Safe Heap: Malloc With Canaries

Måns Abrahamsson, Emil Göransson, Lukas Lannge, Samuel Sendek

Fall 2025

1 Description

This project involves implementing a secure heap memory allocator within the S3K kernel environment. The core objective is to create the standard memory allocation function (`malloc`) with caneries to detect memory corruption caused by buffer overflows. Canaries are used as the hardware platform is very basic and lacks support for virtual memory and only has a limited number of PMP regions.

2 Possible Vulnerabilities Countered

- Adding such a function helps protects against buffer overflow attacks, denying the possibility for a malicious actor to corrupt adjacent metadata or data belonging to other, unrelated allocations.
- The use of canaries is not a perfect solution. The overflow will only be detected during the next context switch
- Real world vulnerabilities that could be prevented:
  - CVE-2023-4863, a heap overflow vulnerability in the chromium web browser
  - *Eternal Blue* used a number of bugs, one of them being a buffer overflow.

3 Minimal Requirements

The minimal requirements would involve creating a memory allocation function, and protecting heap objects using the method described in heapsentry [1].

The first thing that has to be implemented is the basic heap functionality. This is done by using an allocation algorithm and a data structure that tracks which portions of the acquired heap objects are allocated and which memory regions are free. This structure can be extended by including canaries after each object, consisting of a randomly generated magic string that is used for detecting buffer overflows.

To detect overflows, the kernel has to be extended to verify the magic numbers in each canary at each context switch. If a canary has been modified, the process should panic.

3.1 Overflow protection

To prove that the malloc is secure against overflow attacks, we also intent to develop an educational attack, showing how the heap/malloc function is initially vulnerable to overflow exploitation, and later, how the canary implementation protects against it.

4 Order to implement functions

- Create a heap data structure
- `malloc(size)`: Finds a suitable free block, marks it as used, adjusts its size and returns a pointer to the user data area
- `free(ptr)`: The function that marks an allocated block as free and attempts to merge it with adjacent free blocks to prevent memory fragmentation.
- `secure_malloc(size)` Extend the standard malloc to use have Canary value management, that both allows us to generate new canaries, but also some logic for storing our canary values.
- Extend the kernel context switching logic to verify the integrity of all the canaries of the next process.

5 Optional Requirements

We could extend the protected heap objects by using PMP regions in between heap objects that are not readable or writable. If an overflow occurs, an interrupt will imminently occur, making it more protected than simple canaries. On RISC-V processors, there are a limited number of PMP regions that can be defined, often as few as 8-16. Because of this, PMP can not be used between all heap objects, but should only be used for certain objects that are believed to be more vulnerable. We can implement two different malloc functions, a standard malloc, that used canaries, and a secure malloc, "`malloc_protected`" that uses PMP regions for increased security. The PMP regions have to redefined at each context switch, such that each process can use all available PMP regions.

References

[1] Nick Nikiforakis, Frank Piessens, and Wouter Joosen. “HeapSentry: Kernel-Assisted Protection against Heap Overflows”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Konrad Rieck, Patrick Stewin, and Jean-Pierre Seifert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 177–196. ISBN: 978-3-642-39235-1.