Secure Containers with EPT Isolation

Chunyan Liu
liuchunyan9@huawei.com

Jixing Gu
jixing.gu@intel.com

Presenters

- **Jixing Gu:**

Software Architect, from Intel CIG SW Team, working on secure container solution architecture design and KVM support.

- **Chunyan Liu:**

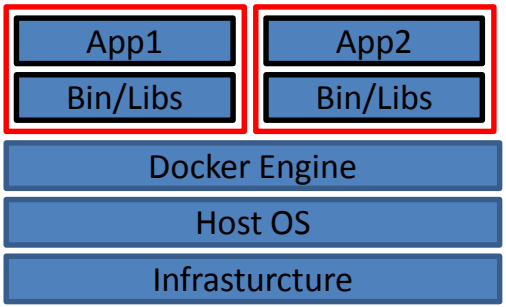
Software Engineer, from Huawei Central Software Institution, working on secure container solution design, guest OS support and Docker tooling integration.

Agenda

- Background
- Secure Container Solution
- Architecture Design
- Work Flow
- Security Cases & Demo
- Benchmark

Container Stack & Security

Containers Stack



- Ease of Development/Deployment
- High performance, low overhead
- Shared kernel
- Namespace isolation

Security?

- Attack surface is large
- Namespace isolation is weak
- Bugs in Linux kernel can allow escape to the host and harm other containers

Security Features Supported by Docker

☑ *Capability*: restrict capabilities of process in container
☑ *Seccomp*: filter access to syscall
☑ *SElinux*: customize privileges for processes, users and files.
☑ *User namespace*: map 'root' in container to non-root user on host
☑ *Fuse*: isolate "/proc", useful for container monitoring system.

CAN: restrict container capabilities and privileges, reduce chances container attacking kernel

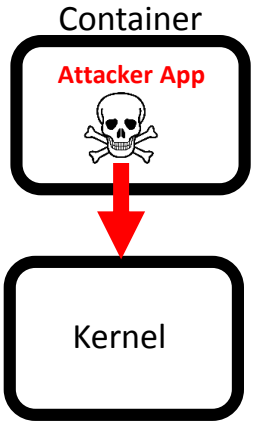
However, it's not enough……

CANNOT: container privilege escalation by exploiting kernel vulnerabilities …

Security Problems We Address

Problem-1:

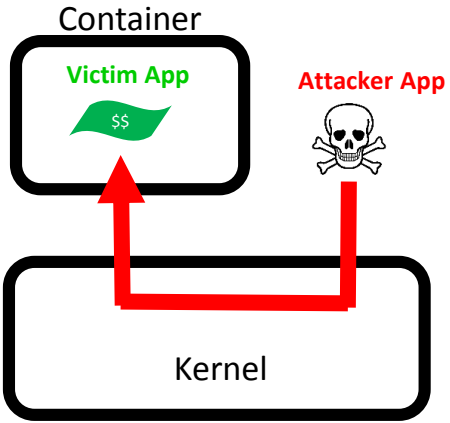
Privilege Escalation



By exploiting kernel vulnerability, like CVE-2017-6074, attackers are able to escalate to root privilege with ret2user

Problem-2:

Memory Data Peek

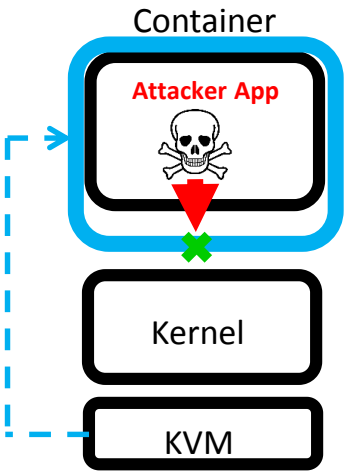


By exploiting kernel modules, like /dev/mem device, attackers w/ root privilege are able to peek containers memory data

Our Solution: Namespace-alike Memory Isolation w/ EPT

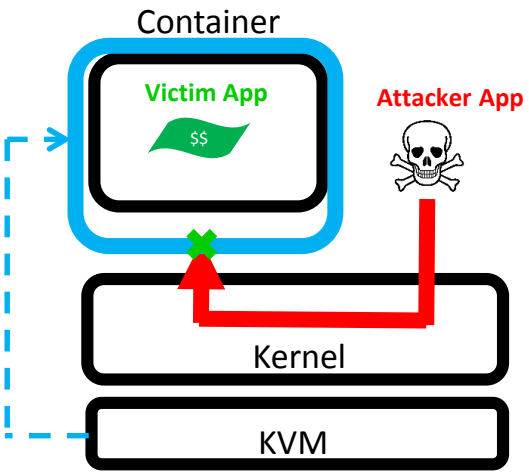
Protection-1:

Defeat Privilege Escalation



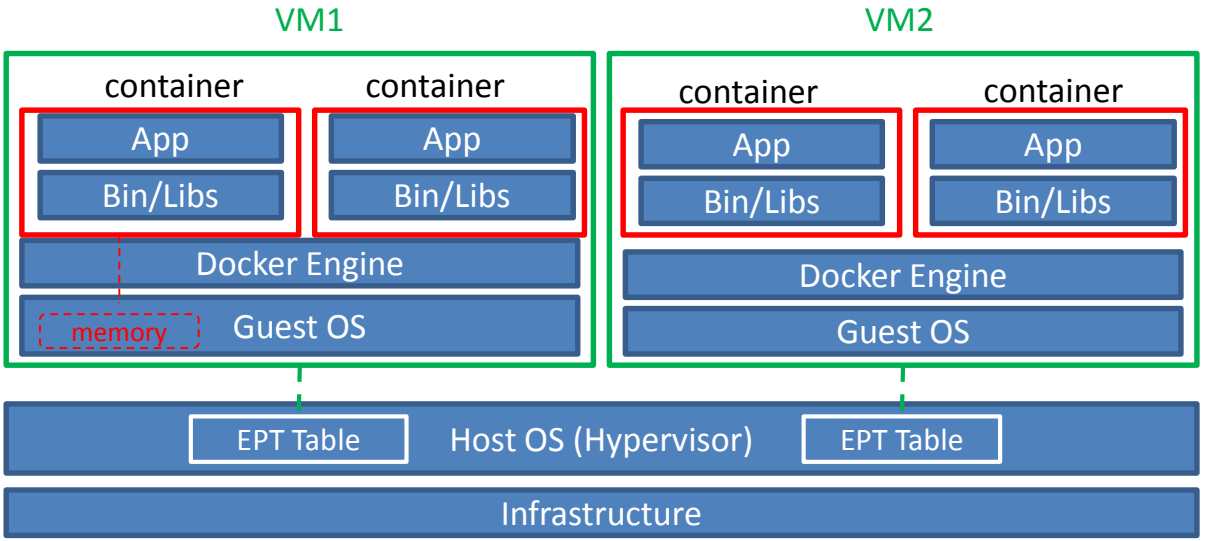
Protection-2:

Defend Memory Data Peek

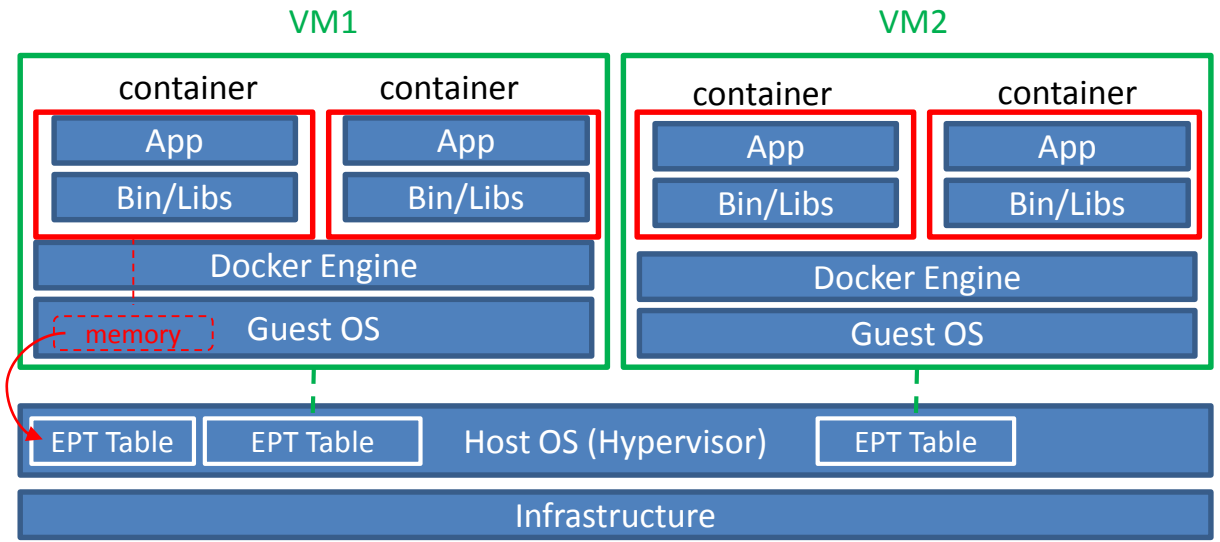


KVM creates a EPT memory region to isolate container user-space memory from kernel, and captures illegitimate cross-EPT code execution or data access behaviors with EPT violation

Typical User Model of Containers

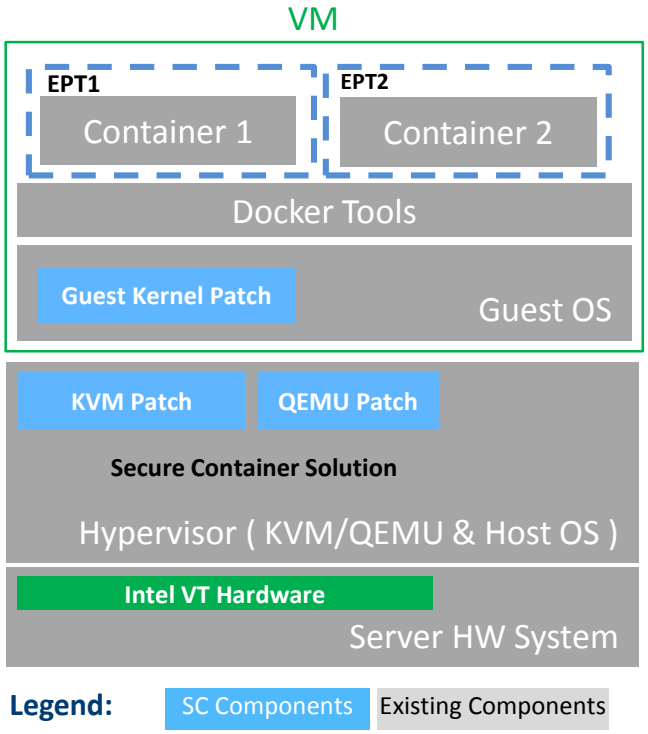


Our Ideas To Secure Containers



- Extract container memory to a new EPT table, separated from VM EPT Table
- Strong EPT isolation between container and VM kernel.

Secure Container SW Stack Up



Secure Container Solution Includes:

- **Secure Container KVM Patch**
  - ☑ Extend KVM existing interfaces for secure container
  - ☑ Create EPT for new container
  - ☑ Add mem page into EPT view
  - ☑ Delete mem page from EPT view
- **Secure Container Guest Kernel Patch**
  - ☑ Handle secure container creation w/ extended interfaces from the underlying KVM
  - ☑ Handle data exchange w/ extended interfaces from KVM
- **Secure Container QEMU Patch**
  - ☑ Support VM management interfaces
- **Tiny Changes To Docker Tools**
  - ☑ Differentiate Secure Container from Common Container

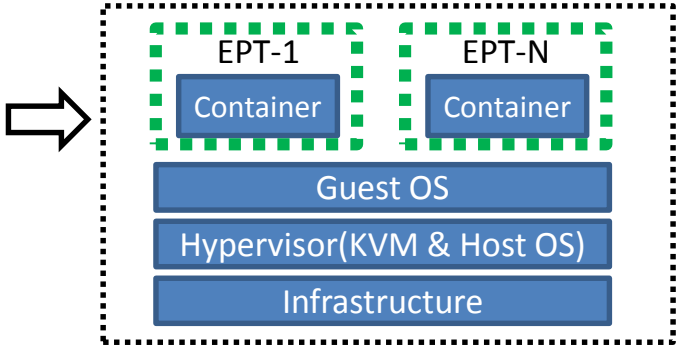
Typical Usage Scenarios

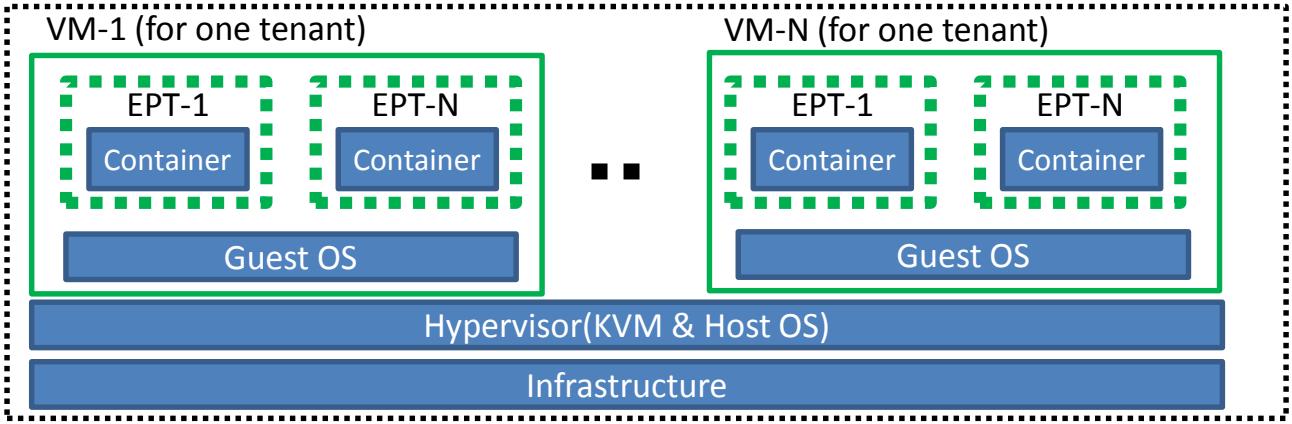
Private Cloud:

Isolate containers in a single shared guest OS

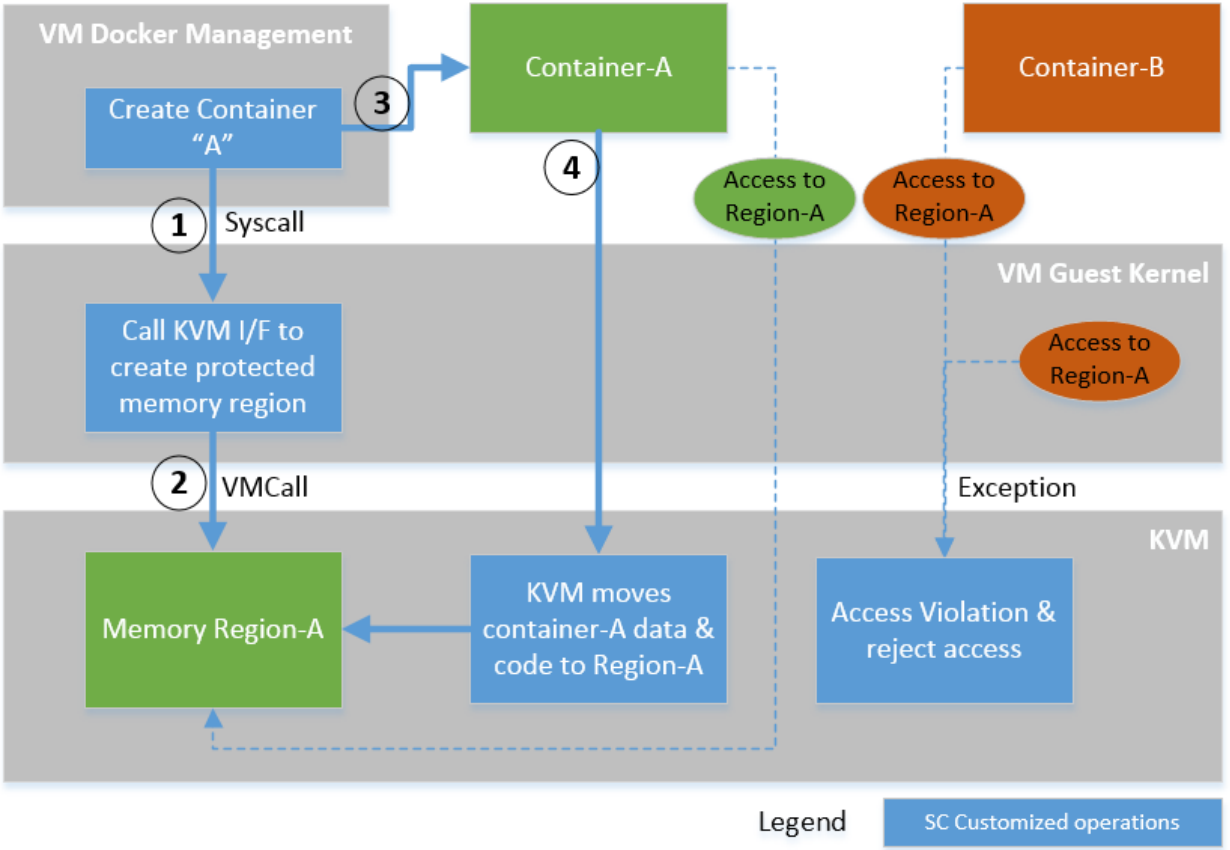
Public Cloud:

Isolate containers from same tenant & leverage VM to isolate between tenants

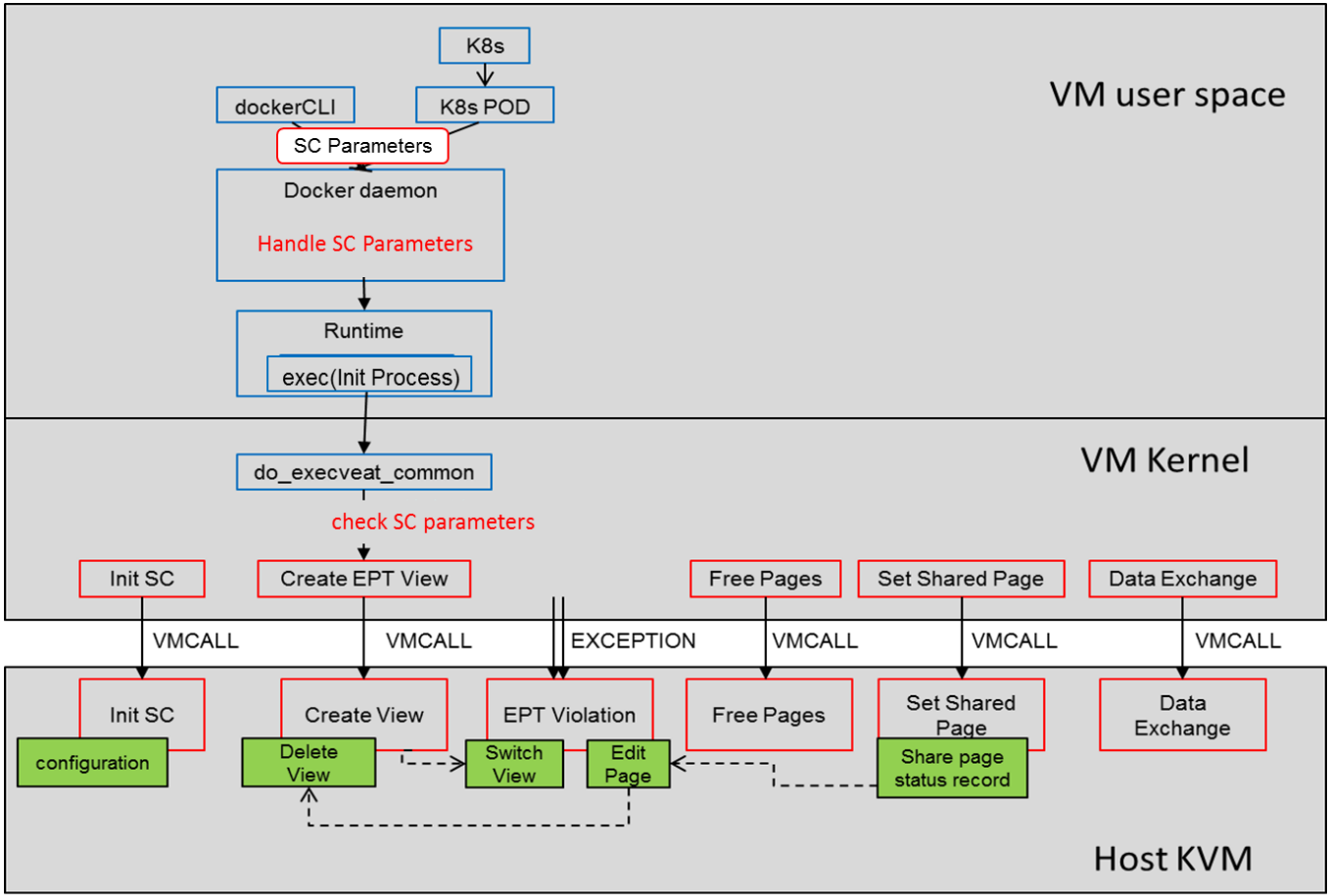




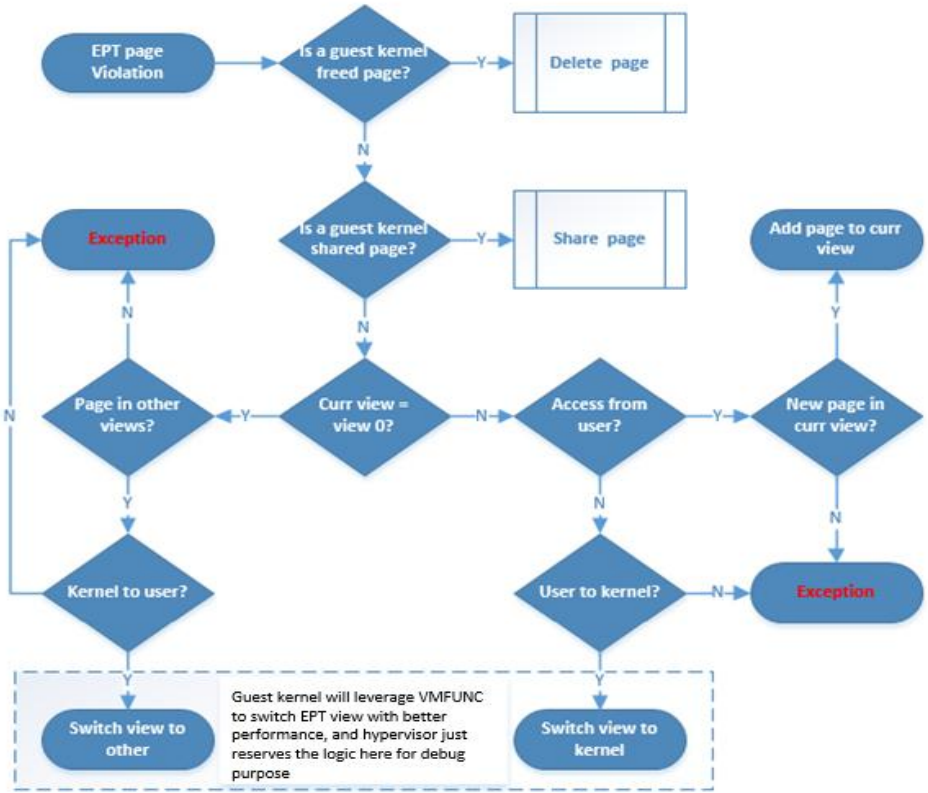
Secure Container Arch and Data Flow



Secure Container Interfaces Calling Flow



EPT Violation Flow – Switch View & Add Page



Kernel to User Condition

- Violation happens under view 0
- VM's CPL = 3
- RIP from user
- GVA from user space
- Page is present in specific user view

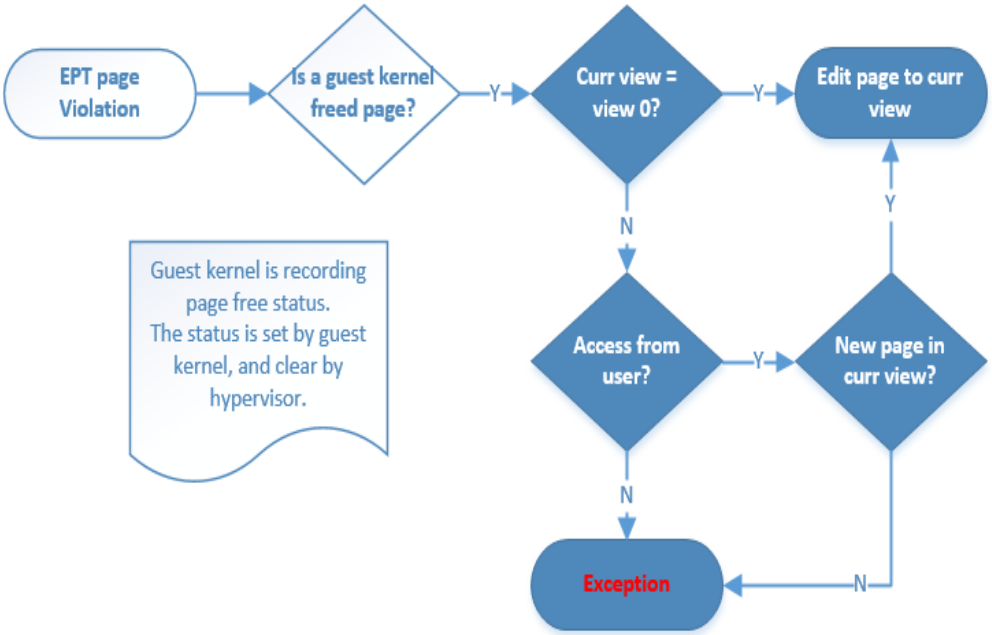
User to Kernel Condition

- Violation happens under user view
- VM's CPL = 0
- RIP from kernel space
- GVA from kernel space

New page in current user view Condition

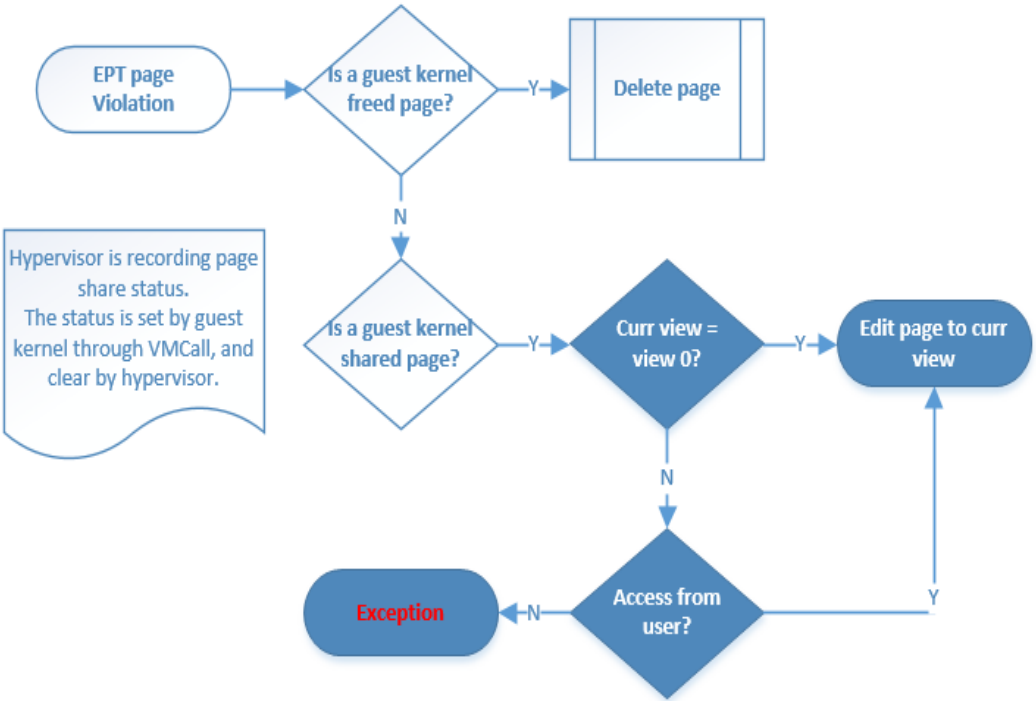
- Requested page is not used by other user views
- Requested page GVA is in user space

EPT Violation Flow – Delete Page



- Delete Page is to delete all access permission for a specific page from an EPT view.
- It entails a page free from guest kernel.
- EPT violation handles a delayed page deletion in hypervisor.
- Hypervisor will erase all the data before deleting a page.

EPT Violation Flow – Share Page



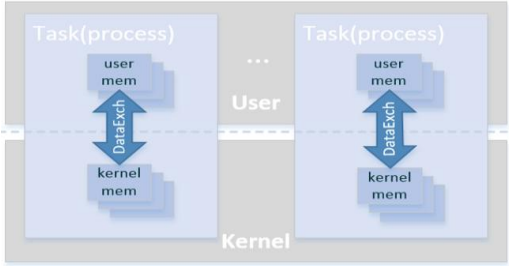
- Pages in file cache may be shared across containers.
- Hypervisor records share page status.
- Shared pages shall be granted to access by different containers.
- Share pages should be created per EPT view.

Data Exchange between user/kernel space

Virtual Function	int DataExchange(struct data_ex_cfg cfg, uint64_t size)	
Hyper Call ID – rax	KVM_HC_SC	
Hyper Call Param 1 – rbx	HC_DATA_EXCHANGE = 4	

Hyper Call Param 2 – rcx	struct data_ex_cfg cfg	struct data_ex_cfg {
		data_exchg_type op;
		union {
		struct {
		uint64_t *mov_src;
		uint64_t *mov_dst;
		uint64_t mov_size;
		}
		struct {
		uint32_t *ptr1;
		uint32_t *ptr2;
		}
		}
		}

Hyper Call Param 3 – rdx	sizeof(struct data_ex_cfg)	
Hyper Call Return Value – rax	0 – success; -1 – fail;	



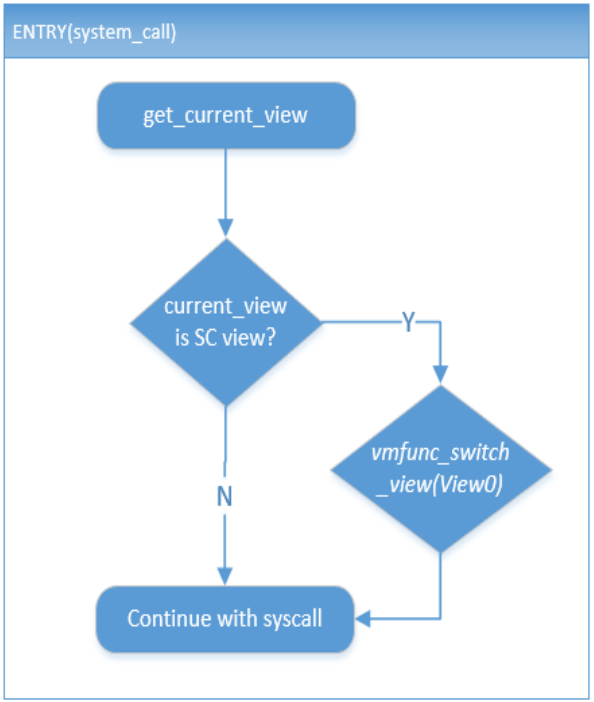
Pseudo Code

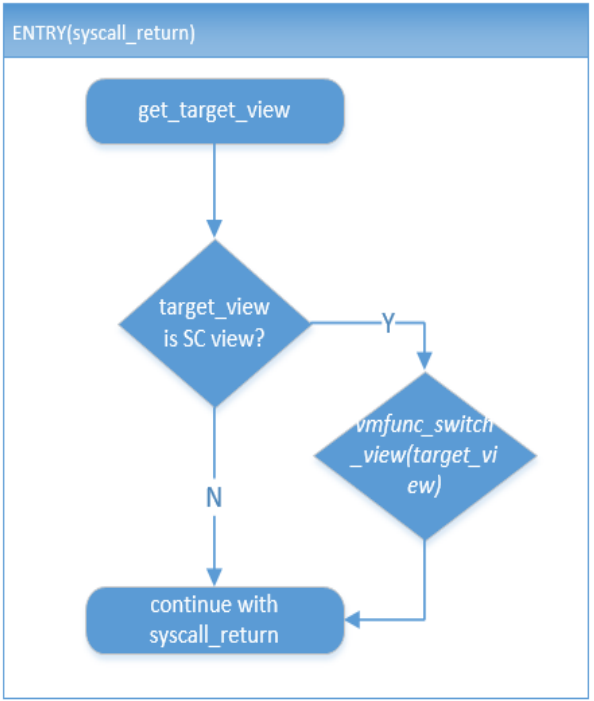
```
struct task_struct {
……
bool sc_flag; // if true, indicate the task is in a secure container.
……
};

int data_exchange(struct data_ex_cfg *cfg, uint64_t size)
{
   if (current->sc_flag == false) { return 1; }
   ……
//Transfer struct data_ex_cfg to hypervisor through hyper call.
HyperCall( KVM_HC_SC, HC_DATA_EXCHANGE, cfg,
                sizeof(struct data_ex_cfg) );
……
}
```

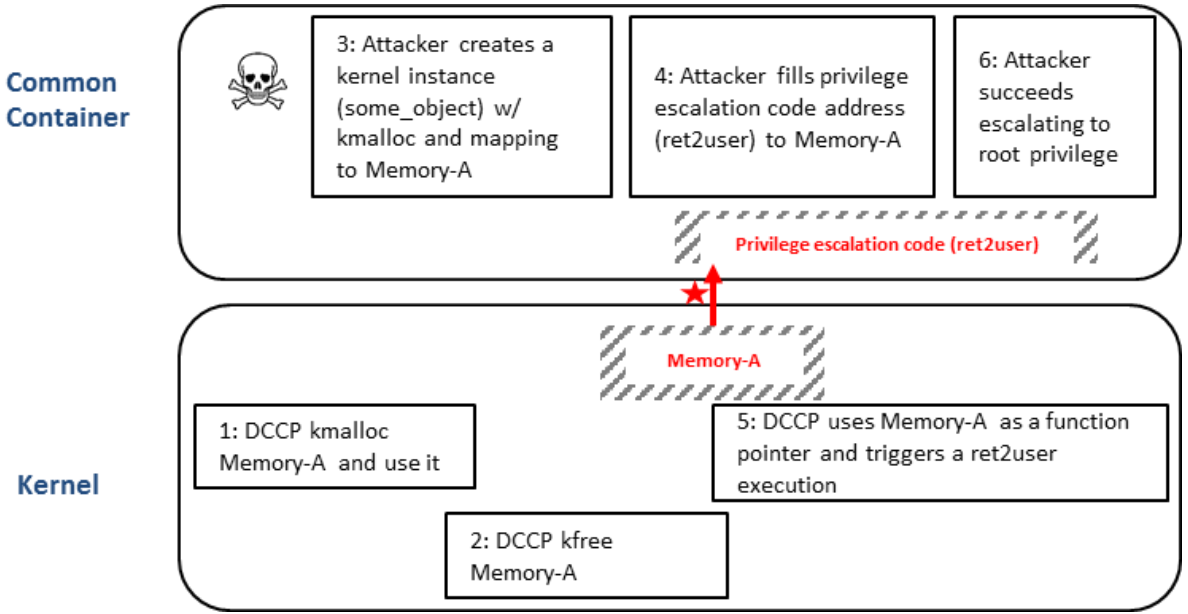
Optimization: VMFUNC Switch EPT View

- Use VMFUNC in places that need to switch view (e.g. syscall)





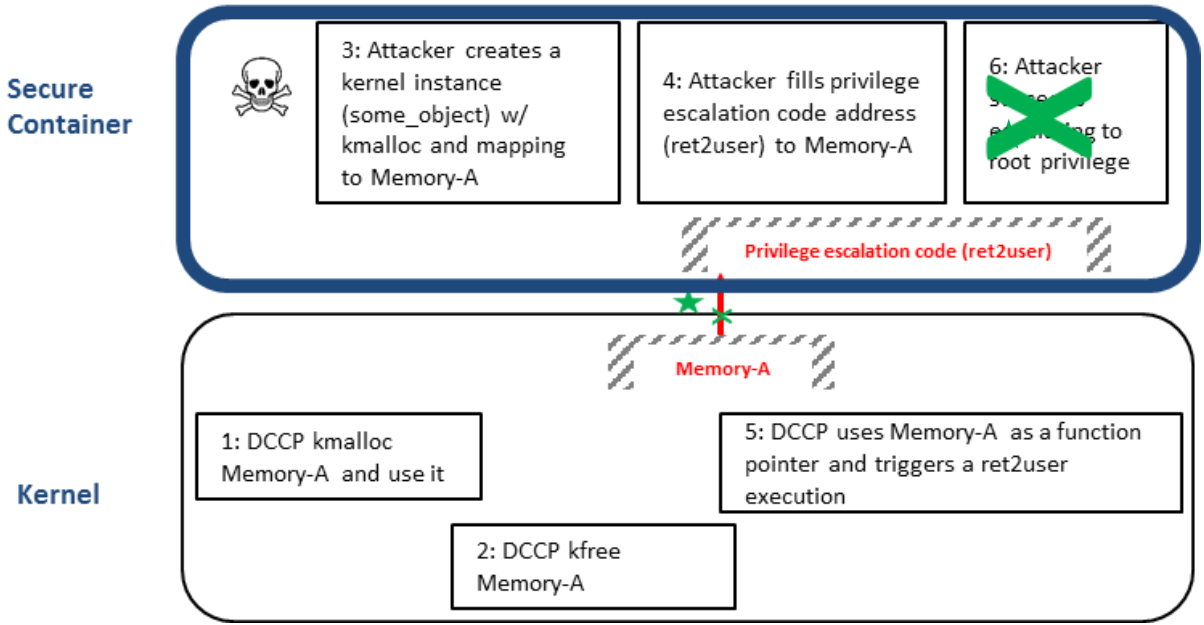
Security Case: CVE-2017-6074 ret2user Attack



Note: ★

1. Kernel trigger user space execution code;
2. CPL (current privilege level) is ring 0;
3. Privilege escalation succeeds;

Security Case: CVE-2017-6074 ret2user Attack



Note:

1. EPT violation happens while kernel trying to execute user space code because kernel space and user space are isolated in different EPT views;
2. KVM checks CPL (current privilege level) is ring 0, and concludes it is not a reasonable system call return and rejects the operation. (CPL should be ring 3 for normal syscall_return);
3. Even SMAP and SMEP were both disabled, secure container can prevent the privilege escalation;

Demo Time

Benchmark

- Current data:

	SC vs CC Overhead	Notes
Boot time	555ms vs 481 ms (~15%)	
Memory footprint	≈	Host memory has little overhead to manage more EPT tables
CPU Performance	≈	
Memory Performance	≈	
Storage IO	18%~26%	Optimization focus (VMFUNC result not updated)
Network IO	28%~30%	Optimization focus (VMFUNC result not updated)

- Next Step:

1. Use VMFUNC to reduce switch view VMEXIT/VMENTRY overhead
2. Improve DataExchange performance by reducing VMCALL times

Thank you!