metadata

1. memory allocation

1.1. metadata-free allocation

allocation without metadata being written to the data memory pool except for some static data structures or pointers.

1.2. in-place metadata allocation

allocation where sensitive allocation metadata is placed in the same memory pool where allocated data resides.

1.2.1 reverse order metadata allocation

allocation where the allocator calculates the position of the metadata token and places the metadata at the end of this region, returning a pointer to the start of the region, that is why the currently allocated region being written by the user program is placed within the allocated region.

1.2.2 direct order metadata allocation

the same as above, except that metadata, is written first, protecting the current metadata block from casual overwrites. it is slightly better in terms of safety but will also have the same caveats in case of buffer underflow, allowing corruption of control data.

1.2 detached metadata allocation

allocation which completely separates data allocation pools as sensitive allocation metadata for manipulation allocations. this is the safest approach to the allocation of data.

it is similar to pp.1.1 except that pp.1.1 completely omits metadata semantics with exception of fixed-size generic types or static data structures.

1.2.1 queued allocation

allocation based on additional memory pools. idea is to keep track list of items to de-allocate and take pointers from that list. this is the first implementation allowing non-linear allocation/de-allocation on a primary memory pool.

2. pointer metadata

it is also probably would be very helpful to return additional metadata and keep track of the size of the region addressed by the pointer.

2.1 language support for metadata

it is questionable why the language authors did not include something like pointer metadata info in C language itself, causing massive blown of several classes of errors list pointer arithmetic, buffer overflow, and buffer underflow.

```
struct ptr_metadata {
    void* ptr;
    u32 size;
}
```