11-2013

Defending against heap overflow by using randomization in nested virtual clusters

Chee Meng TEY
Singapore Management University, cmtey.2008@smu.edu.sg

Debin GAO
Singapore Management University, dbgao@smu.edu.sg

DOI: https://doi.org/10.1007/978-3-319-02726-5_1

Defending against heap overflow by using randomization in nested virtual clusters

Chee Meng Tey and Debin Gao

Singapore Management University, Singapore
{cmtey.2008,dbgao}@smu.edu.sg

Heap based buffer overflows are a dangerous class of vulnerability. One countermeasure is randomizing the location of heap memory blocks. Existing techniques segregate the address space into clusters, each of which is used exclusively for one block size. This approach requires a large amount of address space reservation, and results in lower location randomization for larger blocks.

In this paper, we explore the possibility of using a cluster for 2 or more block sizes. We show that a naive implementation fails because attackers can easily predict the relative location of 2 blocks with 50% probability. To overcome this problem, we design a novel allocator algorithm based on virtual clusters. When the cluster size is large, the randomization of larger blocks improves by 25% compared to existing techniques while the size of the reserved area required decreases by 37.5%.

1 Introduction

Randomization of heap memory location belongs to a larger class of anti-malware techniques collectively known as address space layout randomization (ASLR). These techniques attempt to defeat attackers by limiting their knowledge of the absolute or relative location of particular memory objects, and have gained widespread acceptance among mainstream OS [1–4]. There are also standalone allocator projects [5–7] that provide ASLR for various OS.

One of the ways in which heap memory is randomized involves the location of memory blocks returned by the `C` library function `malloc`. Existing techniques segregate the address space into clusters, each of which is divided into equally sized slots. Both the alignment and size of slots are power of 2 multiples of the minimum (typically 16 bytes). To handle a memory allocation, the allocator rounds up the requested block size to the next power of 2 multiple of the minimum, determines the cluster to use, randomly chooses an unused slot in that cluster and returns its location.

An example of the memory layout of such an allocator is shown in Figure 1. There are 2 salient features of such a method of allocation. Firstly, due to alignment restrictions, the larger the block size, the fewer the number of choices to place the block. The relative and absolute location of large blocks are therefore easier to guess. Secondly, a large area of the address space needs to be reserved.

In this paper, we study an alternative allocation algorithm where blocks of different sizes can be allocated from the same cluster by first structuring

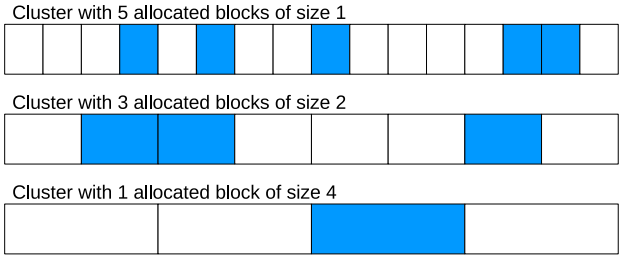


Fig. 1. A heap memory snapshot for an application using a randomizing heap allocator. Shaded blocks are allocated. In this snapshot, the application has allocated 5 blocks of size 1, 3 blocks of size 2 and 1 block of size 4. In practice, allocators do not allocate blocks of 1 byte. Instead, a minimum block size of 16 bytes is common. The block sizes in this paper can be interpreted as either bytes or multiples of the minimum.

such a cluster as a set of nested virtual clusters. We name this 'Virtual Cluster Allocation' (VCA). VCA improves both the randomness and space utilization. When the number of block sizes allocated from each cluster is increased to 2, the randomization of larger blocks improves by 25% compared to existing techniques while the size of the reserved area required decreases by 37.5%.

In the rest of this paper, we first show why a naive implementation of mixed block size allocations results in poor randomization. Next, we describe the intuition and the algorithm for VCA. We derive and prove the reserved space requirement for VCA. Finally, we describe the limitations and conclusions.

Related works Randomization is one of the major countermeasures against buffer overflow attacks. Our paper focuses on the randomization of memory blocks returned by the `libc malloc` function. Project similar in scope include the OpenBSD [8] allocator and the Diehard series of randomized allocators [5, 6]. Randomization may also involve other parts of the memory structure such as the location of stack or shared libraries [1–4, 7, 9], the instruction set encoding [10–13], and even the data [14].

2 Naive implementation

In a naive implementation, when a block needs to be allocated, the corresponding cluster is first identified. Next, all available slots meeting the alignment and size constraints are identified. A slot is then chosen randomly to fulfil the allocation request. We demonstrate in this section, through an example, the techniques that an attacker can use to place blocks to ensure a high probability of achieving a particular relative ordering. This attack requires an attacker who is able to control the heap evolution. Scenarios where this is possible include Javascript based malware [15].

For our example, there is a cluster of size 16 which is initially free. The allocator handles requests for blocks of size 1, 2 and 4. The attacker is able to

request allocation for 3 types of blocks: type A, type B and type C, each of size 1, 2 and 4 respectively. The steps used by the attacker are: (a) allocate three type C objects: C1, C2, C3, (b) allocate one type B objects: B1, and (c) allocate two type A objects: A1 and A2. Figure 2 illustrates a possible memory layout.

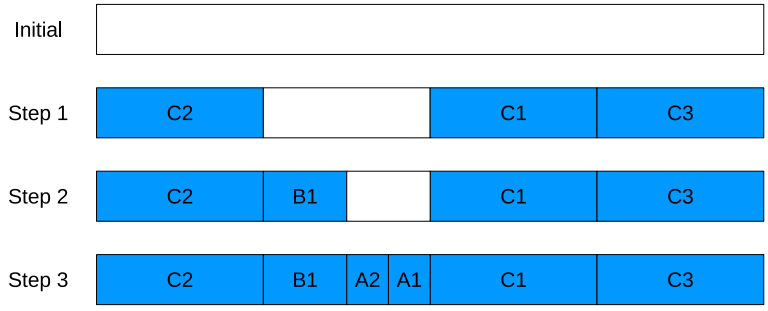


Fig. 2. Example showing how an attacker can control the sequence of allocations to ensure a high probability of achieving a desired placement order.

Fig. 3. Possible permutations in the placement of B1, A1 and A2.

In step (a), there are 4 available slots to place C1, C2 and C3, given the alignment restrictions. The allocator picks 3 of the slots randomly, leaving a single free slot of size 4. A single size 4 free slot can be divided into 2 size 2 free slots, each of which can in turn be divided into 2 size 1 free slots. To allocate a block for B1, since there are only 2 availble size 2 slots, the allocator picks 1 of the 2 slots randomly, leaving only a single slot of size 2. In step (c), the allocator places the objects A1 and A2 randomly into the remaining free area of the cluster.

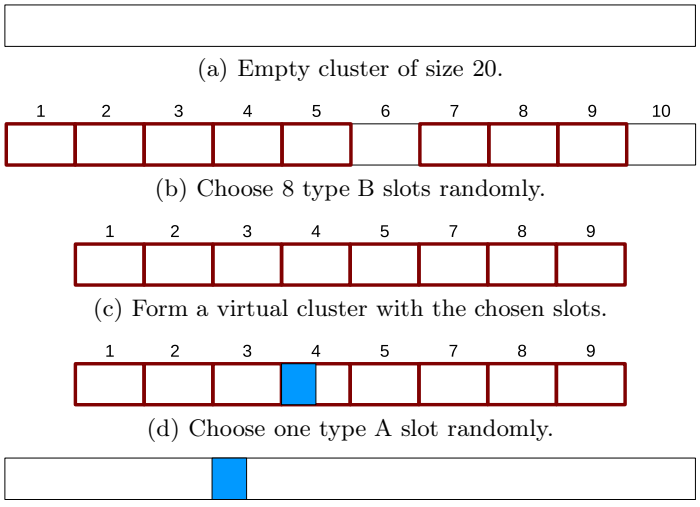
To calculate the probability that A1 will be followed by A2, we refer to Figure 3. There are 4 possible permutations, each equiprobable. In 2 of the 4 permutations, A1 is followed by A2. The probability of this occurring is therefore $2/4 = 50\%$. Similarly, in 1 of the 4 permutations, B1 is followed by A2. The probability of this occurring is therefore $1/4 = 25\%$. A naive implementation of randomization where more than 1 block size can be allocated from a single cluster therefore does not guarantee high entropy.

3 Intuition

In Figure 1, so long as the attacker's total allocation size is limited to a maximum quota of 16, all possible permutations of block locations are equally likely. This

equiprobable property holds regardless of the order of allocation and deallocation made by the attacker. Intuitively, the problem with the naive implementation of Section 2 is that by varying the order of his allocation, an attacker can violate the equiprobable property.

We illustrate our solution to this problem using a simple example which involves only type A blocks (of size 1) and type B blocks (of size 2). To allocate a type B block, the allocator picks a free type B slot randomly (similar to the naive implementation). To handle allocations of type A blocks, the allocator chooses available type B slots randomly and uses them to form a *virtual* cluster. A free type A slot is then chosen randomly from the virtual cluster. Figure 4 shows the formation of a virtual cluster (of size 16) from an empty cluster (of size 20) and the allocation of a type A block from this virtual cluster.



(a) Empty cluster of size 20.

(b) Choose 8 type B slots randomly.

(c) Form a virtual cluster with the chosen slots.

(d) Choose one type A slot randomly.

(e) Locate the corresponding slot in the parent cluster.

Fig. 4. Allocating type A blocks using a virtual cluster.

Note that a new virtual cluster is formed for every allocation. When there are prior allocations of either type A or type B blocks, dummy type B slots may be added to the virtual cluster. A cluster can therefore contain up to 5 different types of type B slots (see Figure 5):

1. Dummy slots.
2. Type B slots from which 1 type B block has been allocated. We name them type B_B slots. The number of such slots is denoted by s_B.
3. Type B slots from which 2 type A blocks has been allocated. We name them type B_2 slots. The number of such slots is denoted by s_2.

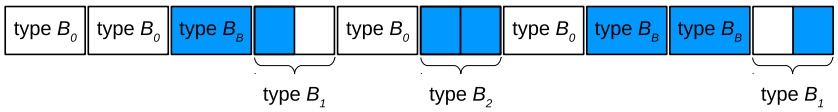


Fig. 5. Different types of type B slots. In this example, $s_0 = 4$, $s_1 = 2$, $s_2 = 1$, $s_B = 3$.

4. Type B slots from which exactly 1 type A block has been allocated. We name them type B_1 slots. The number of such slots is denoted by s_1.
5. Free (unused) type B slots. We name them type B_0 slots. The number of such slots is denoted by s_0.

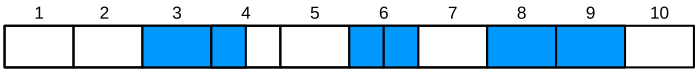
Figure 6 shows the allocation process when there are prior allocations. From the virtual cluster, one type A slot is chosen randomly from among the dummy and available slots. If a dummy slot is chosen, the selection process is repeated. However, the repeated selection can only be made from the B_0 slots in the virtual cluster and not from the B_1 slots. The rest of the process is similar to that of Figure 4.

The use of a virtual cluster within a cluster ensures that all possible permutations of block locations are equiprobable, because the randomization within each cluster is similar in principle to that of Figure 1. The random formation of the virtual cluster from the slots of the parent cluster ensures that there is no correlation between the location of slots of different sizes. The concept of nesting a virtual cluster within a parent cluster can extended to more than 2 block sizes.

4 Computation of cluster size

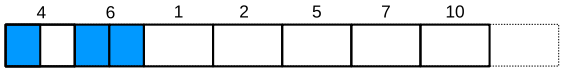
In this section, we show how the cluster size c can be chosen when the total amount of allocated memory is constrained by a quota q. The choice of c primarily involves a space-randomness tradeoff. The smaller the reserved space, the better the virtual and physical address space utilization efficiency; but it is also easier for an attacker to guess the absolute and relative location of each block. However, even if small space is desired, the reserved space cannot be as small as q. That is because fragmentation may result in available space that cannot be used. Figure 7 shows an example where c and q are 20 and 16 respectively. Even though the total allocation is only 14, this cluster can not handle any further request for type B blocks (type A blocks can still be allocated). A lower bound therefore exists for c.

For VCA, the theoretical lower bound depends on how the size of the virtual cluster, v, is chosen. Note that v must be even because the virtual cluster contains type B slots which are of size 2. For simplicity, we also assume q is even. If $v \geq 2q$, c, in theory, must be at least $2q$. Otherwise, there is at least 1 fragmentation pattern which the allocator cannot handle. If $v = q$, c must be at least $1.5q$. The reason is because, in a problem involving only 2 block sizes, the worst fragmentation occurs with B_1 blocks. If $v \geq 2q$, the worst fragmentation occurs

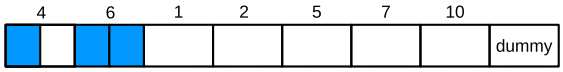


(a) A cluster of size 20 with prior allocations.

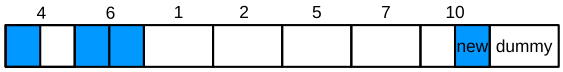
(b) Include all B_1 and B_2 slots in the virtual cluster. B_2 slots cannot be used for allocation. However, their inclusion affects the selection probabilities.



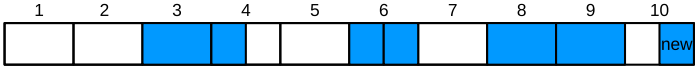
(c) Include all B_0 slots. If there are more slots than available space, a subset of B_0 slots is chosen randomly for inclusion.



(d) If there are more space than B_0 slots, dummy slots are added.



(e) Choose a new type A slot randomly from the free slots and dummy slots.



(f) Locate the corresponding slot in the parent cluster.

Fig. 6. Allocating type A blocks when there are prior allocations.

when the cluster consists entirely of B_1 blocks, resulting in a cluster size of $2q$. If $v = q$, the worst fragmentation occurs when the cluster contains a mix of B_1 and B_B blocks. Let a be the number of type A blocks that has been allocated. Let b be the number of type B blocks that has been allocated. Figure 8 shows the maximum number of B_1, B_2 and B_B slots that can be created as a varies. The maximum of $1.5q$ occurs when $a = 0.5q$ and $b = 0.25q$.

When q is large, some fragmentation patterns are so rare that they are virtually impossible. This suggests that a probabilistic bound for c exists and can be lower than $1.5q$. In such a case, the tradeoff involves not just space and randomness, but also the chance of failure. If c is chosen carefully, the chance of failure may be low enough that it is inconsequential. In the remainder of this section, we show that this intuition is correct.

Let s be the total number of type B_1 and B_2 slots (that is, $s = s_1 + s_2$). For ease of implementation, the procedure in Figure 6 can be simplified as follows:

1. Compute the ratio $r_{a,s} = (2s - a)/(v - a)$.

Fig. 7. Fragmentation reduces usable space. For ease of analysis, VCA has a constraint that adjacent free type A slots in neighbouring type B slots (such as A1 and A2) cannot be merged and used for allocating type B blocks.

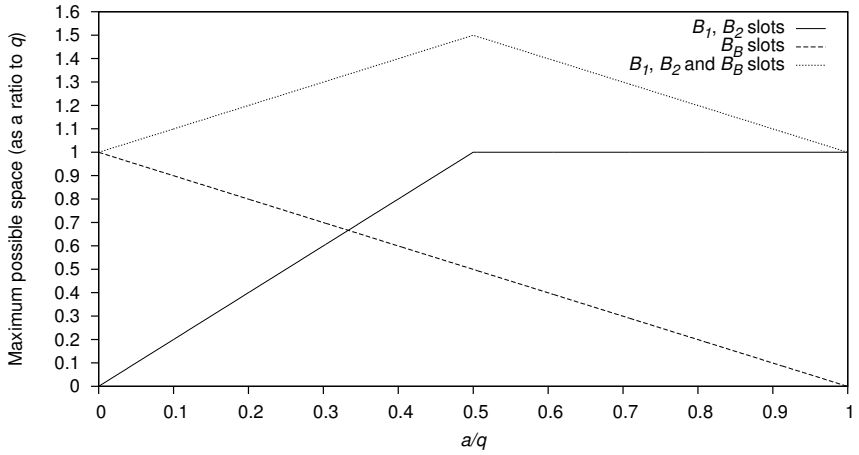


Fig. 8. Maximum space usage due to fragmentation when $v = q$.

2. Compute a random number r' between 0 and 1.
3. If $r' < r_{a,s}$ choose a type A slot from among the B_1 slots randomly.
4. If $r' >= r_{a_s}$ choose a B_0 slot from the parent cluster randomly and choose one of the two available type A slots (within the chosen B_0 slot) randomly.

We now prove some allocator properties. Let $p_{a,s}$ be the probability that when a type A blocks has been allocated, the total number of type B_1 and B_2 slots is s. Figure 9 shows an example of the distribution of $p_{a,s}$ when $v = q = 10$. Note that $p_{a,s}$ does not depend on b, the number of type B blocks allocated.

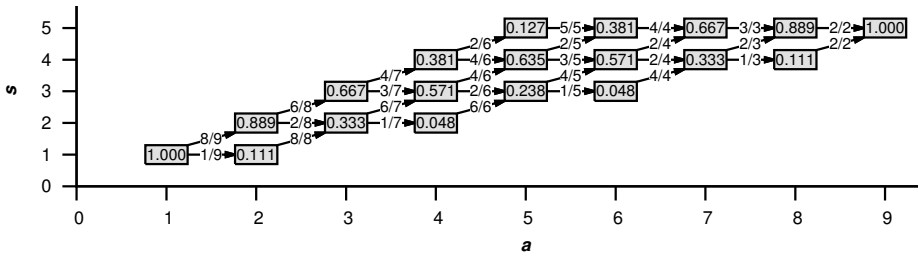


Fig. 9. The value of $p_{a,s}$ (in shaded boxes) as a function of s and a when $v = q = 10$.

In Figure 9, the labels on the horizontal arrows equal $r_{a,s}$ and indicate the probability that s remains unchanged when a is incremented by 1 (because a B_1 slot was used, resulting in a B_2 slot). The labels on the diagonal arrows equal $1 - r_{a,s}$ and indicate the probability that s increases by 1 when a is incremented by 1 (because a B_0 slot was used, resulting in a B_1 slot).

Let the most likely value of s be denoted by s_{max}. We will show that the probability density function of s is unimodal and the further s is from s_{max}, the lower the probabililty. Next, we show that the larger the quota q, the larger the probability $\Pr[s_{max}(1-\epsilon) < s < s_{max}(1+\epsilon)]$ where ϵ is a small constant. In other words, the larger q is, the less likely s differs significantly from s_{max}. Figure 10 illustrates this property. The lower probabilistic bound of c is then given by the solution of an optimization problem.

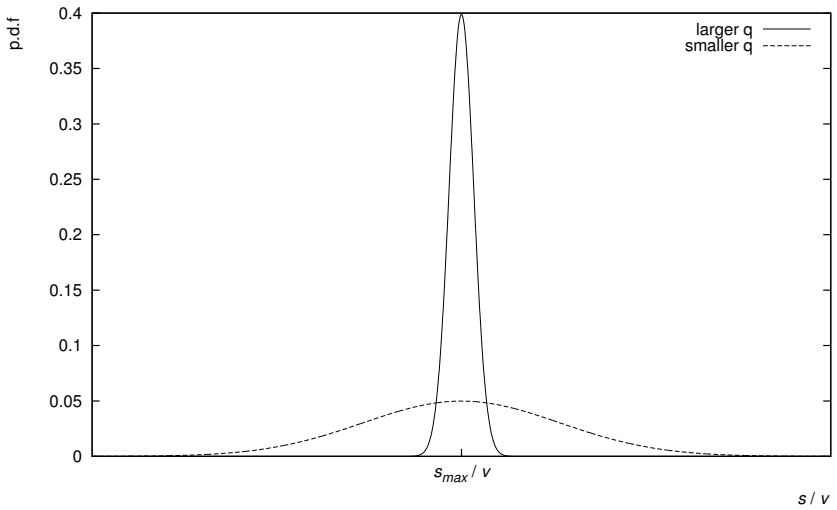


Fig. 10. The larger q is, the more likely s is close to s_{max}. Note that unlike a normal distribution, the probability density function of s is asymmetrical about the mode. The asymmetry generally diminishes as q becomes larger.

4.1 Computation of s_{max}

Theorem 1. *If $p_{a,s} \neq 0$ and $p_{a,s+1} \neq 0$, then the ratio between them is given by:*

$$h_{a,s} = \frac{p_{a,s+1}}{p_{a,s}} = \frac{2(v-2s)(a-s)}{(2s-a+1)(2s-a+2)} \tag{1}$$

Proof. The proof is by induction. The base case can be shown to be true for $p_{2,1}$ and $p_{2,2}$. In the inductive step, it can be shown that Equation 1 holds for $p_{a+1,s}$ and $p_{a+1,s+1}$ whenever any of the following conditions are true:

1. $p_{a,s-1} = 0, p_{a,s} \neq 0, p_{a,s+1} \neq 0$ and Equation 1 holds for $p_{a,s}$ and $p_{a,s+1}$.
2. $p_{a,s-1} \neq 0, p_{a,s} \neq 0, p_{a,s+1} = 0$ and Equation 1 holds for $p_{a,s-1}$ and $p_{a,s}$.
3. $p_{a,s-1} \neq 0, p_{a,s} \neq 0, p_{a,s+1} \neq 0$ and Equation 1 holds for $p_{a,s-1}$ and $p_{a,s}$ as well as for $p_{a,s}$ and $p_{a,s+1}$.

Conditions 1 and 2 are corner cases while condition 3 is the general case. Condition 1 occurs when a is odd, $1 < a < v - 1$ and $s = (a + 1)/2$. Condition 2 occurs when $1 < a < v/2$ and $a = s$. Due to brevity of space, we only provide the proof for the general case:

$$\begin{aligned} p_{a+1,s} &= p_{a,s-1}(1 - r_{a,s-1}) + p_{a,s} r_{a,s} \\ &= p_{a,s-1} \frac{v - 2s + 2}{v - a} + p_{a,s} \frac{2s - a}{v - a} \\ &= p_{a,s} \frac{(2s - a - 1)(2s - a)}{2(v - 2s + 2)(a - s + 1)} \frac{v - 2s + 2}{v - a} + p_{a,s} \frac{2s - a}{v - a} \\ &= p_{a,s} \frac{2s - a}{v - a} \left(\frac{2s - a - 1}{2(a - s + 1)} + 1 \right) \\ &= p_{a,s} \frac{2s - a}{v - a} \left(\frac{a + 1}{2(a - s + 1)} \right) \end{aligned} \tag{2}$$

$$\begin{aligned} p_{a+1,s+1} &= p_{a,s}(1 - r_{a,s}) + p_{a,s+1} r_{a,s+1} \\ &= p_{a,s} \frac{v - 2s}{v - a} + p_{a,s+1} \frac{2s - a + 2}{v - a} \\ &= p_{a,s} \frac{v - 2s}{v - a} + p_{a,s} \frac{2(v - 2s)(a - s)}{(2s - a + 1)(2s - a + 2)} \frac{2s - a + 2}{v - a} \\ &= p_{a,s} \frac{v - 2s}{v - a} \left(1 + \frac{2(a - s)}{2s - a + 1} \right) \\ &= p_{a,s} \frac{v - 2s}{v - a} \left(\frac{a + 1}{2s - a + 1} \right) \end{aligned} \tag{3}$$

Dividing (3) by (2) yields the desired result for $h_{a+1,s}$. □

Theorem 2. *There is exactly 1 turning point for the probability density function of s within the problem domain.*

Proof. A turning point occurs when the gradient equals 0. Since s is discrete, the turning point occurs when $p_{a,s}$ equals $p_{a,s+1}$, or equivalently, when $h_{a,s} = 1$.

From (1), we have

$$\begin{aligned}
\frac{2(v-2s)(a-s)}{(2s-a+1)(2s-a+2)} &= 1 \\
2(v-2s)(a-s) &= (2s-a+1)(2s-a+2) \\
2(va-vs-2sa+2s^2) &= 4s^2-2sa+4s-2sa+a^2-2a+2s-a+2 \\
2va-2vs &= 6s+a^2-3a+2 \\
-6s-2vs &= -2va+a^2-3a+2 \\
s &= \frac{-2va+a^2-3a+2}{-6-2v} \\
&= \frac{2a-\frac{a^2}{v}+3\frac{a}{v}-\frac{2}{v}}{\frac{6}{v}+2} \qquad (4)
\end{aligned}$$

Theorem 3. *The turning point for the probability density function of s is a maximum turning point.*

Proof. We only need to show that $h_{a,s+1} < h_{a,s}$ for all s. Since $h_{a,s_{max}} = 1$, $h_{a,s+1} < h_{a,s}$ implies that when $s < s_{max}$, $h_{a,s} > 1$. Similarly, when $s > s_{max}$, $h_{a,s} < 1$. In other words, the gradient is postive before the turning point and negative after the turning point, implying a maximum turning point. From 1:

$$\begin{aligned}
h_{a,s+1} &= \frac{2(v-2s-2)(a-s-1)}{(2s-a+3)(2s-a+4)} \\
&< \frac{2(v-2s)(a-s)}{(2s-a+3)(2s-a+4)} \\
&< \frac{2(v-2s)(a-s)}{(2s-a+1)(2s-a+2)} \\
&= h_{a,s}\square
\end{aligned}$$

The probability density function therefore has the shape of Figure 10.

Theorem 4. *When q is large,*

$$s_{max} = a - \frac{a^2}{2v} \qquad (5)$$

Proof. When q is large, v is also large. (5) is obtained from (4) by eliminating the insignificant terms when v is large. □

4.2 Lower bound for c

The lower bound for the cluster size c depends on the worst case value of $s_1+s_2+s_B$ for all possible attacker allocation[1] strategies. We have shown that for large q

[1] We need only consider the set of allocation only strategies because due to VCA's equiprobable property (see Section 3), each strategy that involves allocation and deallocation can be mapped to an equivalent strategy involving only allocation.

(and therefore v), s_{max} is a good approximation for s, which equals $s_1 + s_2$. Also, since one B_B slot is created whenever one type B block is allocated, $b = s_B$. The lower bound of c therefore corresponds to the upper bound of the worse case value for $s_{max} + b$. For a 2 block size problem, this can be obtained from the following optimization problem:

> Determine a, the number of type A blocks, and b, the number of type B blocks, so as to maximize $s_{max}+b$, subject to the constraints:
> 1. $a \geq 0$
> 2. $b \geq 0$
> 3. $a + b \leq q$

The solution to this problem is:

$$a = \frac{v}{2} \tag{6}$$

$$b = \frac{2q - v}{4} \tag{7}$$

Substituting (6) and (7) into (5), the lower probabilistic bound for c is given by:

$$\begin{aligned} c_{min} &= s_{max} + s_B \\ &= a - \frac{a^2}{2v} + b \\ &= \frac{v}{2} - \frac{(\frac{v}{2})^2}{2v} + \frac{2q - v}{4} \\ &= \frac{v}{2} - \frac{v}{8} + \frac{2q - v}{4} \\ &= \frac{3v}{8} + \frac{2q - v}{4} \end{aligned} \tag{8}$$

If v is chosen to be the minimum possible (i.e. q), then (8) simplifies to:

$$c_{min} = \frac{3q}{8} + \frac{2q - q}{4} = \frac{5q}{8} \tag{9}$$

Note that (9) is expressed in units of type B slots. Each type B slot has a size of 2. So the minimum size is $5q/4$. Compared to existing techniques, which reserve 1 cluster of size q each for type A and B blocks respectively, the randomization of type B blocks improves by 25% (from q to $5q/4$) while the size of the reserved area required decreases by 37.5% (from $2q$ to $5q/4$).

It should be noted that, in practice, a probabilistic allowance ϵ is needed to ensure VCA has a low chance of failure even when the worst case allocation strategy is used (see Equations 6 and 7). In such a case, the larger ϵ is, the less likely $s > c_{min} + \epsilon$. As q becomes larger, the required allowance increases in absolute terms, but decreases relative to q. If the cluster size c is set at exactly

c_{min}, then approximately 50% of the time, s will exceed c when the worst case strategy is used.

In dynamic storage allocation parlance, the lower bound of c is also known as the worst case external fragmentation (WCEF). Robson proved that, when only 2 block sizes are involved and without randomization, the WCEF will never be better (lower) than $1.5q$ [16]. Our work in this section adds 2 interesting contributions to the analysis of WCEF. Firstly, to the best of our knowledge, we are the first to show that it is possible to have a probabilistic bound through the use of randomness. Secondly, we show that, for problems involving large q, the probabilistic bound ($1.25q$) is lower than Robson's limit ($1.5q$).

4.3 Computation of $p_{a,s}$

Generally, $p_{a,s}$ can be computed by applying the following formula recursively:

$$p_{a,s} = p_{a-1,s-1}(1 - r_{a-1,s-1}) + p_{a-1,s}r_{a-1,s} \tag{10}$$

Referring to Figure 9, it can be seen that calculating $p_{a,s}$ using this method involves summing the probabilities along all possible paths starting from $p_{1,1}$ and ending with the desired $p_{a,s}$. There are 2 observations which help in simplifying the computation. Firstly, it can be observed in Figure 9, that there exists a symmetry about the line $a = v/2 = 5$. Secondly, there is a repetitive structure such that certain common numerator and denominator terms appear in all paths. All numerator and denominator terms on the diagonal arrows are common, while the denominator terms on the horizontal arrows are common. As an example, for all paths leading to $p_{4,3}$ in Figure 9, the common denominator terms are 9, 8 and 7, while the common numerator terms are 8 and 6.

In general, the common denominator terms on the paths to $p_{a,s}$ depend only on a and v (but not s). They are:

$$v - 1, v - 2, \ldots, v - a + 1$$

On the other hand, the common numerators terms on the paths to $p_{a,s}$ depend only on s and v (but not a). They are:

$$v - 2, v - 4, \ldots, v - 2(s - 1)$$

As an example, if all common terms are removed from Figure 9, the remaining numerator terms are shown in Figure 11. The product of the remaining numerator terms along each path can be characterised using a sequence $T(x, y)$, where:

1. Each term of T is formed from the multiplication of x sub-terms
2. All sub-terms are positive integers
3. The first sub-term never exceeds y and
4. Each sub-term never exceeds the preceding sub-term by more than 1.

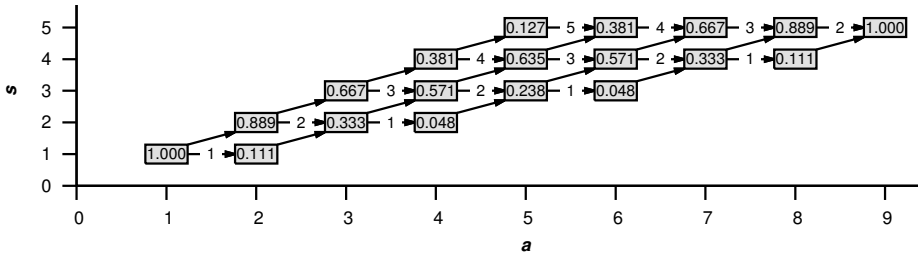


Fig. 11. Figure 9 with the common terms removed.

For example,

$$\begin{aligned}
T_{2,4} =& 1{\cdot}1,\ 1{\cdot}2,\\
& 2{\cdot}1,\ 2{\cdot}2,\ 2{\cdot}3,\\
& 3{\cdot}1,\ 3{\cdot}2,\ 3{\cdot}3,\ 3{\cdot}4,\\
& 4{\cdot}1,\ 4{\cdot}2,\ 4{\cdot}3,\ 4{\cdot}4,\ 4{\cdot}5\\
=& 1, 2, 2, 4, 6, 3, 6, 9, 12, 4, 8, 12, 16, 20\\
T_{3,2} =& 1{\cdot}1{\cdot}1,\ 1{\cdot}1{\cdot}2,\\
& 1{\cdot}2{\cdot}1,\ 1{\cdot}2{\cdot}2,\ 1{\cdot}2{\cdot}3,\\
& 2{\cdot}1{\cdot}1,\ 2{\cdot}1{\cdot}2,\\
& 2{\cdot}2{\cdot}1,\ 2{\cdot}2{\cdot}2,\ 2{\cdot}2{\cdot}3,\\
& 2{\cdot}3{\cdot}1,\ 2{\cdot}3{\cdot}2,\ 2{\cdot}3{\cdot}3,\ 2{\cdot}3{\cdot}4\\
=& 1, 2, 2, 4, 6, 2, 4, 4, 8, 12, 6, 12, 18, 24
\end{aligned}$$

These sequence are unique and not found in the OEIS database of integer sequences [17]. Note that $T_{0,y} = 1$, by definition. Let $S_{x,y}$ be the summation of all terms in $T_{x,y}$. Then $p_{a,s}$ can be computed from the product of $S_{a-s,2s-a+1}$ and the common numerators divided by the common denominators:

$$p_{a,s} = S_{a-s,2s-a+1} \left(\frac{[v-2][v-4][\ldots][v-2(s-1)]}{[v-1][v-2][\ldots][v-a+1]} \right) \tag{11}$$

4.4 Alternative method for computing $p_{a,s}$

There exists an alternative way to compute $p_{a,s}$. From Equation 2, we have a relation between $p_{a+1,s}$ and $p_{a,s}$. We can rewrite this equation as:

$$p_{a,s} = p_{a-1,s} \left(\frac{2s-a+1}{v-a+1} \right) \left(\frac{a}{2(a-s)} \right) \tag{12}$$

Similarly from Equation 3, we have a relation between $p_{a+1,s+1}$ and $p_{a,s}$. We can rewrite this equation as:

$$p_{a,s} = p_{a-1,s-1} \left(\frac{v-2s+2}{v-a+1} \right) \left(\frac{a}{2s-a} \right) \tag{13}$$

We also know that $p_{1,1} = 1$ for all v. To compute $p_{a,s}$, we can choose *any* path from $p_{1,1}$ to the desired $p_{a,s}$ and apply Equations 12 and 13. If $a = s$, there is only 1 path:

$$p_{1,1},\ p_{2,2},\ p_{3,3},\ \ldots,\ p_{s,s} \tag{14}$$

The formulas for this path are:

$$\begin{aligned}
p_{2,2} &= p_{1,1}\left(\frac{v-2}{v-1}\right)\left(\frac{2}{2}\right) \\
&= \left(\frac{v-2}{v-1}\right) \\
p_{3,3} &= p_{2,2}\left(\frac{v-4}{v-2}\right)\left(\frac{3}{3}\right) \\
&= \left(\frac{v-2}{v-1}\right)\left(\frac{v-4}{v-2}\right) \\
p_{4,4} &= p_{3,3}\left(\frac{v-6}{v-3}\right)\left(\frac{4}{4}\right) \\
&= \left(\frac{v-2}{v-1}\right)\left(\frac{v-4}{v-2}\right)\left(\frac{v-6}{v-3}\right) \\
&\ldots \\
p_{n,n} &= \left(\frac{v-2}{v-1}\right)\left(\frac{v-4}{v-2}\right)\left(\frac{v-6}{v-3}\right)\cdots\left(\frac{v-2(n-1)}{v-(n-1)}\right)
\end{aligned} \tag{15}$$

If $a > s$, for simplicity, we choose (14) for the first part of the path, then continue with

$$p_{s+1,s},\ p_{s+2,2},\ \ldots,\ p_{a,s} \tag{16}$$

We have:

$$\begin{aligned}
p_{a,s} &= p_{a-1,s}\left(\frac{2s-a+1}{v-a+1}\right)\left(\frac{a}{2(a-s)}\right) \\
p_{a-1,s} &= p_{a-2,s}\left(\frac{2s-a+2}{v-a+2}\right)\left(\frac{a-1}{2(a-s-1)}\right) \\
p_{a-2,s} &= p_{a-3,s}\left(\frac{2s-a+3}{v-a+3}\right)\left(\frac{a-2}{2(a-s-2)}\right) \\
p_{s+1,s} &= p_{s,s}\left(\frac{s}{v-s}\right)\left(\frac{s+1}{2(1)}\right)
\end{aligned} \tag{17}$$

Together, Equations 15 and 17 provide an alternative method of computing $p_{a,s}$. Interestingly, Equations 15, 17 and 11 also allow us to derive an expression for $S_{x,y}$. Substituting Equations 15 and 17 into 11 and simplifying, we get:

$$S_{a-s,2s-a+1} = \frac{(2s-a+1)(2s-a+2)(\ldots)(a)}{2^{a-s}(a-s)!} \tag{18}$$

Substituting $x = a - s$ and $y = 2s - a + 1$, we get:

$$S_{x,y} = \frac{(y)(y+1)(\ldots)(2x+y-1)}{2^x x!} \tag{19}$$

5 Limitations

Randomly allocating blocks from a large cluster results in poor spatial locality, which depending on the size of the cache and the application usage may affect cache performance. This problem affects all randomized allocators including VCA. This tradeoff however results in improved security against heap based buffer overflow attacks.

In practice, a probabilistic allowance, ϵ needs to be added to the cluster size. For a 2 block size problem, this increases the cluster size beyond the probabilistic bound of $1.25q$ (but never beyond the theoretical bound of $1.5q$). For a fixed probability of failure, the lower q is, the greater the magnitude of this allowance (relative to q).

To extend VCA to more than 2 block sizes, one way is to use multiple clusters with power of 2 block sizes. For example, on a platform with a page size of 4096 bytes, VCA would use 4 types of clusters, each handling block sizes of (i) 16 bytes and 32 bytes, (ii) 64 bytes and 128 bytes, (iii) 256 bytes and 512 bytes (iv) 1024 bytes and 2048 bytes. Requests greater than or equal to the page size can be allocated from the system directly (e.g. using the mmap system call). The rounding of allocation sizes to power of 2 may lead to wastage of storage known as internal fragmentation. This weakness is however shared by existing randomized allocators as well as the binary buddy allocator.

A second way of extending VCA to more than 2 block sizes is by nesting virtual clusters recursively. This results in a multi-variable optimization problem. It can be shown that a unique solution exists for the extended problem. The analysis of this problem is however omitted due to brevity of space.

Yet another possible way of extending VCA is to consider nesting more than 1 virtual cluster within a single parent cluster. For example, if the size of the parent cluster is a multiple of 6, then the cluster may support allocations of block size 6 directly, and allocations of block sizes 1, 2 and 3 using 3 virtual clusters. We have not analysed the feasibility of this approach and leave it as future work.

6 Conclusions

In this paper, we show that it is possible to improve the randomization while reducing the space requirement of randomized heap allocators by allocating more than 1 block size from a single cluster. With 2 block sizes, compared to existing randomized allocators, the randomization of larger blocks improves by 25% while the size of the reserved area required decreases by 37.5%.

References

1. The PaX Team: Homepage of the PaX Team http://pax.grsecurity.net.
2. Android community: Android security overview http://source.android.com/tech/security/index.html.
3. Otto Moerbeek: A new malloc(3) for OpenBSD http://www.openbsd.org/papers/eurobsdcon2009/otto-malloc.pdf.
4. Ollie Whitehouse: An Analysis of Address Space Layout Randomization on Windows VistaâĎć http://www.symantec.com/avcenter/reference/Address_Space_Layout_Randomization.pdf.
5. Berger, E.D., Zorn, B.G.: Diehard: probabilistic memory safety for unsafe languages. In: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation. PLDI '06, New York, NY, USA, ACM (2006) 158–168
6. Novark, G., Berger, E.D.: Dieharder: securing the heap. In: Proceedings of the 17th ACM conference on Computer and communications security. CCS '10, New York, NY, USA, ACM (2010) 573–584
7. Li, L., Just, J.E., Sekar, R.: Address-space randomization for windows systems. In: Proceedings of the 22nd Annual Computer Security Applications Conference. (2006) 329–338
8. OpenBSD: The OpenBSD project http://www.openbsd.org.
9. Bhatkar, S., DuVarney, D.C., Sekar, R.: Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In: Proceedings of the 12th USENIX security symposium. Volume 120., Washington, DC. (2003)
10. Barrantes, E.G., Ackley, D.H., Palmer, T.S., Stefanovic, D., Zovi, D.D.: Randomized instruction set emulation to disrupt binary code injection attacks. In: Proceedings of the 10th ACM conference on Computer and communications security, ACM (2003) 281–289
11. Barrantes, E.G., Ackley, D.H., Forrest, S., Stefanović, D.: Randomized instruction set emulation. ACM Transactions on Information and System Security (TISSEC) **8**(1) (2005) 3–40
12. Boyd, S.W., Kc, G.S., Locasto, M.E., Keromytis, A.D., Prevelakis, V.: On the general applicability of instruction-set randomization. Dependable and Secure Computing, IEEE Transactions on **7**(3) (2010) 255–270
13. Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering code-injection attacks with instruction-set randomization. In: Proceedings of the 10th ACM conference on Computer and communications security, ACM (2003) 272–280
14. Cadar, C., Akritidis, P., Costa, M., Martin, J.P., Castro, M.: Data randomization. Technical report, Microsoft Research (2008) Technical Report MSR-TR-2008-120.
15. Daniel, M., Honoroff, J., Miller, C.: Engineering heap overflow exploits with javascript. In: Proceedings of the 2nd conference on USENIX Workshop on offensive technologies. WOOT'08, Berkeley, CA, USA, USENIX Association (2008) 1:1–1:6
16. Robson, J.M.: An estimate of the store size necessary for dynamic storage allocation. J. ACM **18**(3) (July 1971) 416–423
17. OEIS: The On-Line Encyclopedia of Integer Sequences (Aug 2013) http://oeis.org/.