Simple Analysis of Software Virtualization of Memory in Unicorn Engine

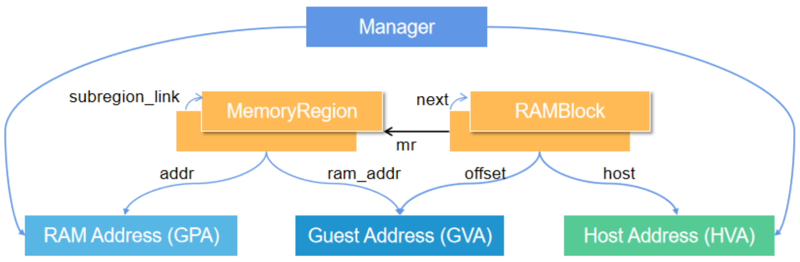
liyansong included in Virtualization Vulnerability Analysis

2022-04-14 1009 words 5 minutes views

Unicorn Engine is cut from QEMU, it can be said to be a branch of QEMU, but it is a completely independent project. Unicorn provides a wealth of APIs, allowing us to easily experience many features brought by QEMU virtualization technology, but it also faces some security issues. We take the analysis of CVE-2022-29694 as an example to briefly outline the principle of Unicorn memory virtualization. Only some simple analysis is made here, so that everyone can quickly grasp the whole picture of QEMU virtualization.

# Important Structure

We first introduce two important structures in QEMU, namely `MemoryRegion` and `RAMBlock`. You can also look at the specific code in the following chapters, and then look back at the role of each domain member in them to achieve a better understanding.



Structure of memory manager

MemoryRegion is an important structure in QEMU memory management, and it is a structure that manages each memory space of GVA(Guest Virtual Address).

> C

RAM Block

RAM Block represents a memory stick in the virtual machine, namely **GPA(Guest Physical Address)**. All RAMBlocks will be connected to the linked list through the next field, and the linked list header is `ram_list.blocks` .

C

```c
struct RAMBlock {
    struct MemoryRegion *mr;          // GPA->GVA
    uint8_t *host;                    // GPA->HVA
    ram_addr_t offset;                // The offset addre
    ram_addr_t used_length;           // used length
    ram_addr_t max_length;
    uint32_t flags;
    /* RCU-enabled, writes protected by the ramlist loc
    QLIST_ENTRY(RAMBlock) next;
    size_t page_size;                 // system page size
};
```

# Entry to Memory

`uc_mem_map` is an interface provided by Unicorn to users. We can use this function to apply for a memory space for the simulator. This space can not only store the instructions to be simulated, but also be used as the stack of the virtual machine.

C

```c
    /* init unicorn */
    UC_INIT(uc);

    /* only defined in MIPS */
    if (uc->mem_redirect) {
        address = uc->mem_redirect(address);
    }

    /* For memory safety checks, check the incoming ad
    res = mem_map_check(uc, address, size, perms);
    if (res) {
        return res;
    }

    /* mem_map mapped_blocks
     * memory_map GVA
     */
    return mem_map(uc, address, size, perms,
                   uc->memory_map(uc, address, size, p
}
```

Apply for Memory Region

Continue to analyze `memory_map`, this function is defined in `qemu/softmmu/memory.c` file, used to formally apply for Memory Region. As can be seen from the function `memory_region_add_subregion`, Unicorn uses uc->system_memory as a parent set to manage all Regions.

( )

Memory region in QEMU

Code

C

```c
    MemoryRegion *ram = g_new(MemoryRegion, 1);

    /* RAM allocation */
    memory_region_init_ram(uc, ram, size, perms);
    if (ram->addr == -1) {
        // out of memory
        return NULL;
    }

    /* system_memory->subregion = ram
     */
    memory_region_add_subregion(uc->system_memory, beg:

    if (uc->cpu) {
        tlb_flush(uc->cpu);
    }

    return ram;
}
```

`memory_region_init_ram` is an important function for the virtual machine to request RAM.

C

```c
void memory_region_init_ram(struct uc_struct *uc,
                            MemoryRegion *mr,
                            uint64_t size,
                            uint32_t perms)
{
    /* init MR */
    memory_region_init(uc, mr, size);
    mr->ram = true;
    if (!(perms & UC_PROT_WRITE)) {
        mr->readonly = true;
    }
    mr->perms = perms;
    mr->terminates = true;
    mr->destructor = memory_region_destructor_ram;
    /* important! apply for GPA and HVA */
    mr->ram_block = qemu_ram_alloc(uc, size, mr);
}
```

The function `qemu_ram_alloc` is the encapsulation of `qemu_ram_alloc_from_ptr` , which is used to apply for RAMBlock.

RAMBlock initialization

C

```c
RAMBlock *qemu_ram_alloc_from_ptr(struct uc_struct *uc,
                                   MemoryRegion *mr)
{
    RAMBlock *new_block;
    ram_addr_t max_size = size;

    size = HOST_PAGE_ALIGN(uc, size);
    max_size = HOST_PAGE_ALIGN(uc, max_size);
    new_block = g_malloc0(sizeof(*new_block));
    if (new_block == NULL)
        return NULL;
    new_block->mr = mr;
    new_block->used_length = size;
    new_block->max_length = max_size;
    assert(max_size >= size);
    new_block->page_size = uc->qemu_real_host_page_size
    new_block->host = host;
    if (host) {
        new_block->flags |= RAM_PREALLOC;
    }
    ram_block_add(mr->uc, new_block);

    return new_block;
}
```

The function `ram_block_add` is used to add a memory stick to the system. This function first applies for PVA(Physical Virutal Address), and then adds RAMBlock to the system space.

CVE-2022-29694

The QEMU virtual machine applies for memory, and finally needs to apply for the corresponding memory on the host. Then the problem arises. When a virtual machine applies for a large memory, the actual application will inevitably fail. The following function does not return the status of the application result, and the current RAMBlock has not been added to `ram_list.blocks` .

Alloc memory in GPA->HVA

The upper layer function does not judge whether the applied HVA is successful, but the virtual machine defaults that the applied GVA has been successful. When the virtual machine is shut down, QEMU memory free is triggered. At this time, the block is not QLIST, resulting in null pointer dereference.

Null pinter to free QEMU ram

As shown below, when the virtual machine applies for large memory, the GVA is successful, but the GPA->HVA using the `mmap` function fails. Unsynchronized allocated memory results in a null pointer.

Analysis of vulnerability

# Fixs

The first repair method is relatively simple, directly using `QLIST_SAFE_REMOVE` provided by QEMU.

Patch 1

The second repair solution seems more reasonable, because the essence of this problem is the inconsistency of the judgment of applying for GVA and GPA, so it is necessary to add consistency judgment results of them.

# Conclusion

This article is only a preliminary analysis of the process of Unicorn Engine memory virtualization, and introduces the general scheme for QEMU to apply for memory for virtual machines. QEMU memory virtualization is a complex process, and more mechanisms behind it need to be further studied in the future.

Updated on 2022-05-29

Unicorn, QEMU, CVE

Back | Home

ELF Static Injection to Load Malicious Dynamic Link Library