 byte instead of the required 4 or 8 bytes.

Slide 46: ...

Slide 52: The first if-structure checks that a member function isn't NULL. After that, the second one checks that s isn't NULL. If s was indeed NULL, this would mean that the first if-structure has dereferenced s (which is NULL).

Slide 53: The else-clause of the if-structure is only executed if subnet is NULL, but then it calls a member function of subnet (subnet-¿Name()), which would be dereferencing NULL.

Slide 54: (I think the "mp" seen on the slide should be "tmp".) I think the bug is in the dereferencing of tmp, without checking that it might be NULL. Another possibility is that the member mListNodePrev might be NULL, which would cause a NULL-derefence later on.

5 Dynamic memory management

5.1 C memory management

5.1.1 C99 memory allocation functions (heap)

- malloc
- alligned_alloc (see below)
- realloc: change the size of memory
- calloc: allocates and sets to 0

5.1.2 Alignment

This is a restriction on the memory address of certain objects in older systems. Objects have to lie neatly in aligned byte slots. Attempts to use misaligned data would result in a bus error and terminate the program. Modern systems are able to handle it correctly (but slower). This can be important for exploitation.

5.1.3 Alloca

This is a function that uses the stack for dynamic memory allocation. It isn't part of the C99 standard. Obviously this can cause stack overflows.

5.2 Common errors

5.2.1 Initialisation Errors

Not properly initialising memory. Malloc and free do not zero out memory.

5.2.2 Failure to check return values

Assume that the function will succeed can result in edge-case bugs. Malloc will return NULL if there is not enough free heap memory.

5.2.3 Using Freed memory

Unless all pointers have been NULLed or invalidated, it's still possible to access the freed memory. Example on slide 60-61. This leads to a vulnerability (see next summary).

5.2.4 Multiple frees on same memory

This is usually caused by careless copy-pasting or sloppy error-handling. It can result in a corrupted heap memory manager, segfaults, vulnerabilities, or memory leaks.

5.2.5 Memory Leaks

A memory leak happens when a program fails to free dynamically allocated memory after it has been used. The result is that this memory becomes unavailable to the program or the rest of the system, leading to memory exhaustion. If an attacker can exploit memory leaks, it can be used for an DOS attack.

5.2.6 Dereferencing a NULL pointer

A null-pointer dereference takes place when a pointer with a value of NULL is used as though it pointed to a valid memory area. If an attacker can intentionally trigger a null pointer dereference, the attacker might be able to use the resulting exception to bypass security logic or to cause the application to reveal debugging information that will be valuable in planning subsequent attacks.

5.3 Memory allocator

(see next summary)

5.4 Mitigations

- set pointers to NULL after freeing them
- use ASLR (see later)
- testing

6 Required reading: HAOE 0x260 up to 0x280

I'll just briefly state what each subchapter talks about, since it's mostly C basics.

0x261 Strings

Strings. GCC -o flag. NULL terminators. The stack stores return addresses when calling functions.

0x262 Signed, Unsigned, Long, and Short

Numerical values are signed by default. Two's complement. Use sizeof() to find the actual size for that architecture.

0x263 Pointers

Use pointers instead of copying large amounts of data. Asterisk and ampersand operators.

0x264 Format Strings

Format string = character string with escape characters to insert variables in a certain format. Some format parameters are listed.

0x265 Typecasting

Typecasting = temporarily change a variable's data type. Void pointer. Compiler is the only thing that cares about data types.

0x266 Command-Line Arguments

Argc and argv. argv[0] = name of binary. Too few program arguments can result in a SEGFAULT.

0x267 Variable Scoping

Contex. Local and global variables. Stack frames. Static variables.

0x270 Memory Segmentation

Five segments: text (code), data (initialised global/static vars), bss (uninitialised global/static vars), heap (dynamic memory), and stack (local variables and stack frames). Stack abstract data structure. Frame pointer or local base pointer. Figure of a stack frame (p73). Relative location of the 5 segments (Figure p75).

0x271 Memory Segments in C

Which variables get stored in which segment.

0x272 Using the Heap

malloc() and free().

0x273 Error-Checked malloc()

Use a function to avoid duplicated error checks around mallocs.

7 Other notes

- Godbolt is a tool that visualises what C/C++ code corresponds to what Assembly code.