Efficient and sequcre memory allocator

/\* TO DO \*/

line 1: 1st Given Name Surname   
line 2: *dept. name of organization   
(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

/\* TO DO \*/

line 1: 4th Given Name Surname  
line 2: *dept. name of organization*  
*(of Affiliation)*  
line 3: *name of organization   
(of Affiliation)*line 4: City, Country  
line 5: email address or ORCID

*Abstract* — /\* TO DO \*/

Keywords — /\* TO DO \*/

# Introduction

Modern applications cannot be imagined without processing large amount of data. Memory allocation is one of major challenges that developers face in programming. Most operation systems are not real time based, so any direct memory request to such OS can interrupt evaluations for undetermined period of time. Moreover, some applications that use custom memory managers in pursuit of efficiency can suffer from attacks like “use-after-free” or overflow-like. Therefore, the goal of this paper is to propose efficient, and secure algorithm of memory allocation.

The rest of the paper is organized as follows. In Section 2, we demonstrate some existing ideas and techniques in this area. In Section 3, we demonstrate methods to memory storing and formulate the allocation algorithm.

# Related work

Since memory allocation problem is ubiquitous in computer science, this academic work to determine the efficient and secure allocation mechanism becomes more important these days. In the following we want to present an overview of related work in this area.

As aforementioned, systems with insufficient user data validation might deal with dynamic memory storage. Dewey et al. (2015) formulates the “use after free” vulnerability and conditions of such attacks. The paper has focused on C++ com-piled binaries where the memory manager cannot find and update pointers to program objects when they are moved. This same vulnerability might be presented in other languages. The authors have demonstrated the example of code with attack proof of concept.

Further, Qiang Zeng et al. (2019) classifies different attack types into such systems. The paper above notices how programs might be patched to gain required defense against “use after free” type. However, the method they mentioned requires 64 bits of metadata for every memory allocation call and 4Kb of guard pages. In this work we propose the algorithm with less memory usage.

The other approach to prevent vulnerability was demonstrated and analyzed by Jonathan Ganz et al. (2017). It suggests randomizing the address space and re-turning random address pointers. Moreover, authors mentioned that most operating systems use current approach. With respect to our work, we also use randomization approach to obtain security.

As mentioned in [4], methods like TSLF are the most effective ones from the time complexity perspective. At the same time the additional memory usage is re-quired due to the hash map. Moreover, Masmano et al. (2004) has introduced the TSLF algorithm that has O(1) time complexity and demonstrated the proof of such behavior. The authors also overviewed the segregated free list like method and compared it with the provided one. It is mentioned that such allocators do not use the hash map and therefore they are free of its memory. Our paper introduces the time and memory balanced protocol of allocation based on segregated free list type.

Another question which corresponds with memory allocation is fragmentation. Nikola Zlatanov (2015) has overviewed this phenomenon and suggested to define a series of partition pools with block sizes in a geometric progression. We suppose that such approach tends to use memory inefficient. Nevertheless, as men-tioned in [7], the current phenomenon can be avoided. In addition, realizations of the best policies are already known and might be implemented on the client side.

In terms of safety and efficiency at the same time, Beichen Liu et al. (2019) has introduced so-called “SlimGuard” allocator that is designed to be secure and effective. The authors have compared memory and time usage of SlimGuard with different state-of-the art memory management algorithms. Similar to this, we introduce lightweight allocator which performance still needs to be tested and compared with others.

As can be seen in the literature review above, state-of-the art memory management allocators are lack of either time/memory efficiency or attack protection. Only experimental methods try to approach the optimal state in both directions. In our study we have collected different ideas and proposed time and memory balanced allocator.

# Proposed Algorithm

## OS Memory management

Most of the operation systems already have memory simple allocation mechanisms. Despite that the consumed time can reach high values. Therefore, one of the ideas of memory management is to request a large chunk of memory from OS and mark the memory inside the application program.

From the security perspective, most of the vulnerabilities are based on the fact that hackers can somehow predict the address of the working object. That is why the universal protection would be the address randomization within the preallocated chunk.

## Basic allocation

As was mentioned earlier, the basic principles in marking the large chunk of the data. The memory given from OS can be divided into several parts by creating control blocks inside the allocated memory. The control block is a simple structure that holds the number of allocated bytes and the index of the previous control block. In term of C language, it can be represented as

struct control\_block  
{  
 void\* *previous*;  
 unsigned long long *size*;  
};

The *size* bytes of data should be stored right after the control block.

It can be seen that memory control blocks are united into the bidirectional list data structure. That is why the on a memory request it is possible to look through the all allocated control blocks and quickly find the fitting one.

To understand which block is used currently by the application, it is possible to encode the bit flag of the *size* field. If the *size* is divided by 2 therefore the block is free and otherwise it is in use.

## Split and Consume

Efficient memory management implies that there will be just enough bytes while allocation process. Therefore, it is natural to split big control blocks and return to the application the fitting one. The crucial part of the allocation security is to randomize the offset before the returning block (and insert another control block there if possible).

It is worth mentioning that a lot of small allocation and deallocation can lower the memory capacity almost by a half. That is why another idea would be merging two adjacent control blocks into one bigger. So, in this scheme, all allocations and deallocations would reset the state of the control block list.

# Conclusion

Different applications that implement custom allocation mechanism might lack enough secure or time/memory efficiency. In this paper we have presented common allocation approaches and implemented new memory management algorithm that is balanced in both requirements. This can protect programs vulnerable to attacks like “use after free”.

In the future we could develop a virtual laboratory for testing memory allocation mechanisms. This could help to measure the security and efficiency level of the current, existing, and future algorithms and compare them. Last but not least pro-posed algorithm could be reconstructed from multithreaded perspective in order to support a wider spectrum of applications.

##### References

1. David Dewey, Bradley Reaves, P. Traynor “Uncovering Use-After-Free Conditions in Compiled Code” // International Conference on Availability, Reliabil-ity and Security, ARES, 2015
2. Qiang Zeng, Golam Kayas, Emil Mohammed, Lannan Luo, Xiaojiang Du, Junghwan Rhee “HeapTherapy+: Efficient Handling of (Almost) All Heap Vulner-abilities Using Targeted Calling-Context Encoding” // 49th Annual IEEE/IFIP In-ternational Conference on Dependable Systems and Networks (DSN), 2019
3. Jonathan Ganz, Sean Peisert “ASLR: How Robust Is the Randomness?” // IEEE Cybersecurity Development (SecDev), 2017
4. Paul R. WilsonMark S. JohnstoneMichael NeelyDavid Boles “Dynamic Storage Allocation A Survey and Critical Review” // International Workshop on Memory Management, 1995
5. M. Masmano U, I. Ripoll, A. Crespo, J. Real “TLSF: A new dynamic memory allocator for real-time systems” // Proceedings. 16th Euromicro Confer-ence on Real-Time Systems, 2004
6. Nikola Zlatanov “Dynamic Memory Allocation and Fragmentation” // ESC, 2015
7. Mark S. Johnstone, Paul R. Wilson “The Memory Fragmentation Problem: Solved?” // ACM SIGPLAN Notices, 1998
8. Beichen Liu, Pierre Olivier, Binoy Ravindran “SlimGuard: A Secure and Memory-Efficient Heap Allocator” // Middleware '19: Proceedings of the 20th In-ternational Middleware Conference, 2019