Breaking Turtles All the Way Down

An Exploit Chain to Break Out of VMware ESXi

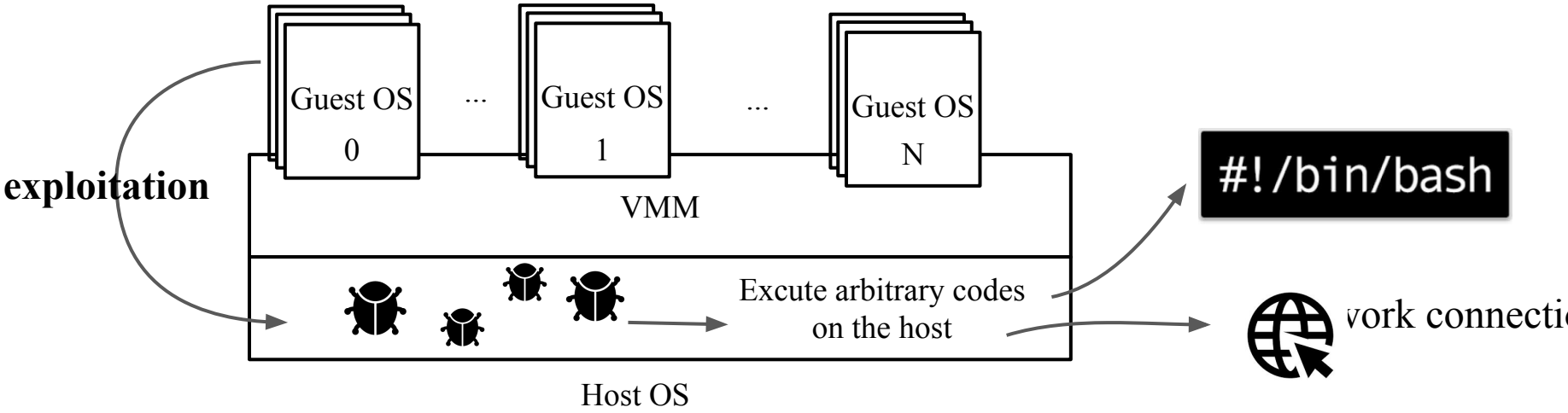
Hanqing Zhao, Yanyu Zhang, *Kun Yang, Taesoo Kim*

Chaitin Security Research Lab
Georgia Institute of Technology
Tsinghua University

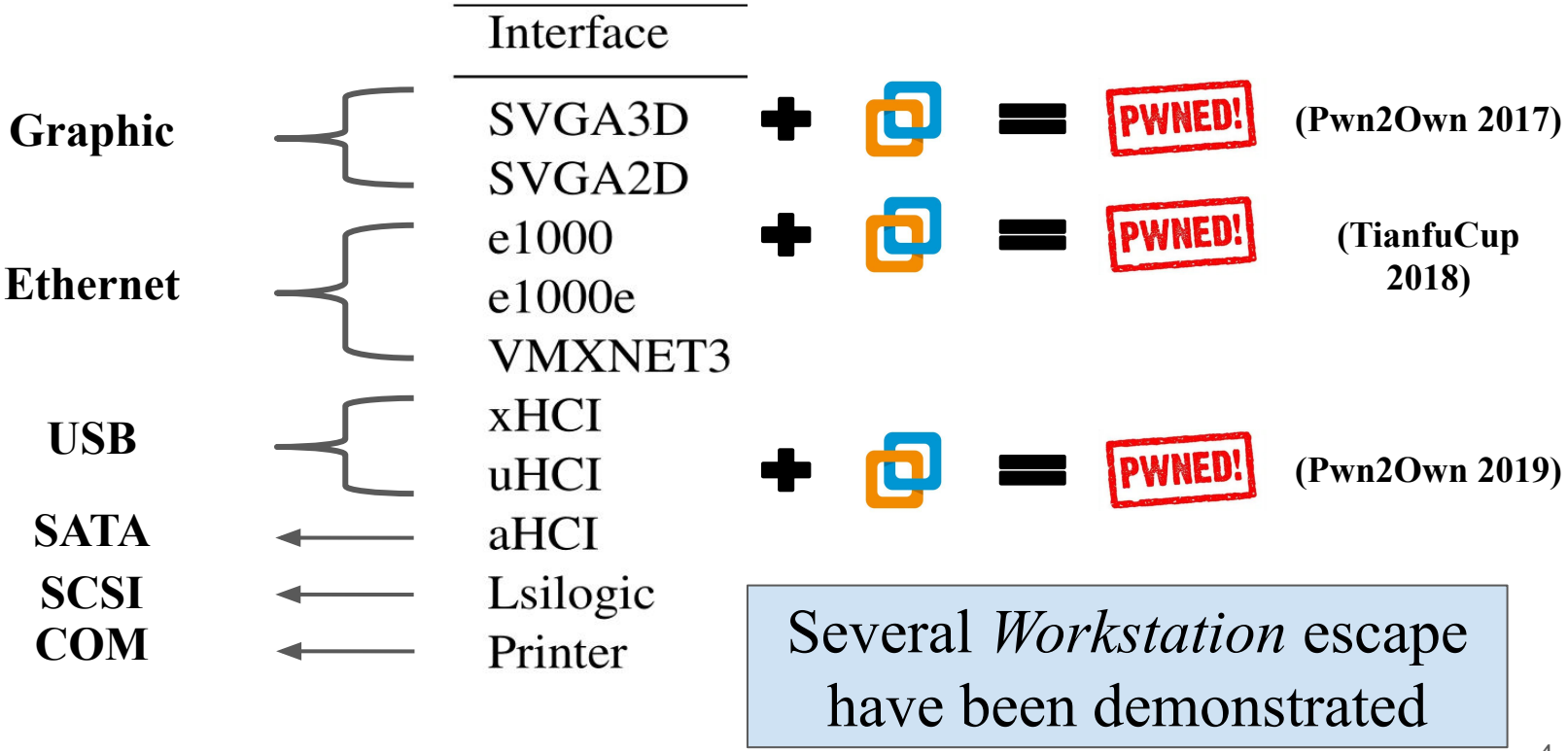
What is ESXi?

- Bare-metal hypervisor
- Widely used in private cloud

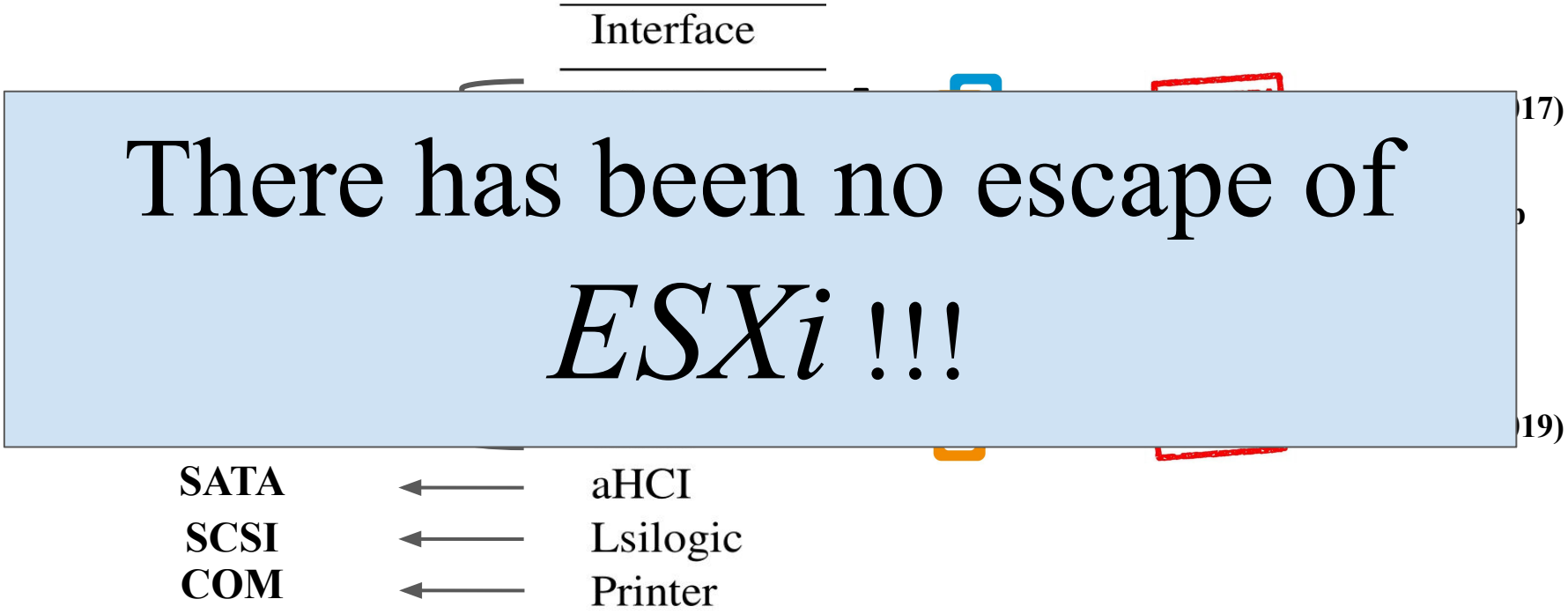
What is VM Escape?



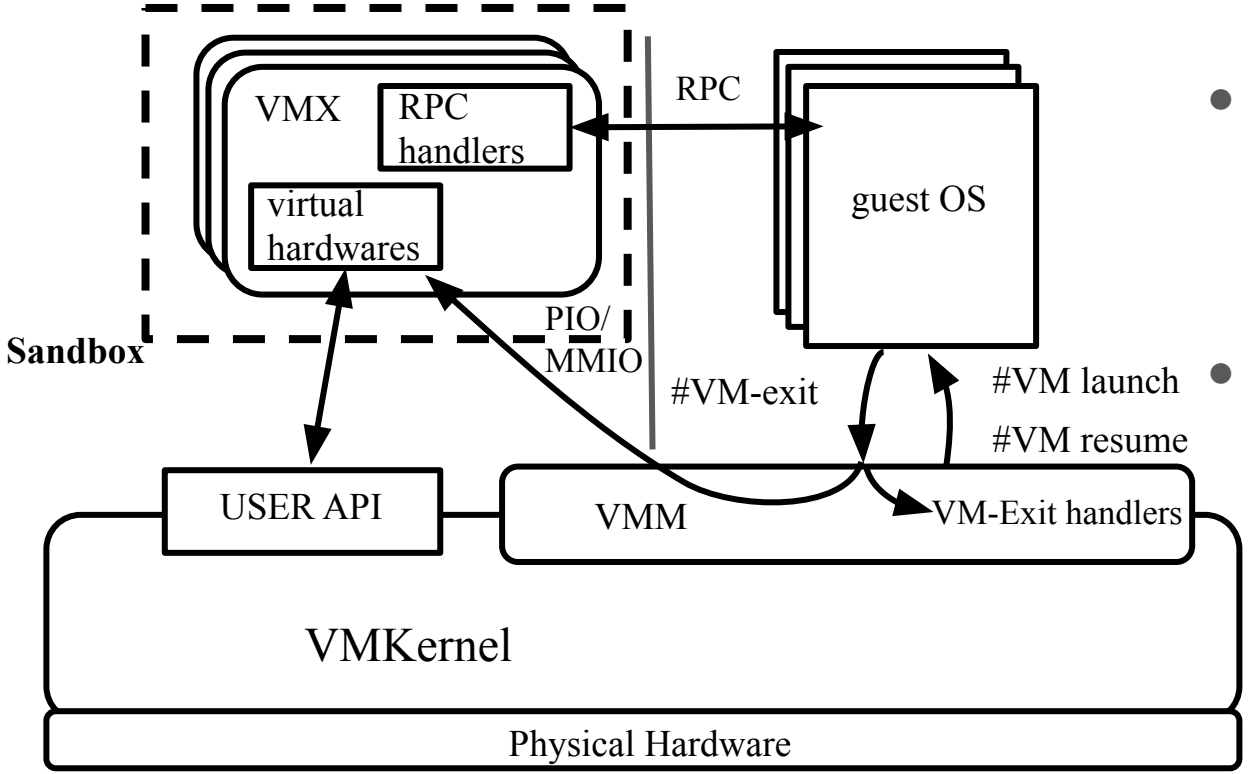
Attack I/O devices



No ESXi Escape? What makes it so challenging?

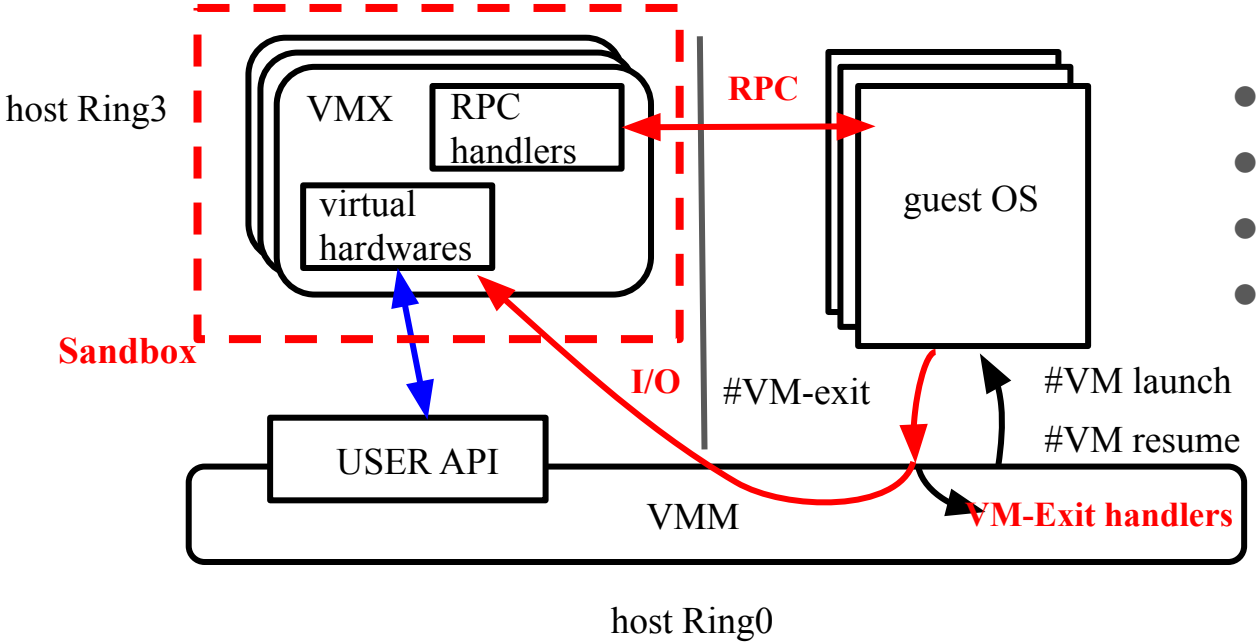


The customized architecture has not been studied



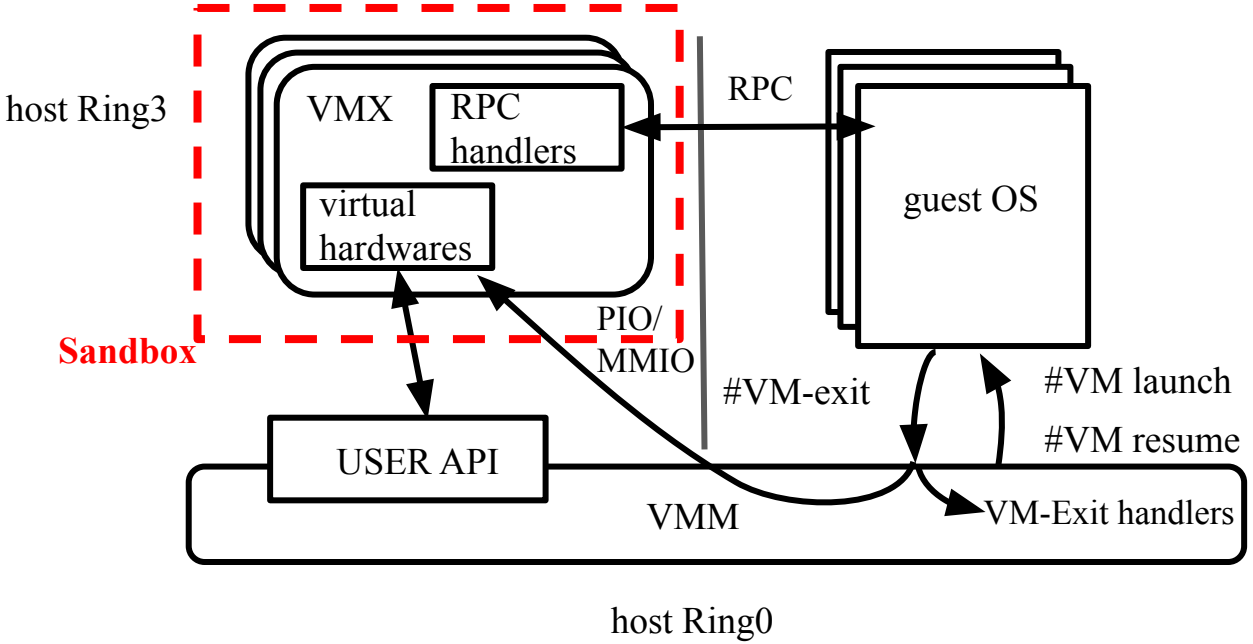
- VMKernel
  - VMM(hypervisor)
  - User world API
  - VMFS
  - Drivers
- VMX Process
  - Virtual hardwares
  - RPC handlers

Limited attack surfaces



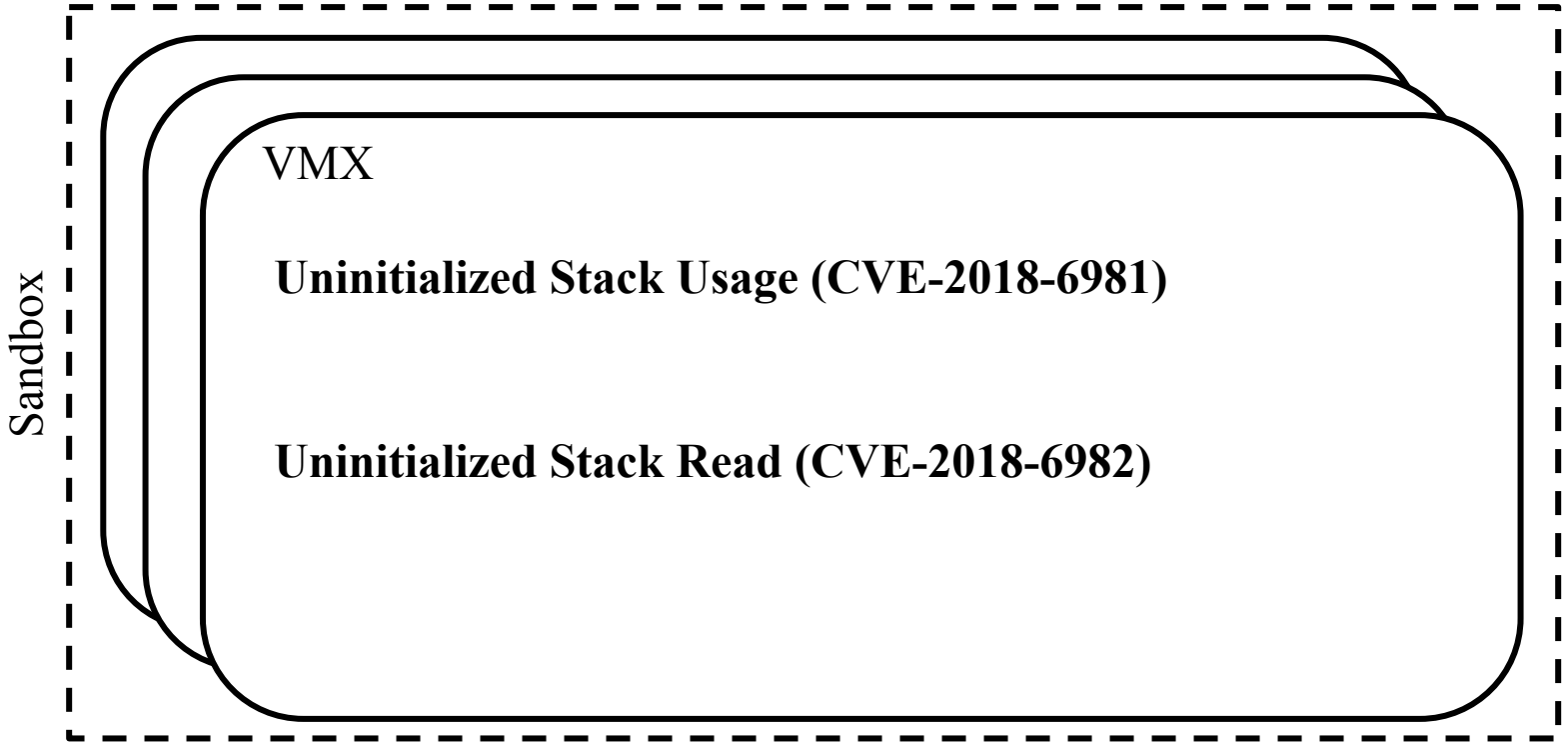
- VM-Exit handlers
- Virtual Hardwares
- RPC
- compact VMX process

Challenging mitigations and protections

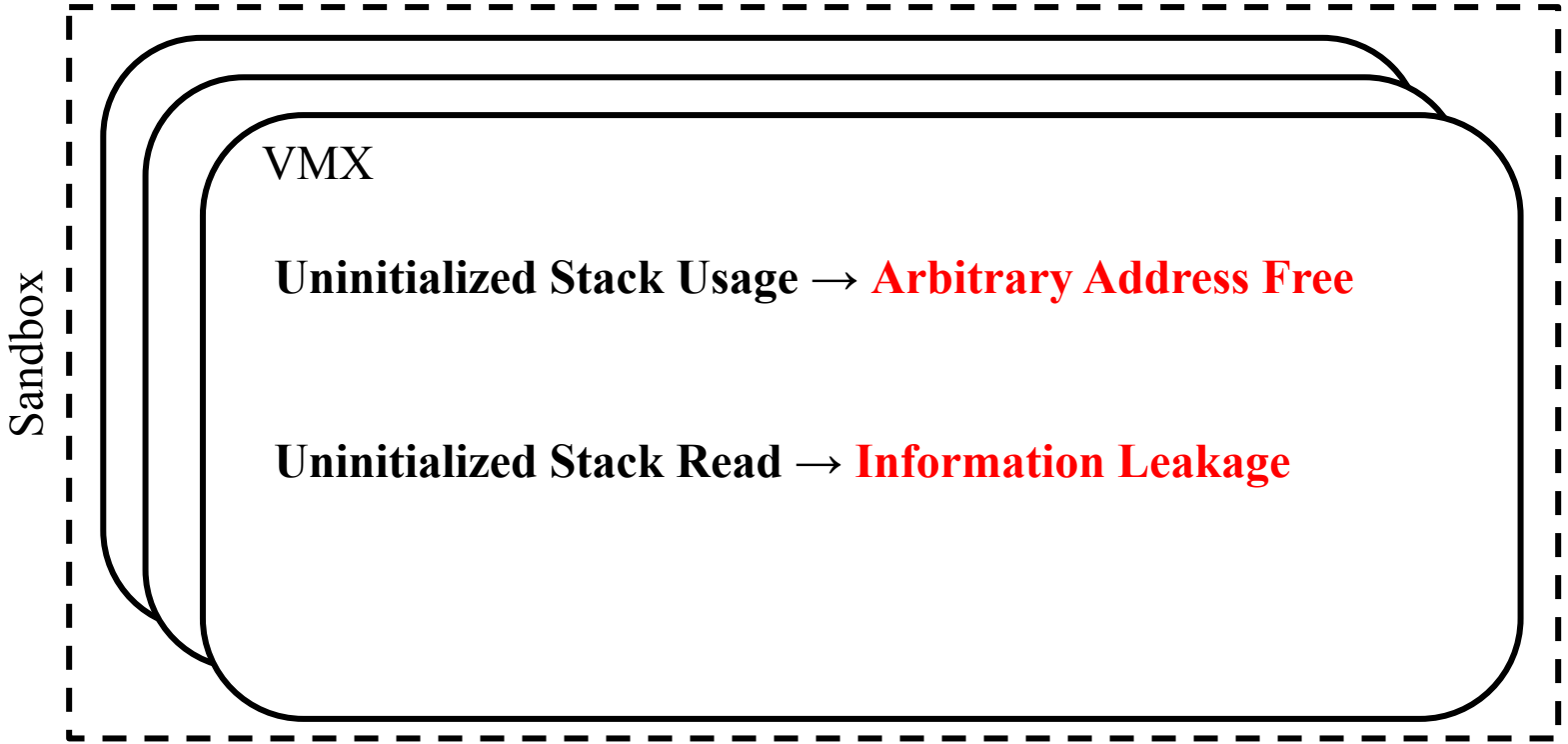


- ASLR
- NX/DEP
- sandboxing

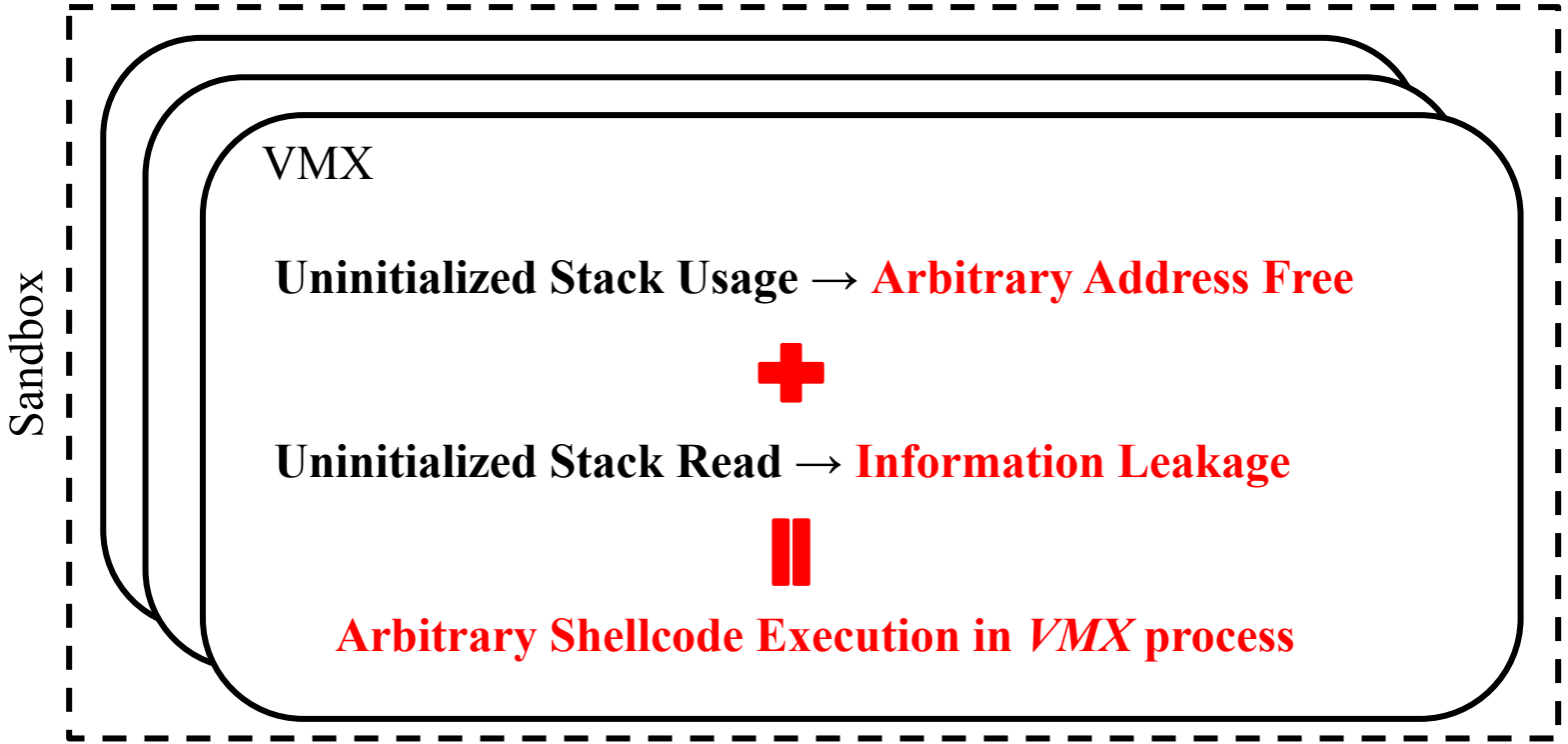
Overview: Vulnerabilitles and Exploitation



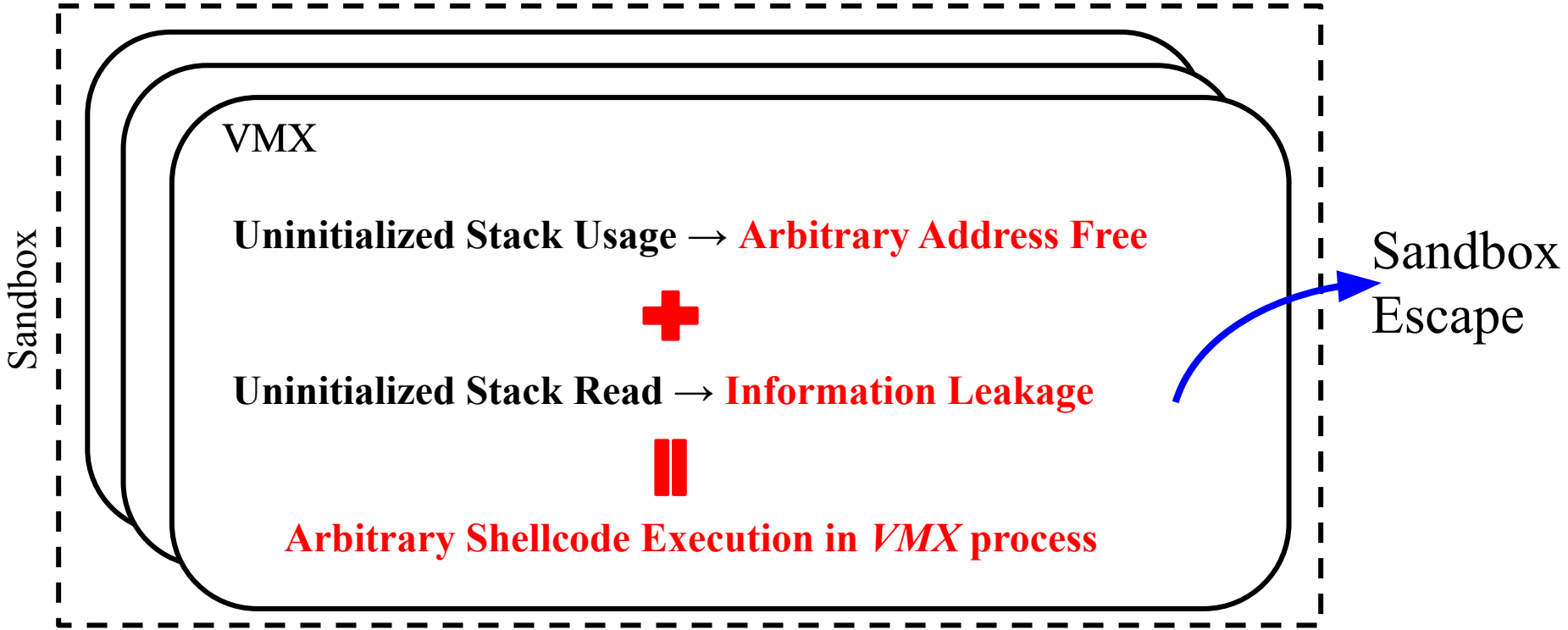
Overview: Vulnerabilitles and Exploitation



Overview: Vulnerabilitles and Exploitation



Overview: Vulnerabilitles and Exploitation



CVE-2018-6981: Uninitialized Stack Variable

```
void __usercall vmxnet3_reg_cmd() {
  ...
  case 4: // VMXNET3_CMD_UPDATE_MAC_FILTERS
*     dma_memory_create(..., &page);
      vmxnet3_cmd_update_mac_filters(v6, &page, a5);
*     destruct_page_struct(&page);
  }
    break;
  ...
}
char __fastcall dma_memory_create(unsigned addr, uint64
size, ..., page_struct *page)
{
  // check the addr
*  if ( addr > v5 || !size || size > v5 - addr + 1 )
*    return 0;
  set_page_struct(addr, size, a3, a4, page);
  return 1;
}
```

```
void destruct_page_struct(page_struct *a1)
{
   if(page_count == 1)
      free(a1->addr)
   // free the pointer on stack
   else
      free(a1->page_array);
}
```

page

translate_size	
page_offset	
page_count	
addr	0x10
page_array	0x18
...	

a struct on stack for address translation

CVE-2018-6981: Uninitialized Stack Variable

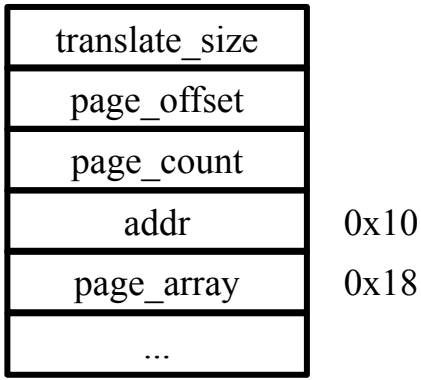
```
void __usercall vmxnet3_reg_cmd() {
  ...
  case 4: // VMXNET3_CMD_UPDATE_MAC_FILTERS
*     dma_memory_create(..., &page);
      vmxnet3_cmd_update_mac_filters(v6, &page, a5);
*     destruct_page_struct(&page);
  }
    break;
  ...
}
```

try to initialize the "page" structure on stack

```
char __fastcall dma_memory_create(unsigned addr, uint64
size, ..., page_struct *page)
{
  // check the addr
*  if ( addr > v5 || !size || size > v5 - addr + 1 )
*    return 0;
  set_page_struct(addr, size, a3, a4, page);
  return 1;
}
```

```
void destruct_page_struct(page_struct *a1)
{
   if(page_count == 1)
      free(a1->addr)
   // free the pointer on stack
   else
      free(a1->page_array);
}
```

page



a struct on stack for address translation

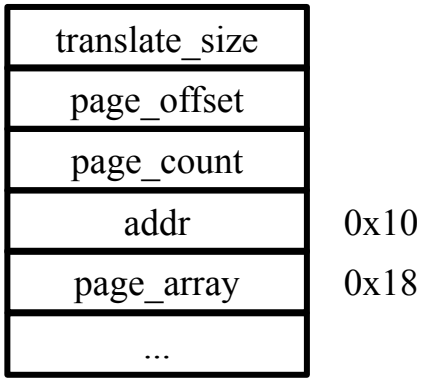
CVE-2018-6981: Uninitialized Stack Variable

```
void __usercall vmxnet3_reg_cmd() {
  ...
  case 4: // VMXNET3_CMD_UPDATE_MAC_FILTERS
*     dma_memory_create(..., &page);
      vmxnet3_cmd_update_mac_filters(v6, &page, a5);
*     destruct_page_struct(&page);
  }
    break;
  ...
}
char __fastcall dma_memory_create(unsigned addr, uint64
size, ..., page_struct *page)
{
  // check the addr
*  if ( addr > v5 || !size || size > v5 - addr + 1 )
*    return 0;
  set_page_struct(addr, size, a3, a4, page);
  return 1;
}
```

check the size and addr

```
void destruct_page_struct(page_struct *a1)
{
   if(page_count == 1)
      free(a1->addr)
   // free the pointer on stack
   else
      free(a1->page_array);
}
```

page



a struct on stack for address translation

CVE-2018-6981: Uninitialized Stack Variable

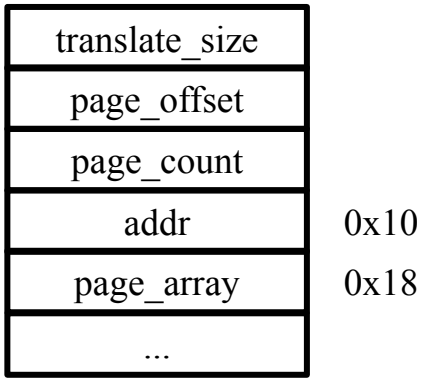
```
void __usercall vmxnet3_reg_cmd() {
  ...
  case 4: // VMXNET3_CMD_UPDATE_MAC_FILTERS
*     dma_memory_create(..., &page);
      vmxnet3_cmd_update_mac_filters(v6, &page, a5);
*     destruct_page_struct(&page);
  }
    break;
  ...
}
char __fastcall dma_memory_create(unsigned addr, uint64
size, ..., page_struct *page)
{
  // check the addr
*  if ( addr > v5 || !size || size > v5 - addr + 1 )
*    return 0;
  set_page_struct(addr, size, a3, a4, page);
  return 1;
}
```

illegal combination! skip initalization

initialization

```
void destruct_page_struct(page_struct *a1)
{
   if(page_count == 1)
      free(a1->addr)
   // free the pointer on stack
   else
      free(a1->page_array);
}
```

page



a struct on stack for address translation

CVE-2018-6981: Uninitialized Stack Variable

```
void __usercall vmxnet3_reg_cmd() {
  ...
  case 4: // VMXNET3_CMD_UPDATE_MAC_FILTERS
*     dma_memory_create(..., &page);
      vmxnet3_cmd_update_mac_filters(v6, &page, a5);
*     destruct_page_struct(&page);
  }
    break;
  ...
}
char __fastcall dma_memory_create(unsigned addr, uint64
size, ..., page_struct *page)
{
  // check the addr
*  if ( addr > v5 || !size || size > v5 - addr + 1 )
*    return 0;
  set_page_struct(addr, size, a3, a4, page);
  return 1;
}
```

initialization

return and enter into the "destruct" function

```
void destruct_page_struct(page_struct *a1)
{
  if(page_count == 1)
    free(a1->addr)
  // free the pointer on stack
  else
    free(a1->page_array);
}
```

page

translate_size	
page_offset	
page_count	
addr	0x10
page_array	0x18
...	

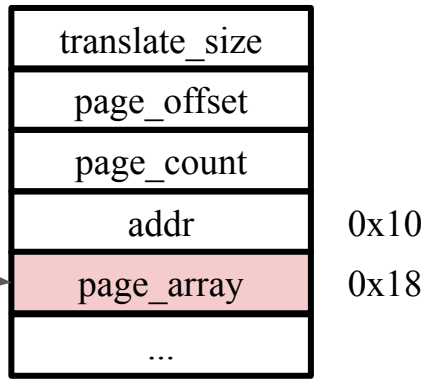
a struct on stack for address translation

Uninit Variable ⟶ Arbitrary Address Free

```
void __usercall handle_port_io(__int64 a1, __int64 a2, __int64 a3) {
 ...

 v3 = *(a1 + 4);
 v4 = *(a1 + 13);
 read_or_write = *(a1 + 48);
 ...
 if ( *(a1 + 60) && (v10 = *(a1 + 52) << 12, v10 > 0x8000) )
   v11 = malloc_heap_memory(v10); // copy the data into heap
 else
   v11 = &v35;
 if ( read_or_write & 1 )
 { if ( *(v8 + 60) )
   {   ...
       v15 = v11;
       do
       { ...
         memcpy(v15, v18, v17); // copy the data into stack
 ...
```

Abusing a function to spray stack!

```
void destruct_page_struct
(page_struct *a1)
{
   ...
   // free the pointer on stack
   free(a1->page_array);
}
```



CVE-2018-6982: Uninitialized Stack Variable

```
bool __fastcall vmxnet3_cmd_get_coalesce(__int64 a1, char a2){
     struct buffer {
          uint64_t member0;
          uint64_t member1;
     };
     size_t length;
     ...
     get_length_from_guest(..., &length);
     if(length != 16)
          return;
     ...
*    buffer.member0 = 0xFA000000003LL;

    // first 8-byte of src is initialized, but 16-byte is read
    // (length == 16)
*    write_back_to_guest(v19, &buffer, length, ...);
    return 1;
  }
  ...
 }
```

At first, all stack variables are uninitlized

CVE-2018-6982: Uninitialized Stack Variable

```
bool __fastcall vmxnet3_cmd_get_coalesce(__int64 a1, char a2){
     struct buffer {
          uint64_t member0;
          uint64_t member1;
     };
     size_t length;
     ...
     get_length_from_guest(..., &length);
     if(length != 16)
          return;
     ...
*    buffer.member0 = 0xFA000000003LL;

    // first 8-byte of src is initialized, but 16-byte is read
    // (length == 16)
*    write_back_to_guest(v19, &buffer, length, ...);
    return 1;
  }
  ...
 }
```

set length

must be 16

CVE-2018-6982: Uninitialized Stack Variable

```
bool __fastcall vmxnet3_cmd_get_coalesce(__int64 a1, char a2){
     struct buffer {
          uint64_t member0;
          uint64_t member1;
     };
     size_t length;
     ...
     get_length_from_guest(..., &length);
     if(length != 16)
          return;
     ...
*    buffer.member0 = 0xFA000000003LL;
     // first 8-byte of src is initialized, but 16-byte is read
     // (length == 16)
*     write_back_to_guest(v19, &buffer, length, ...);
     return 1;
  }
  ...
 }
```

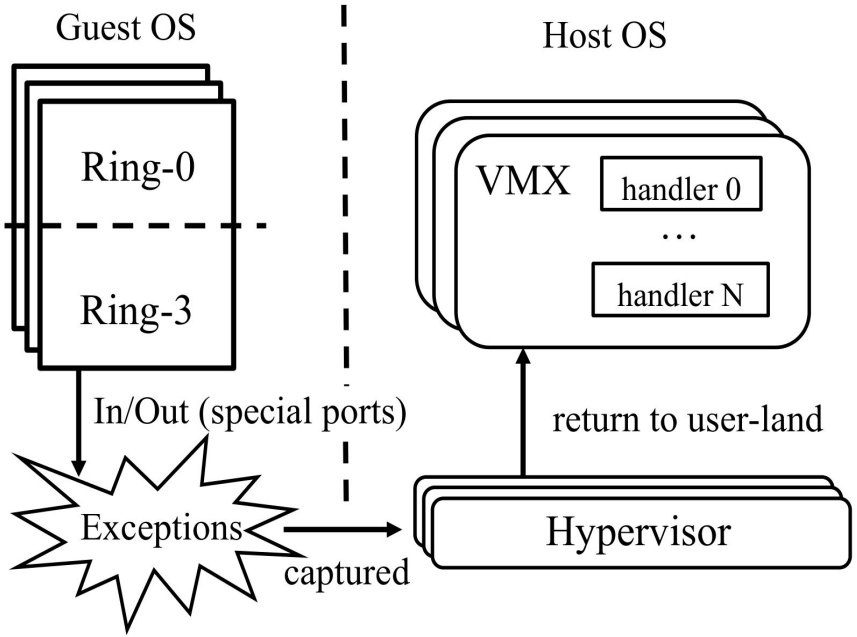
initialize the first 8-byte

CVE-2018-6982: Uninitialized Stack Variable

```
bool __fastcall vmxnet3_cmd_get_coalesce(__int64 a1, char a2){
     struct buffer {
          uint64_t member0;
          uint64_t member1;
     };
     size_t length;
     ...
     get_length_from_guest(..., &length);
     if(length != 16)
          return;
     ...
*    buffer.member0 = 0xFA000000003LL;

     // first 8-byte of src is initialized, but 16-byte is read
     // (length == 16)
*    write_back_to_guest(v19, &buffer, length, ...);
     return 1;
  }
  ...
 }
```

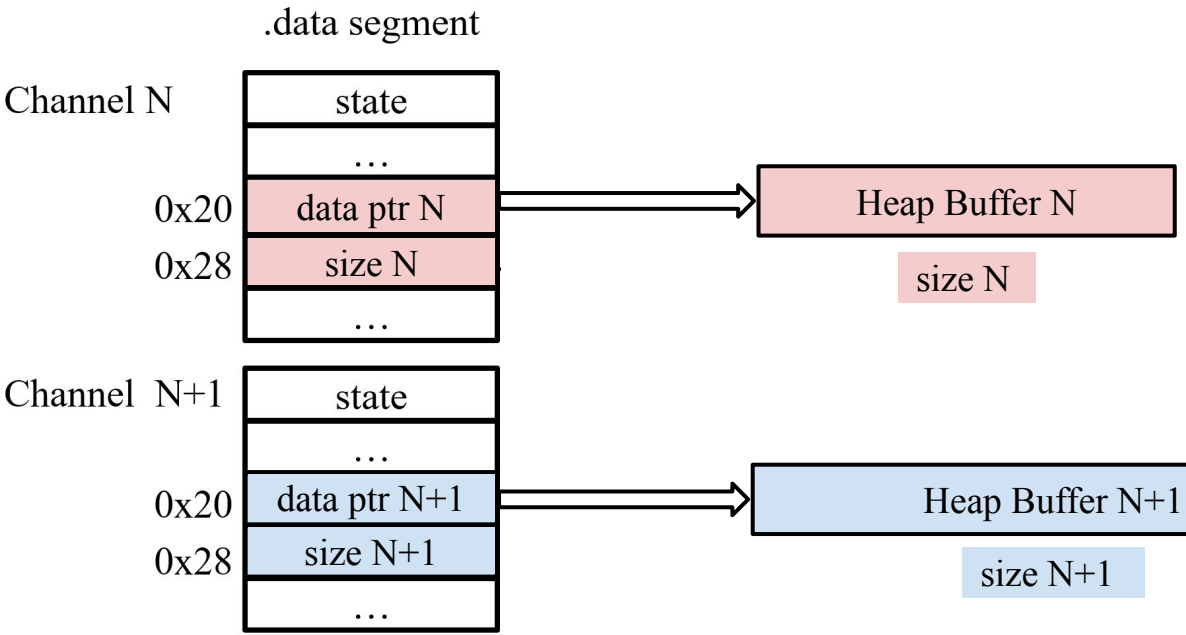
member1(uninitilized) will be write back to the guest OS

Abusing a special RPC named “backdoor”



- A special hypercall
- Guest
  - Send messages through special I/O ports
- Host
  - Handle messages in VMX process

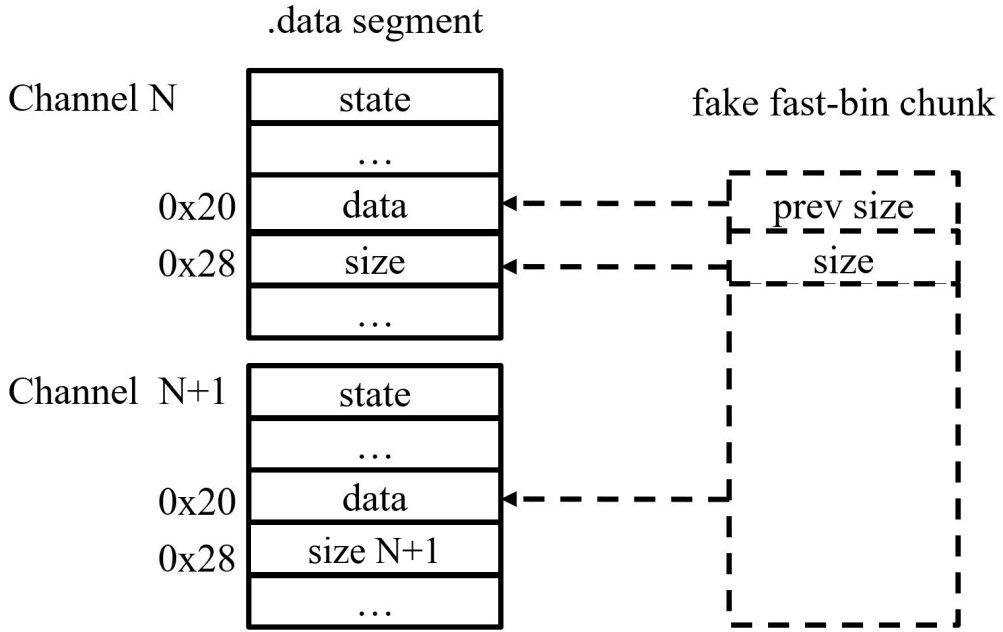
Abusing RPCI to manipulate heap



- Controllable size
- Controllable buffer

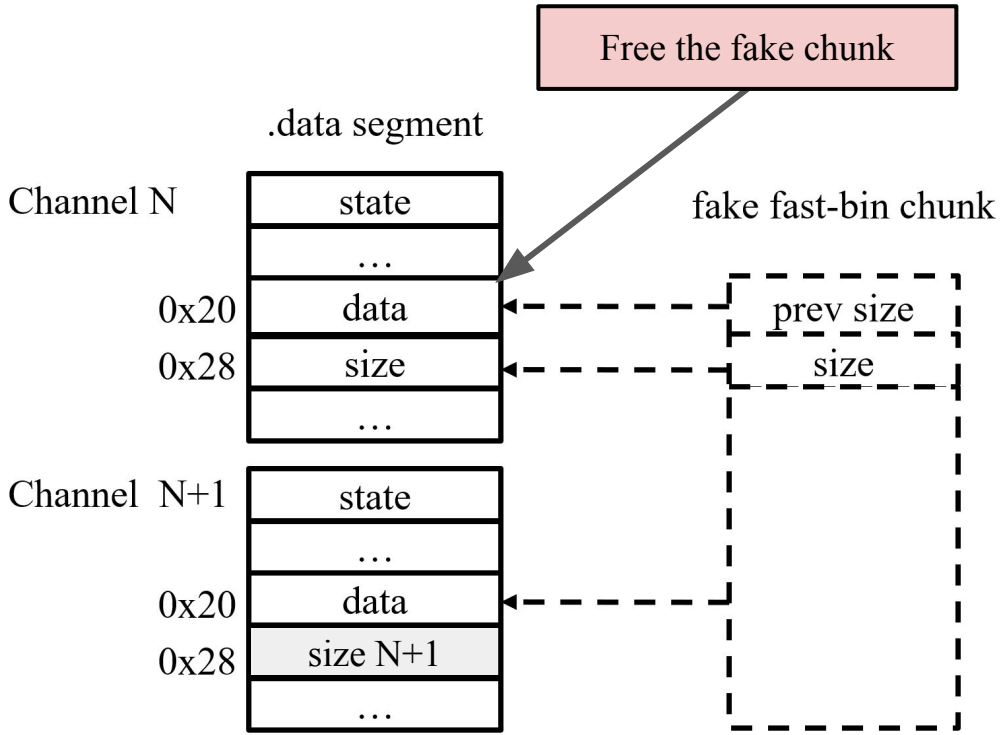
Abusing the feature of heap

- Main idea
  - fake a fast chunk on metadata
- Constraints
  - check current chunk's size
  - check next chunk's size



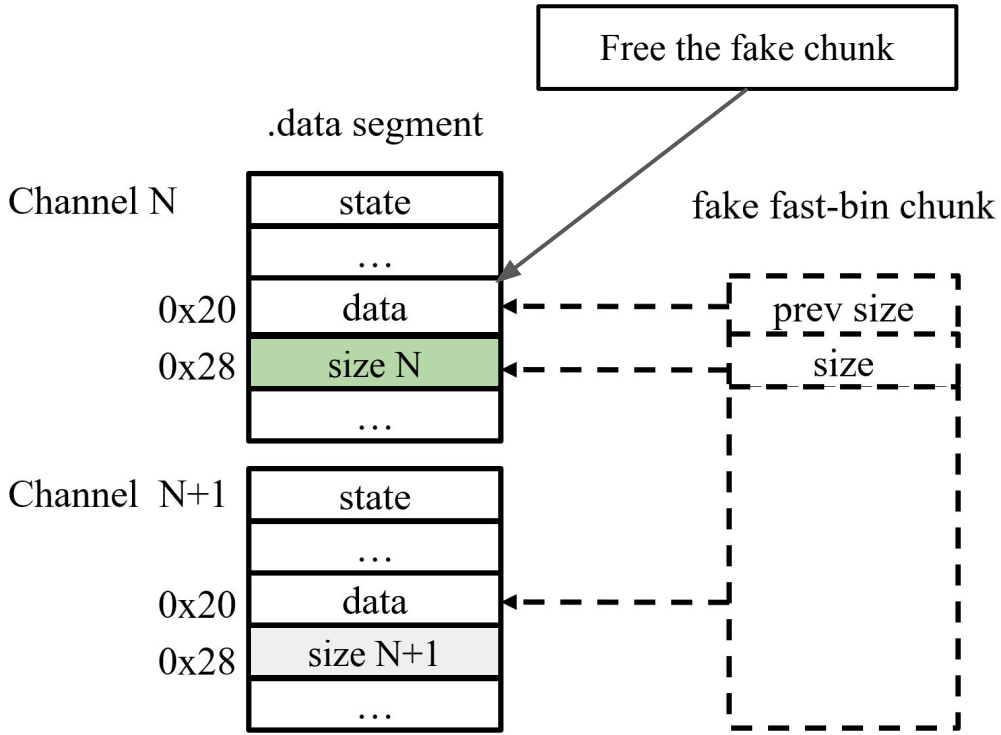
Abusing the feature of heap

- Main idea
  - fake a fast chunk on metadata
- Constraints
  - check current chunk's size
  - check next chunk's size



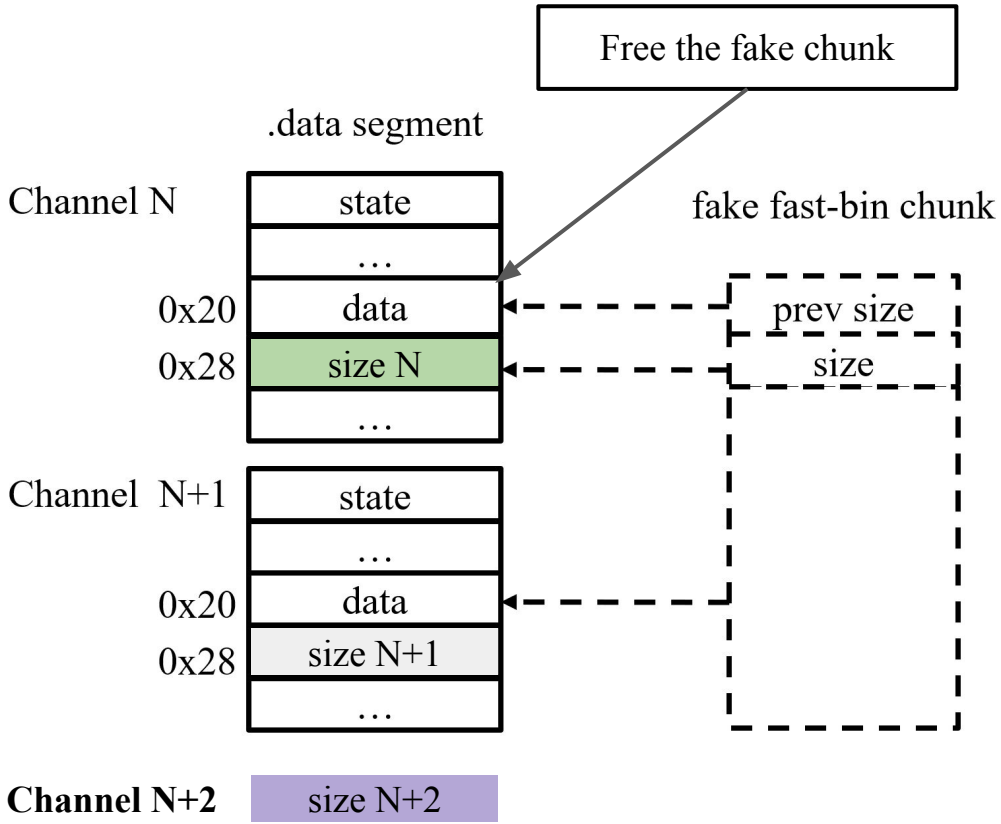
Abusing the feature of heap

- Main idea
  - fake a fast chunk on metadata
- Constraints
  - check current chunk's size
  - check next chunk's size

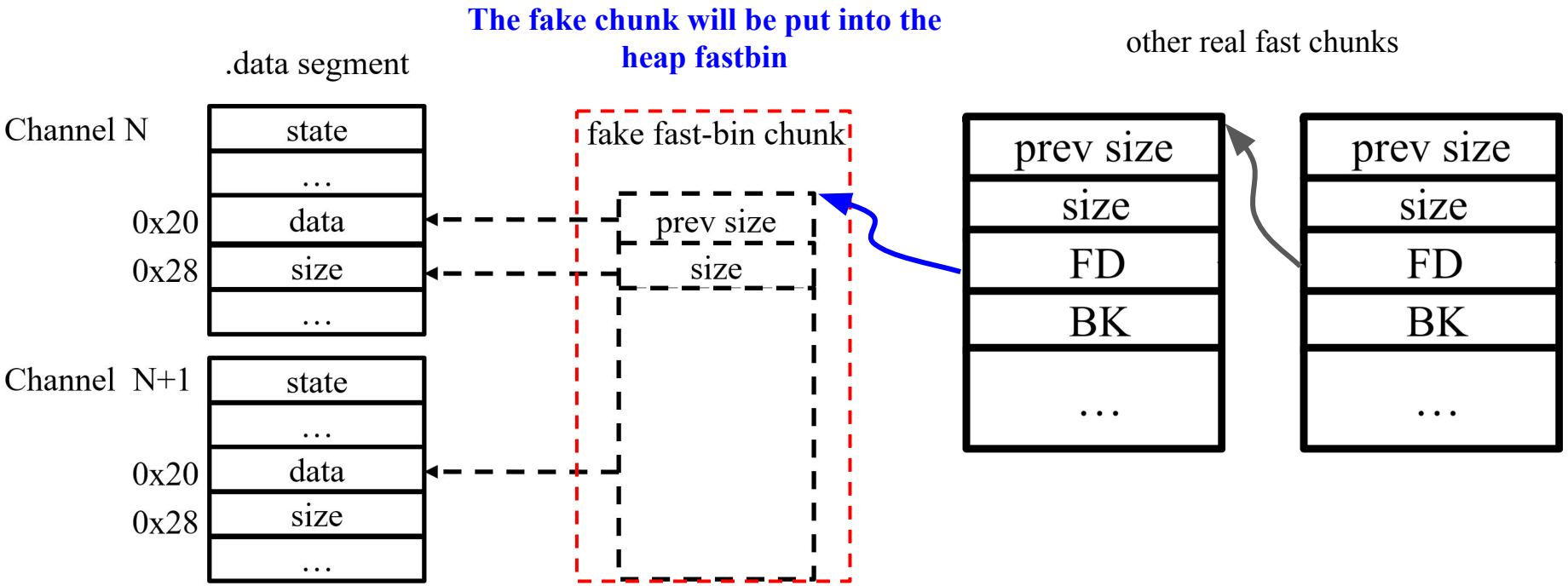


Abusing the feature of heap

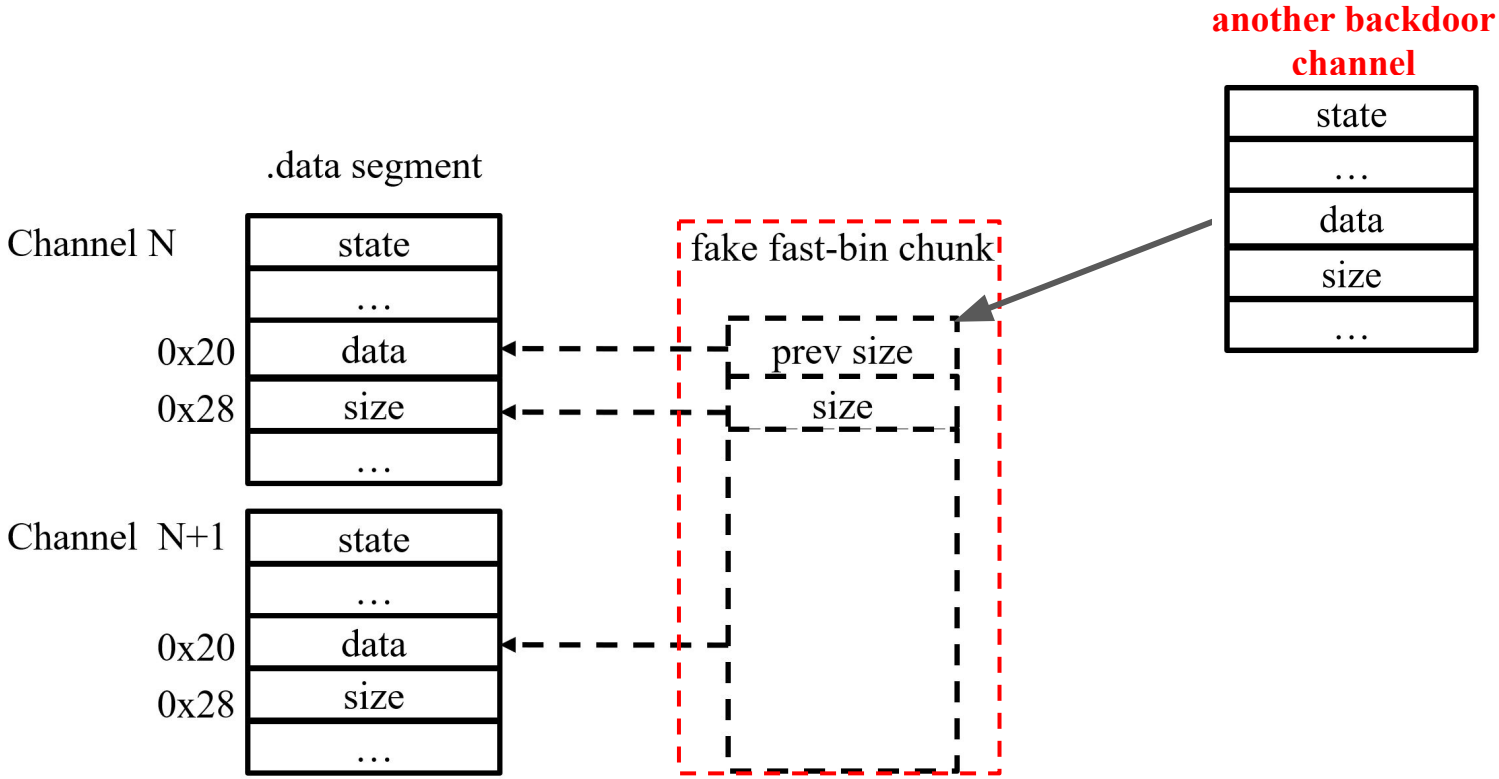
- Main idea
  - fake a fast chunk on metadata
- Constraints
  - check current chunk's size
  - check next chunk's size



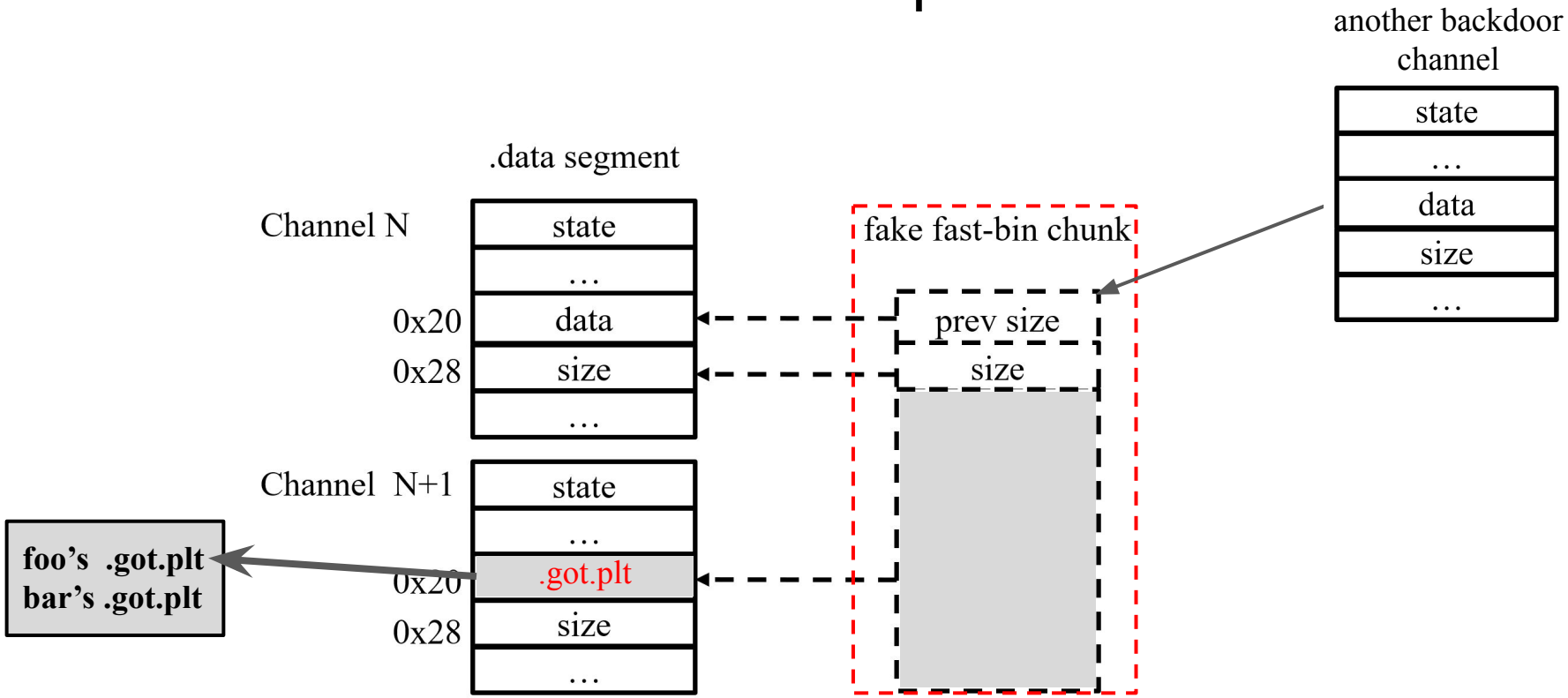
fake chunk → fastbin linklist



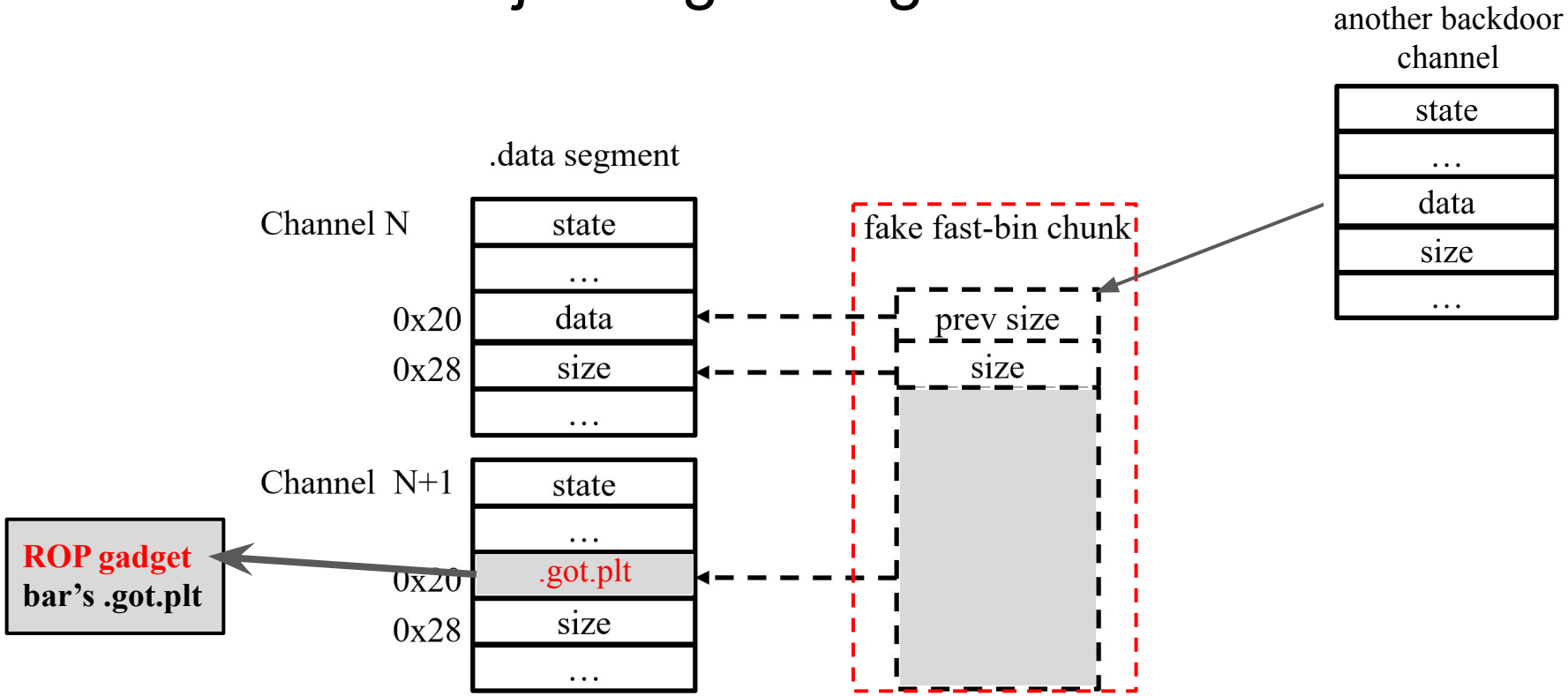
Reallocate the fake chunk with another channel



Overwrite the next channel's pointer



Control Flow Hijacking through the next channel



Bypass the sandbox

- A loophole in the sandbox profile
  - The sandbox grants us to read and write the /var/run directory
- We overwrite the **inetd.conf** file and bind a specific port to /bin/bash

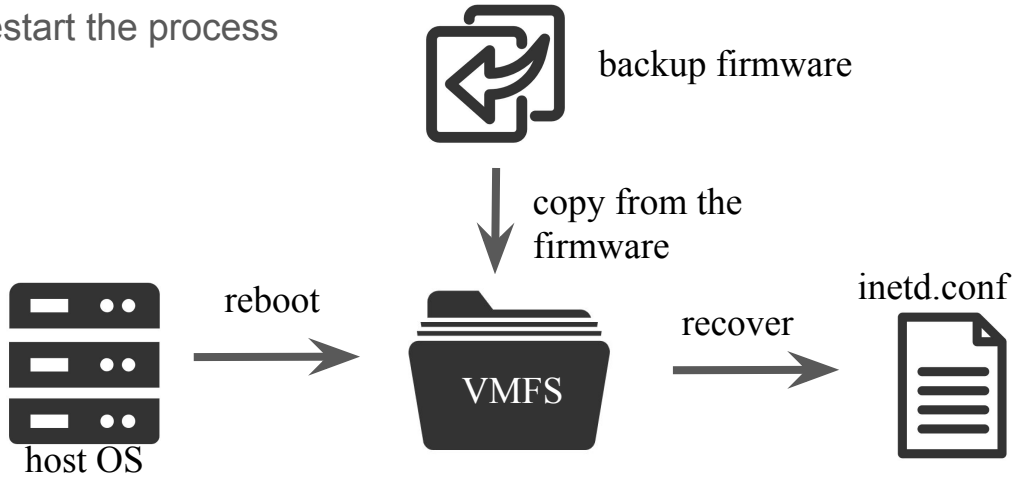
```
# Rules applicable for all VMs

-s genericSys grant
-s ioctlSys grant
-s vsiReadSys grant
...

-c unix_socket_create grant
-c unix_stream_socket_bind grant
-c unix_dgram_socket_bind grant
...

-p inet_socket_bind all grant
-p inet_socket_connect loopback grant
-p inet_socket_connect nonloopback grant
...

-d tpm2emuObj tpm2emuDom file_exec grant

-r /var/run rw
-r /var/lock rw
...
```

Bypass the sandbox

```
# Internet server configuration database

# Remote shell access

ssh      stream   tcp   nowait   root   /usr/lib/vmware/openssh/bin/sshd
...

ssh      stream   tcp6  nowait   root   /usr/lib/vmware/openssh/bin/sshd
...


authd   stream    tcp   nowait   root   /bin/sh     ...

authd   stream    tcp6  nowait   root   /bin/sh     ...
```

Get Shell

- Restart the **`inetd`** process
  - Reboot ✗
  - Kill the process ✓
    - The watchdog helps us to restart the process



Success Rate

Version	Build number	V?	S?	Success	Adaptable
ESXi 6.7	10764712	Y	Y	93.3%	Y
ESXi 6.7	10302608	Y	Y	86.7%	Y
ESXi 6.7	10176752	Y	Y	90.0%	Y
ESXi 6.7	9484548	Y	Y	86.7%	Y
ESXi 6.7	9214924	Y	Y	93.3%	Y
ESXi 6.7	8941472	Y	Y	90.0%	Y
ESXi 6.5	<10719125	Y	Y	N/A	N
ESXi 6.0	Any	N	N	N/A	N/A

V: Vulnerable or not, S: Sandboxed or not

- ESXi 6.7:
  - MAX: 93.3%
  - AVG: 90.0%
- Failing Reasons
  - stack pollution
  - heap stability

Thanks!