Memory protection

Heap memory management

a.

(i)

With reference counting the programmer only shares whether they still need the memeory and the operating system decides if the memeory can be freed and then does so itself. This comes with limitations in situations such as circular references.

(ii)

Mark-and-sweep overcomes the limitation of circular references for reference counting. On the other hand, it induces a certain runtime overhead since all allocations need to be checked.

The stack

a.

The stack frame contains the functions arguments, the old base pointer, the return address and local data.

E.g., from Tannenbaum in Modern Operating Systems:

> A procedure's stack frame holds those input parameters, local variables, and temporary variables that are not kept in registers.

or an excerpt form Hacking the Art of Exploitation

> Each stack frame contains the parameters to the function, its local variables, and two pointers that are necessary to put things back the way they were: the saved frame pointer (SFP) and the return address. The SFP is used to restore EBP to its previous value, and the return address is used to restore EIP to the next instruction found after the function call.

b.

Stack canaries protect against buffer overflows by placing a canary value after the local variables. Commonly, this is done immediately after the function prologue. Immediately before the function epilogue, the canary value is retrieved and compared. If it does not match, it suggests that the initial value has been modified and undefined behavior occurred.

c.

Assuming an arbitrary-write primitive, an attacker is able to overwrite the return address and the return address only. While a shadow stack will detect the attack since the return address changes, stack protector will not be able to since the canary value did not change. Clearly, stack canaries are most effective when an attacker has to overwrite a continous interval of memory starting from the current stack frame to the return pointer.

Hardware mechanisms

a.

(i)

Those jumps are hardcoded at compile time, hence, they are not vulnerable to runtime-attacks.

(ii)

Yes, since in the presence of attacks against hardware shadow stacks this approach can be considered useful in terms of a defense-in-depth strategy. Furthermore, control-flow can also be hijacked in other ways than overwriting the return address which is not defended by shadow stacks.

b.

Using different modifiers is used to restrict the valid scope of pointer. Using the same modifier would allow an attacker to use an authenticated pointer for a purpose it was not intended to be used.

> A modifier value. This is used to determine the "context" of a pointer so that an authenticated pointer can't be taken from one place and reused in another (more on this later). There are some special-case instructions that are hard-coded to use e.g. the stack pointer or zero as the modifier. The modifier is used as the "tweak" for the tweakable MAC computation.

> Much of the difference in security of PA-based protection schemes comes from the choice of modifier. A modifier should be outside the attacker's control, as well as quick and easy to compute from the available data, but if modifiers coincide too often, then this gives an attacker too many opportunities to reuse pointers outside the context that they are meant to be used in.

c.

An example would be if the heap memory is freed and that memory section gets allocated to an already exisitng memeory segment with a different color for a realloc call.

d.

Consider the von-neumann architecture. Data and instruction or other necessary primitives are not distinguishable. CHERI adds this layer such that there is no unintended modification of pointers when intending to manipulate data, hence, data and pointers are seperated.

Miscellaneous

a.

Subtractive attacks work by using mulitple watermarked binaries to remove the watermark. On the other hand, distortion attacks essentially apply an obfuscation step to hide the original watermarking, hence, rendering it unrecognizable.

b.

The purpose of an intermediate representation is to have a lower-level code representation on which first optimization steps can be done while still keeping it architecture independent. This can also be used for program analysis.

c.

Data in memory is suspect to memory corruption vulnerabilites.

Memory Protection CFI Analysis

a.

(i)

w0-1>3 or w0>4 or choice>4

(ii)

206-209: (x0-((x10=adrp x10)+1676))<<2 213: x10>=4

$(x0-((page_address $pc)+1676))>>>2 >= 4$

b.

Target address must be at most 15 bytes from the current pc+1676 distant and the two least significant bits are set to zero.

So the possible target addresses are:

0x40068c 0x400690 0x400694 0x400698

The code at the possible target addresses redirect the control flow to one if the foo_impl* functions.

c.

The compiler raises an exception if the condition from a(ii) is fulfilled at 0x400688.

d.

Using essentially a trampoline enables optimization such that the course-grained check from above is possible and it is not necessary to check that the returned address is one of the 4 foo impementations in four different checks.