GhostRace: Exploiting and Mitigating Speculative Race Conditions

Hany Ragab, Andrea Mambretti, Anil Kurmus, and Cristiano Giuffrida

VUSec

IBM Research | Zurich

In This Talk

In This Talk

- Speculative Race Condition

In This Talk

- Speculative Race Condition
- Inter-Process Interrupt Storming

In This Talk

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)

In This Talk

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)
- PoC Exploit (12KB/s)




In This Talk

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)
- PoC Exploit (12KB/s)
- Mitigation (5%)

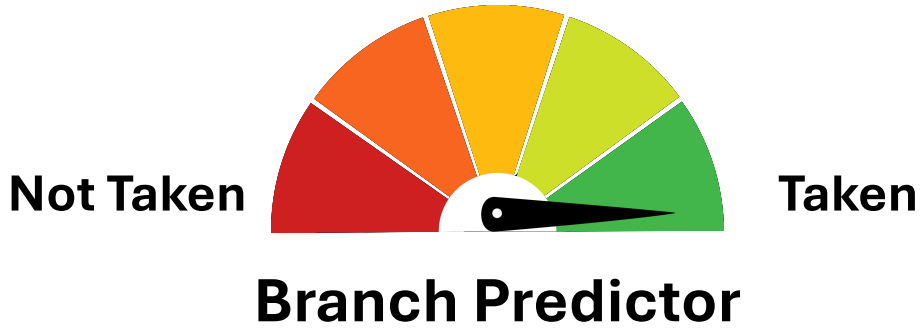




Background

Spectre v1

```
if (condition){
    ...
}
```

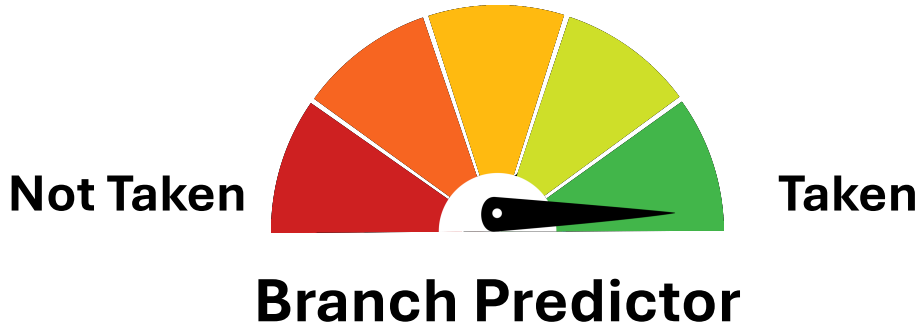


Background

Spectre v1

Conditional Branch Misprediction

```
if (condition){
    ...
}
```



Background

Concurrency Bugs

Background

Concurrency Bugs

SHARED DATA

```
struct A{

  struct B* Bptr;
}
```

```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

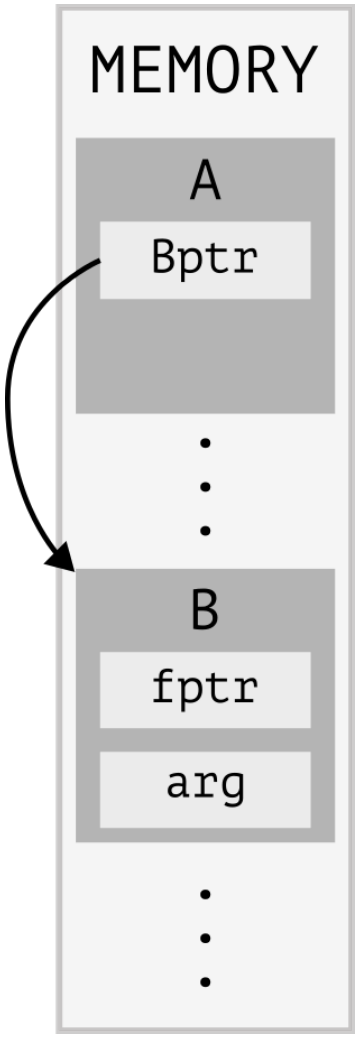
Background

Concurrency Bugs

SHARED DATA

```
struct A{

   struct B* Bptr;
}
```

```
struct B{
  void (*fptr)(...);
  void *arg;
}
```



Background

Concurrency Bugs

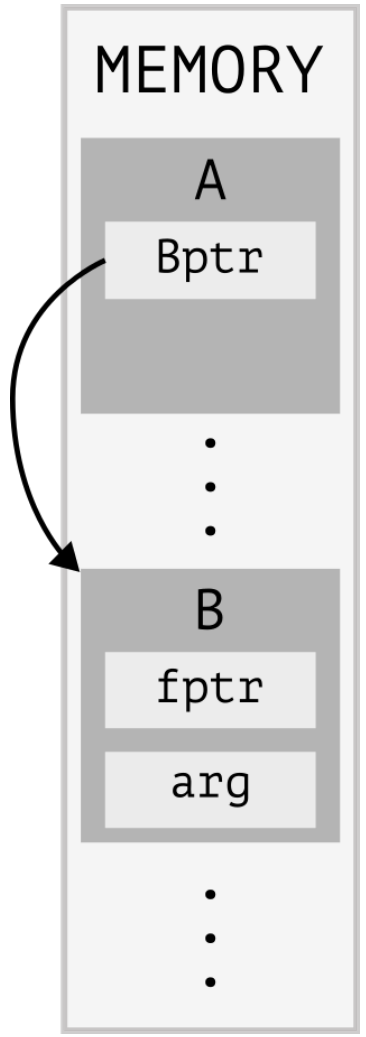
SHARED DATA

```
struct A{
   struct B* Bptr;
}
```

```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```



Background

Concurrency Bugs

SHARED DATA

```
struct A{
   struct B* Bptr;
}
```

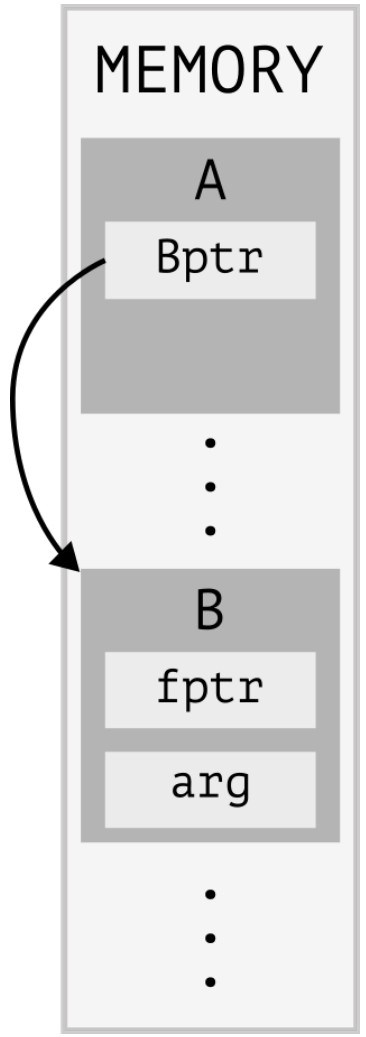
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```

THREAD 2

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```



Background

Concurrency Bugs

SHARED DATA

```
struct A{
   struct B* Bptr;
}
```

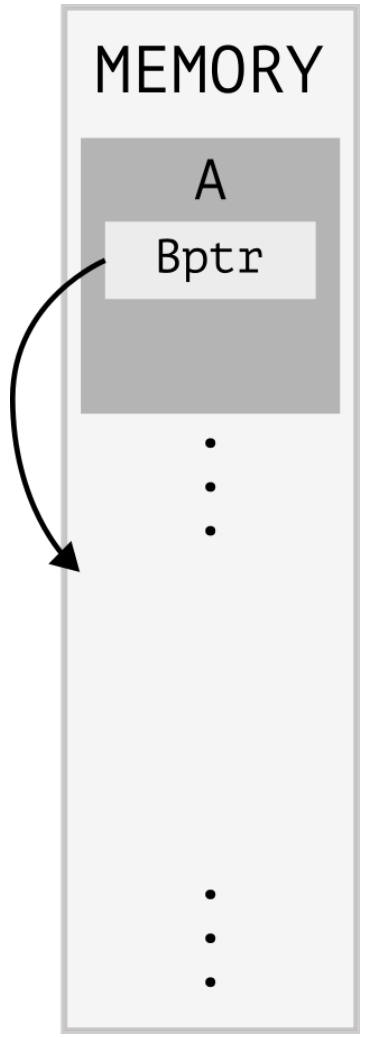
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```

THREAD 2

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```



Background

Concurrency Bugs

SHARED DATA

```
struct A{
   struct B* Bptr;
}
```

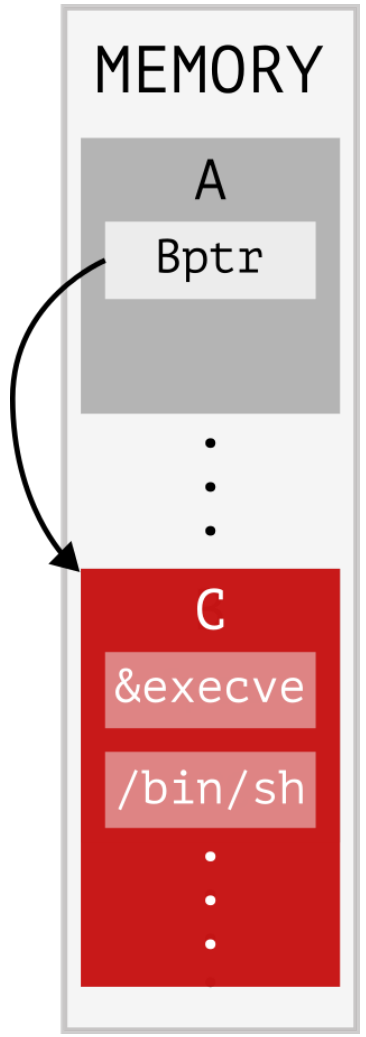
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```

THREAD 2

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```



Concurrent Use-After-Free Attack

SHARED DATA

```
struct A{
   struct B* Bptr;
}
```

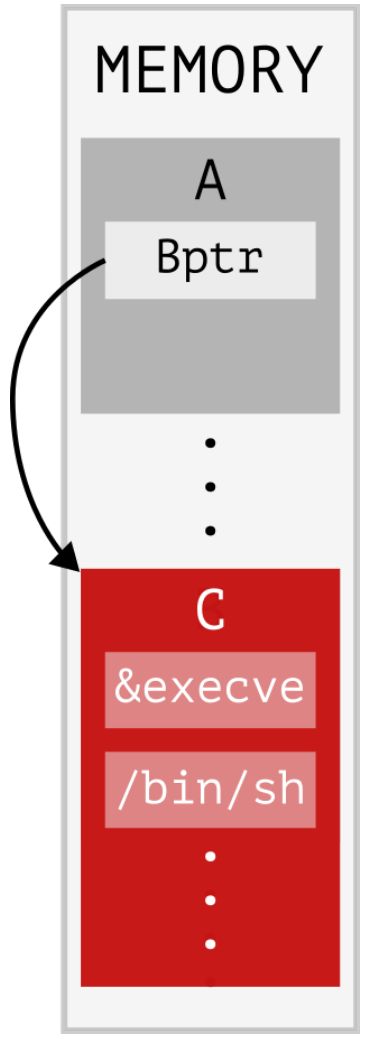
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```

THREAD 2

```
func(A_ptr){

  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;

}
```



Background

Concurrency Bugs

SHARED DATA

```
struct A{
  struct mutex m;
  struct B* Bptr;
}
```

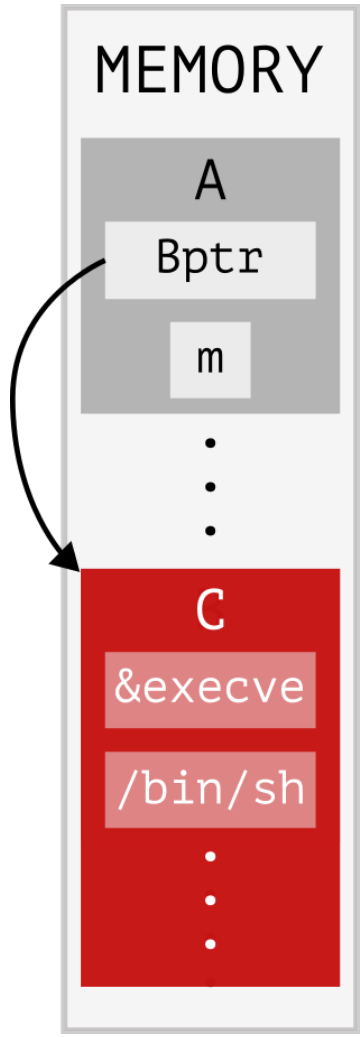
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){
  mutex_lock(&Aptr->m);
  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;
  mutex_unlock(&Aptr->m);
}
```

THREAD 2

```
func(A_ptr){
  mutex_lock(&Aptr->m);
  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;
  mutex_unlock(&Aptr->m);
}
```



Background

Concurrency Bugs

SHARED DATA

```
struct A{
  struct mutex m;
  struct B* Bptr;
}
```

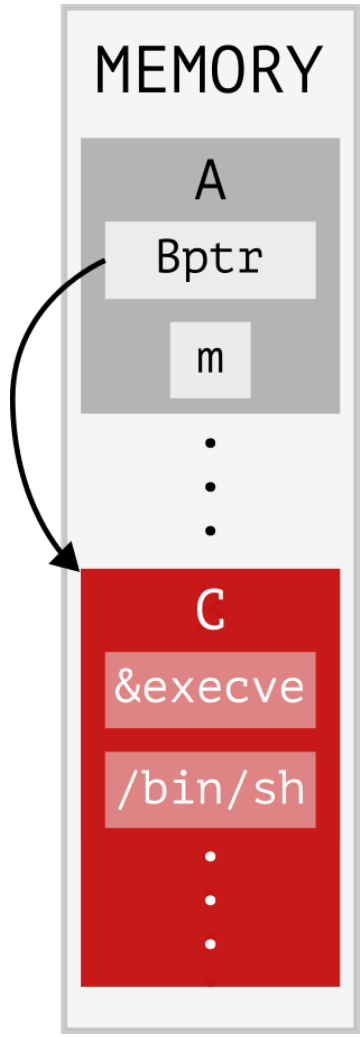
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){
  mutex_lock(&Aptr->m);
  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;
  mutex_unlock(&Aptr->m);
}
```

THREAD 2

```
func(A_ptr){
  mutex_lock(&Aptr->m);
  if(Aptr->Bptr->fptr){
    Aptr->Bptr->fptr(arg);
  }
  kfree(Aptr->Bptr);
  Aptr->Bptr = NULL;
  mutex_unlock(&Aptr->m);
}
```



Race Free Critical Region

SHARED DATA

```
struct A{
   struct mutex m;
   struct B* Bptr;
}
```

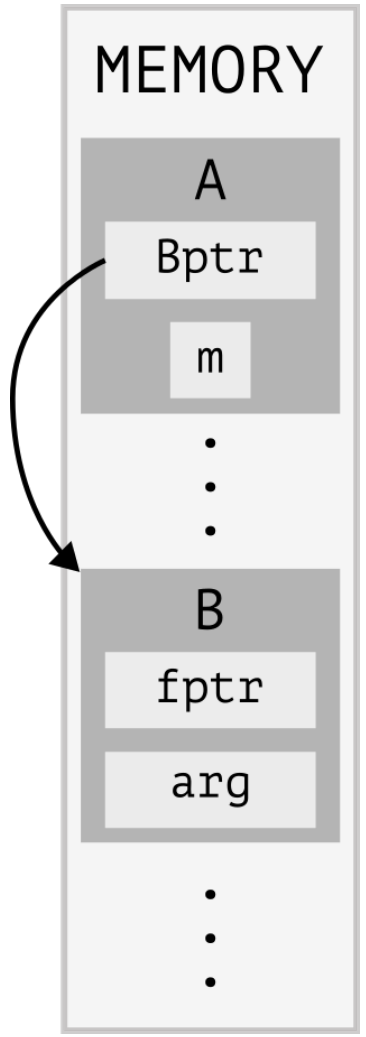
```
struct B{
  void (*fptr)(...);
  void *arg;
}
```

THREAD 1

```
func(A_ptr){
  mutex_lock(&Aptr->m);
   if(Aptr->Bptr->fptr){
      Aptr->Bptr->fptr(arg);
   }
   kfree(Aptr->Bptr);
   Aptr->Bptr = NULL;
  mutex_unlock(&Aptr->m);
}
```

THREAD 2

```
func(A_ptr){
  mutex_lock(&Aptr->m);
   if(Aptr->Bptr->fptr){
      Aptr->Bptr->fptr(arg);
   }
   kfree(Aptr->Bptr);
   Aptr->Bptr = NULL;
  mutex_unlock(&Aptr->m);
}
```



Real Example

NFC Gadget

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...);
   void *cb_context;
   ...
}
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

How do synchronization primitives behave during speculative execution?

How do synchronization primitives behave during speculative execution?

What are the security implications for modern operating systems?

Objective

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...);
   void *cb_context;
   ...
}
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

Objective

Speculatively Hijack The Control-Flow

```
struct nfc_hc
   ...
   struct mute
   ...
   struct hci_
   ...
}
```

```
) {
...);
ntext;
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...);
   void *cb_context;
   ...
}
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

1. Bypass The Sync. Primitive

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...
   void *cb_contex
   ...
}
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

1. Bypass The Sync. Primitive

2. Create an Exploitation Window

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...
   void *cb_contex
   ...
}
```

THREAD 1

```
fc_hci_msg_tx_work(hdev){
 ...
 mutex_lock(&hdev->msg_tx_mutex);
 if(hdev->cmd_pending_msg->cb){
    hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
 }
 kfree(hdev->cmd_pending_msg);
 hdev->cmd_pending_msg = NULL;
 ...
 mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

1. Bypass The Sync. Primitive

2. Create an Exploitation Window

3. Reallocate The Freed Memory With a Malicious Object

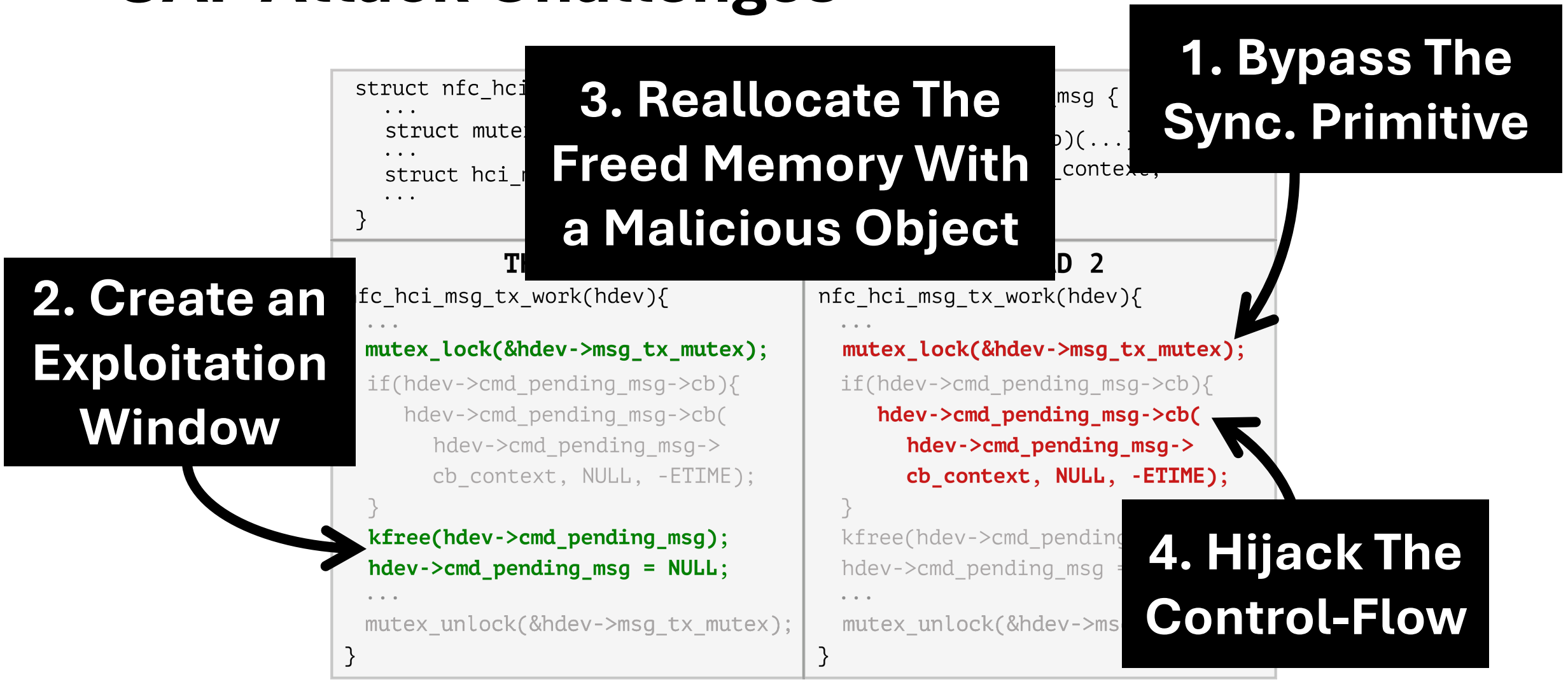
```
struct nfc_hci
  ...
  struct mute
  ...
  struct hci_
  ...
}
```

```
msg {
)(...
_contex
```

```
fc_hci_msg_tx_work(hdev){
...
mutex_lock(&hdev->msg_tx_mutex);
if(hdev->cmd_pending_msg->cb){
   hdev->cmd_pending_msg->cb(
     hdev->cmd_pending_msg->
     cb_context, NULL, -ETIME);
}
kfree(hdev->cmd_pending_msg);
hdev->cmd_pending_msg = NULL;
...
mutex_unlock(&hdev->msg_tx_mutex);
}
```

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges



UAF Attack Challenges

1. Bypass The Sync. Primitive

2. Create an Exploitation Window

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...
   void *cb_contex
   ...
}
```

THREAD 1

```
fc_hci_msg_tx_work(hdev){
 ...
 mutex_lock(&hdev->msg_tx_mutex);
 if(hdev->cmd_pending_msg->cb){
    hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
 }
 kfree(hdev->cmd_pending_msg);
 hdev->cmd_pending_msg = NULL;
 ...
 mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

1. Bypass The Sync. Primitive

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_msg {
   ...
   void (*cb)(...
   void *cb_contex
   ...
}
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

Synchronization Primitives

Synchronization Primitives

Mutex lock on x86 arch:

- Conditional branch

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
```

Synchronization Primitives

Mutex lock on x86 arch:

- Conditional branch
- A series of inlined function calls

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
      ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
        ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg(&lock, , ...)
            ↳arch_try_cmpxchg((&lock, , ...)
              ↳__raw_try_cmpxchg(ptr, ...)({
```

Synchronization Primitives

Mutex lock on x86 arch:

- Conditional branch
- A series of inlined function calls
- Atomic compare and exchange

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
      ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
        ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg(&lock, , ...)
            ↳arch_try_cmpxchg((&lock, , ...)
              ↳__raw_try_cmpxchg(ptr, ...)({
                  asm volatile(
                      "lock cmpxchgq %2, %1"
                      : "=a" (ret), "+m" (*ptr)
                      : "r" (new), "0" (old)
                      : "memory"
                      );
                })
      return true;
  ...
}
```

Synchronization Primitives

Mutex lock on x86 arch:

- Conditional branch
- A series of inlined function calls
- Atomic compare and exchange

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
        ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
            ↳arch_atomic_try_cmpxchg(&lock, , ...)
              ↳arch_try_cmpxchg((&lock, , ...)
                ↳__raw_try_cmpxchg(ptr, ...)({
                    asm volatile(
                        "lock cmpxchgq %2, %1"
                        : "=a" (ret), "+m" (*ptr)
                        : "r" (new), "0" (old)
                        : "memory"
                        );
                  })
      return true;
  ...
}
```

Synchronization Primitives

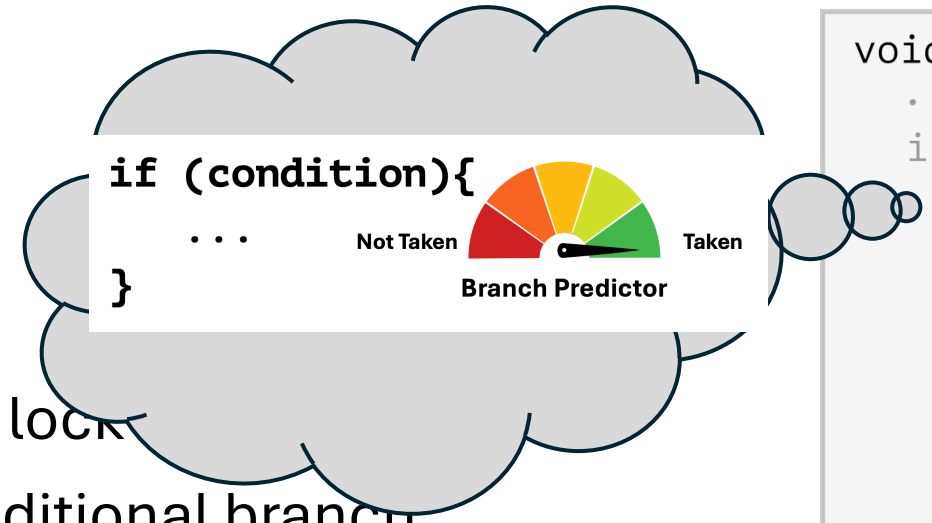
Mutex lock on x86 arch:

- Conditional branch
- A series of inlined function calls
- Atomic compare and exchange

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
      ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
        ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg(&lock, , ...)
            ↳arch_try_cmpxchg((&lock, , ...)
              ↳__raw_try_cmpxchg(ptr, ...)({
                  asm volatile(
                      "lock cmpxchgq %2, %1"
                      : "=a" (ret), "+m" (*ptr)
                      : "r" (new), "0" (old)
                      : "memory"
                      );
})
```

Atomic != Serializing

Synchronization Primitives



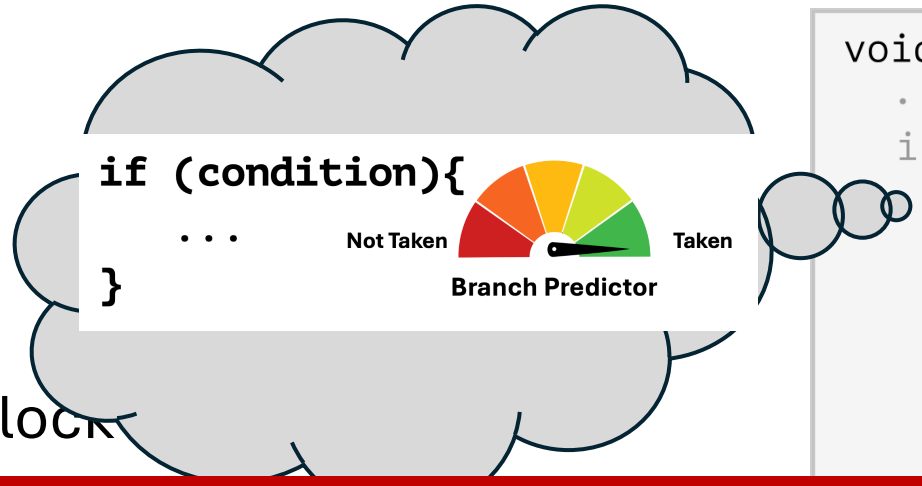
Mutex lock

- Conditional branch
- A series of inlined function calls
- Atomic compare and exchange

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
      └arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
        └arch_atomic_try_cmpxchg_acquire(&lock, , ...)
          └arch_atomic_try_cmpxchg(&lock, , ...)
            └arch_try_cmpxchg((&lock, , ...)
              └__raw_try_cmpxchg(ptr, ...)({
                  asm volatile(
                      "lock cmpxchgq %2, %1"
                      : "=a" (ret), "+m" (*ptr)
                      : "r" (new), "0" (old)
                      : "memory"
                      );
})
```

Atomic != Serializing

Synchronization Primitives



Mutex lock

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
      └arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
        └arch_atomic_try_cmpxchg_acquire(&lock, , ...)
          └arch_atomic_try_cmpxchg(&lock, , ...)
            └arch_try_cmpxchg((&lock, , ...)
              __raw_try_cmpxchg(ptr, ...)({
                asm volatile(
                  "lock cmpxchgq %2, %1"
                  : "=a" (ret), "+m" (*ptr)
                  : "r" (new), "0" (old)
                  : "memory"
                  );
})
```

Speculative Race Condition

UAF Attack Challenges

1. Bypass The Sync. Primitive:
Speculative Race Condition

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_
   ...
   void (*cb
   void *cb_
   ...
}
```

THREAD 1

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

1. Bypass The Sync. Primitive:
Speculative Race Condition

2. Create an Exploitation Window

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_
   ...
   void (*cb
   void *cb_
   ...
}
```

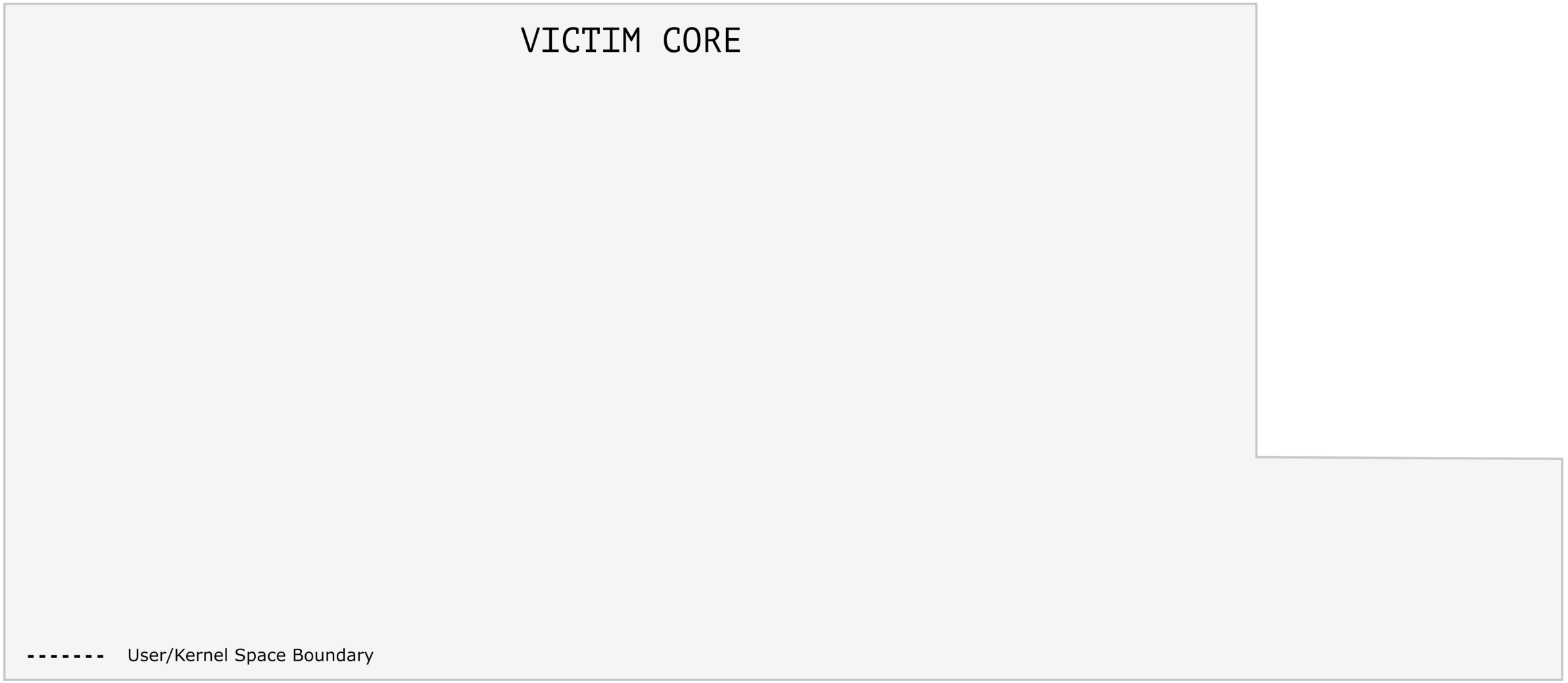
THREAD 1

```
fc_hci_msg_tx_work(hdev){
 ...
 mutex_lock(&hdev->msg_tx_mutex);
 if(hdev->cmd_pending_msg->cb){
    hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
 }
 kfree(hdev->cmd_pending_msg);
 hdev->cmd_pending_msg = NULL;
 ...
 mutex_unlock(&hdev->msg_tx_mutex);
}
```

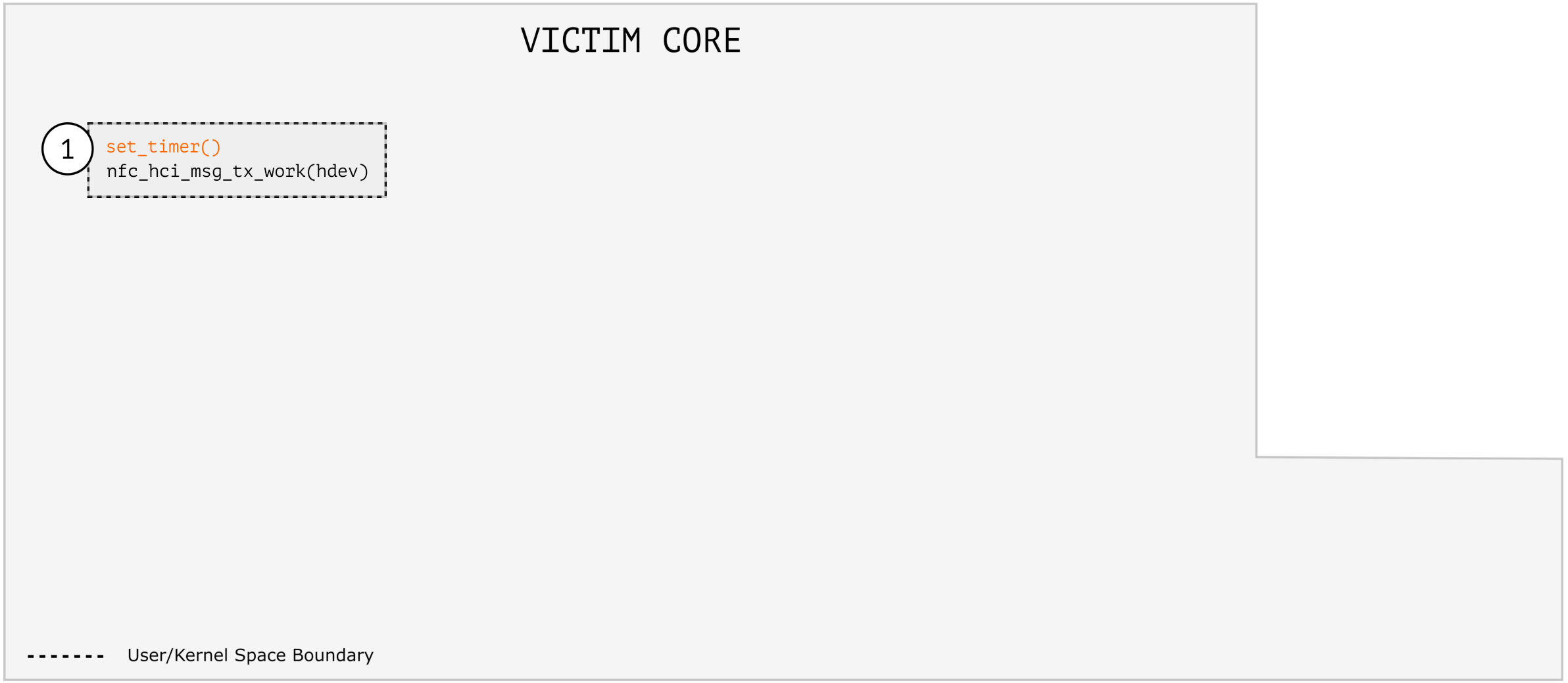
THREAD 2

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
        hdev->cmd_pending_msg->
        cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

Creating UAF Exploitation Window



Creating UAF Exploitation Window



Creating UAF Exploitation Window

VICTIM CORE

1 `set_timer()`
`nfc_hci_msg_tx_work(hdev)`

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

------- User/Kernel Space Boundary

Creating UAF Exploitation Window

VICTIM CORE

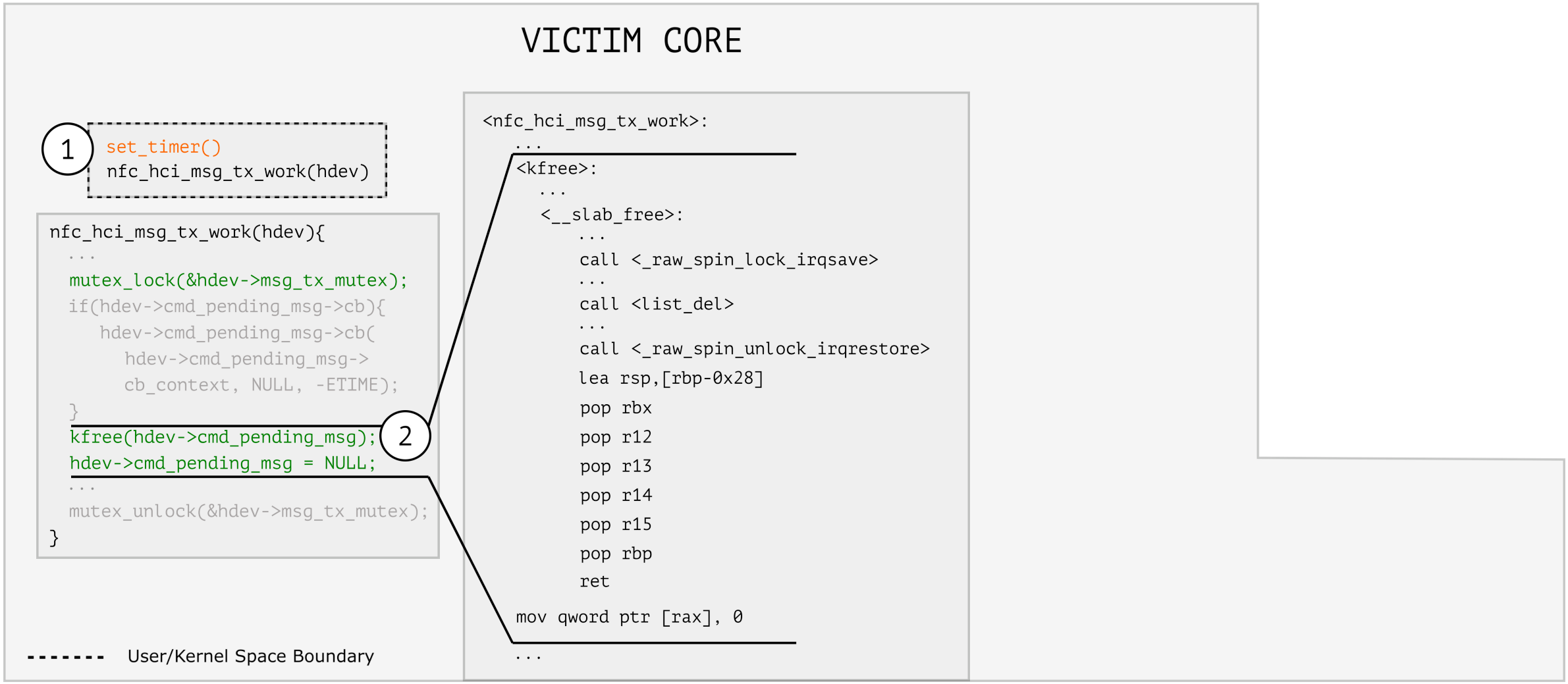
1 `set_timer()`
`nfc_hci_msg_tx_work(hdev)`

```
nfc_hci_msg_tx_work(hdev){
  ...
  mutex_lock(&hdev->msg_tx_mutex);
  if(hdev->cmd_pending_msg->cb){
     hdev->cmd_pending_msg->cb(
       hdev->cmd_pending_msg->
       cb_context, NULL, -ETIME);
  }
  kfree(hdev->cmd_pending_msg);
  hdev->cmd_pending_msg = NULL;
  ...
  mutex_unlock(&hdev->msg_tx_mutex);
}
```

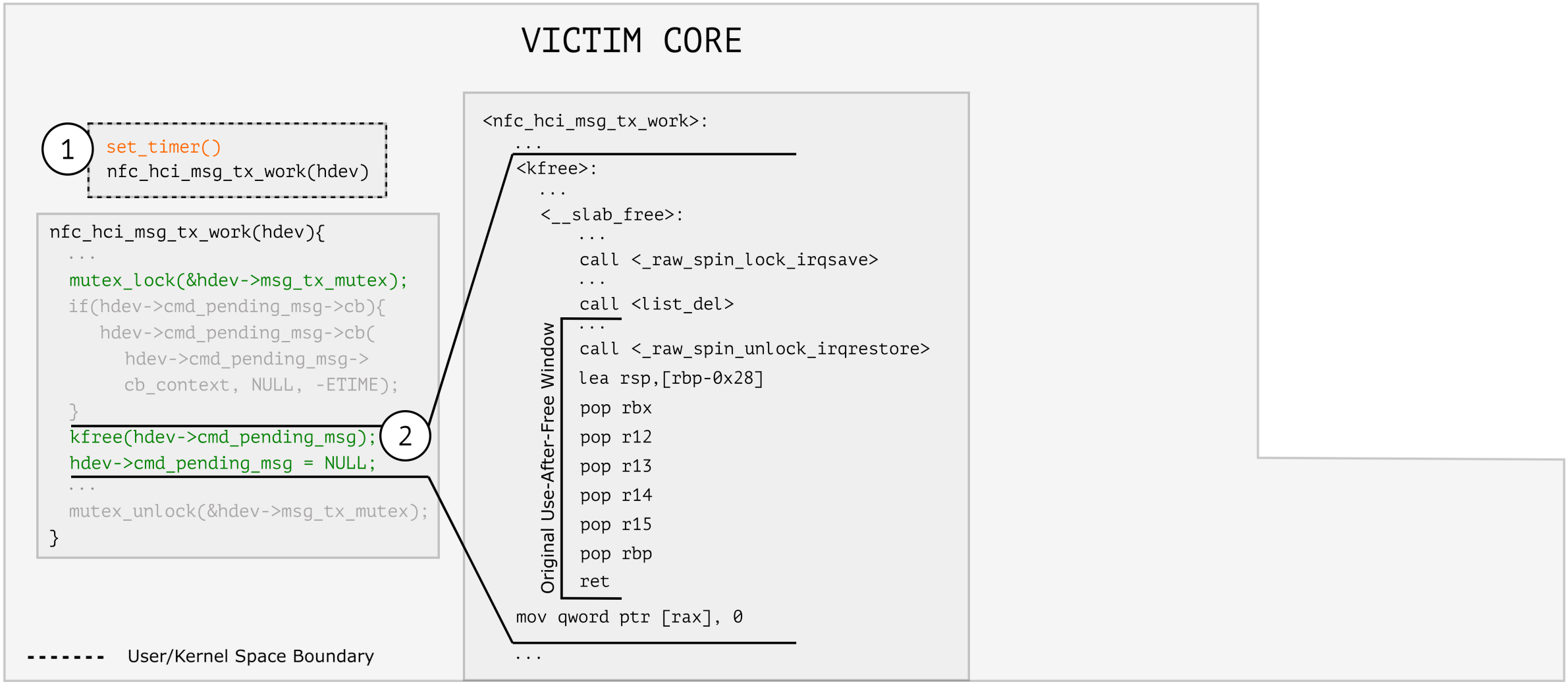
2

------- User/Kernel Space Boundary

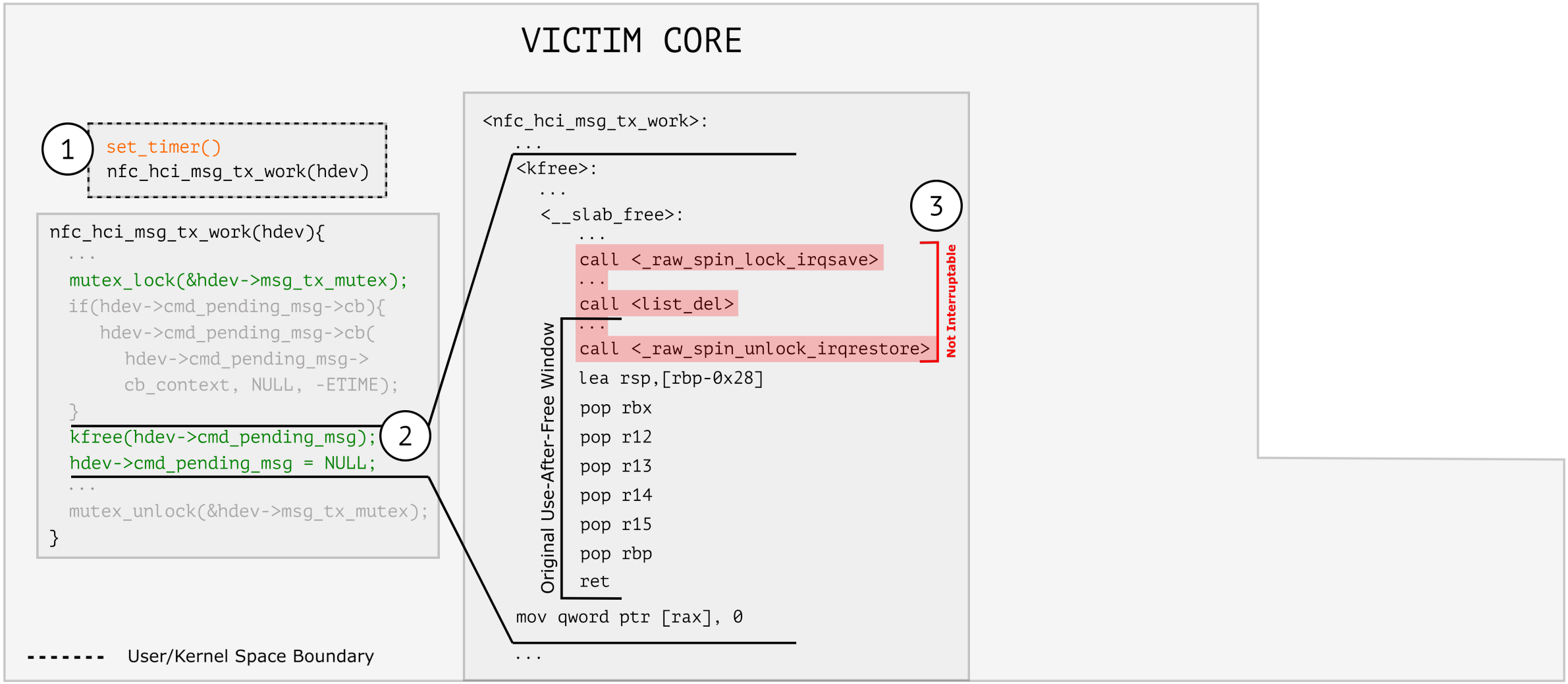
Creating UAF Exploitation Window



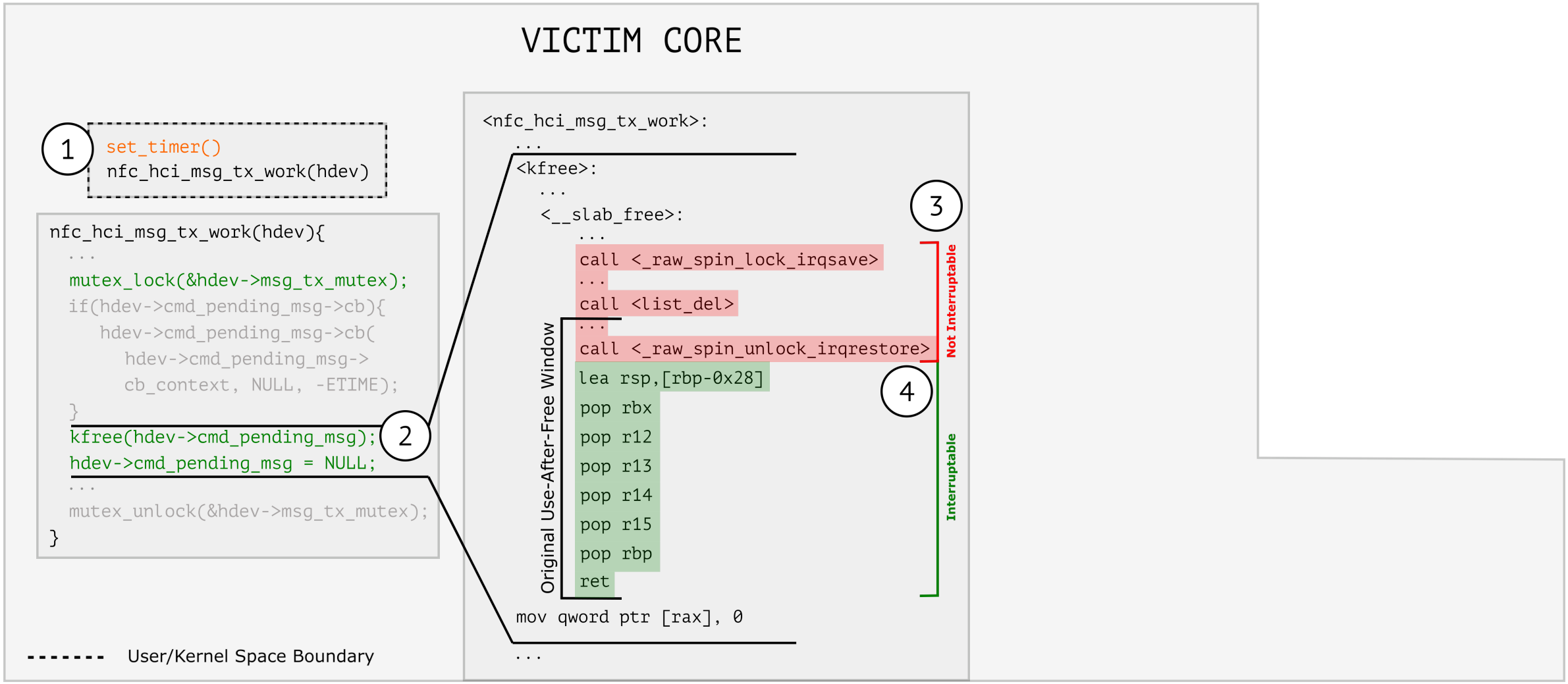
Creating UAF Exploitation Window



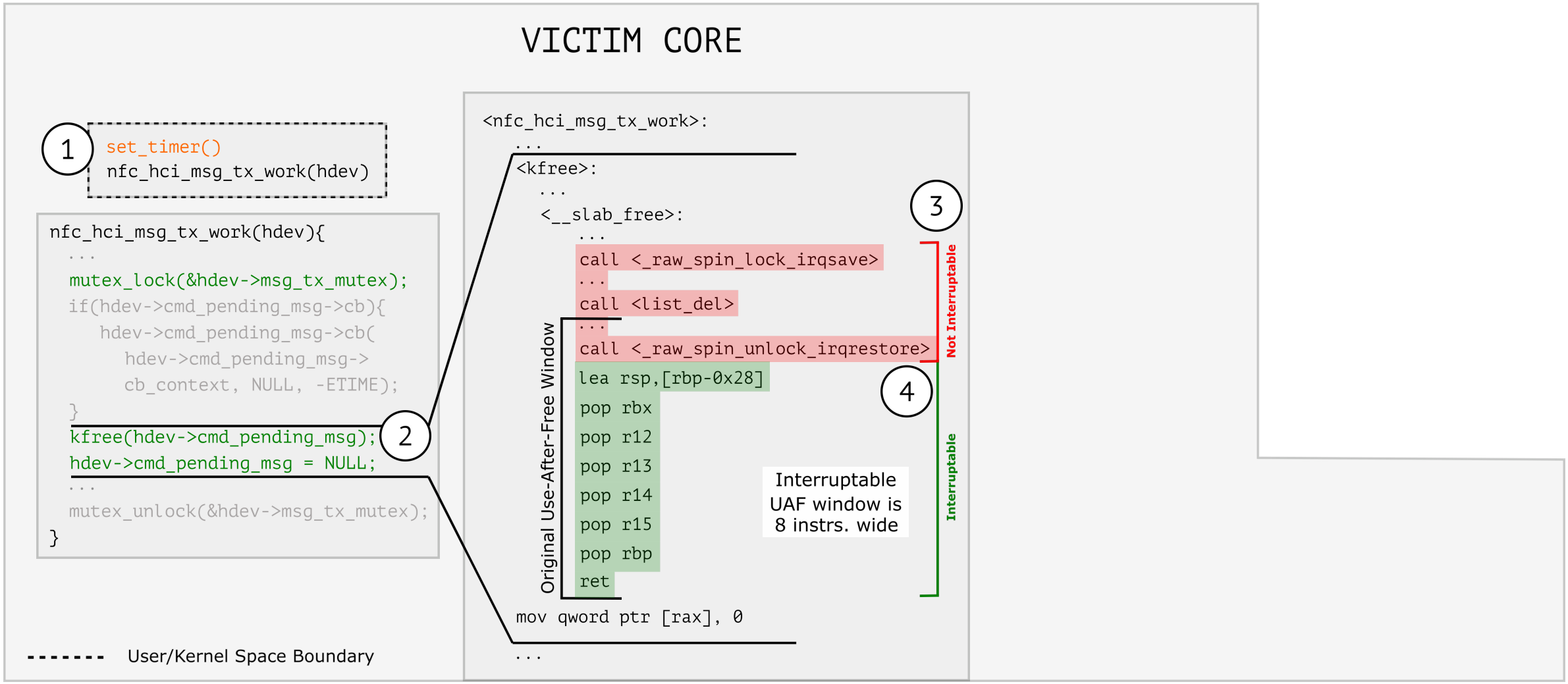
Creating UAF Exploitation Window



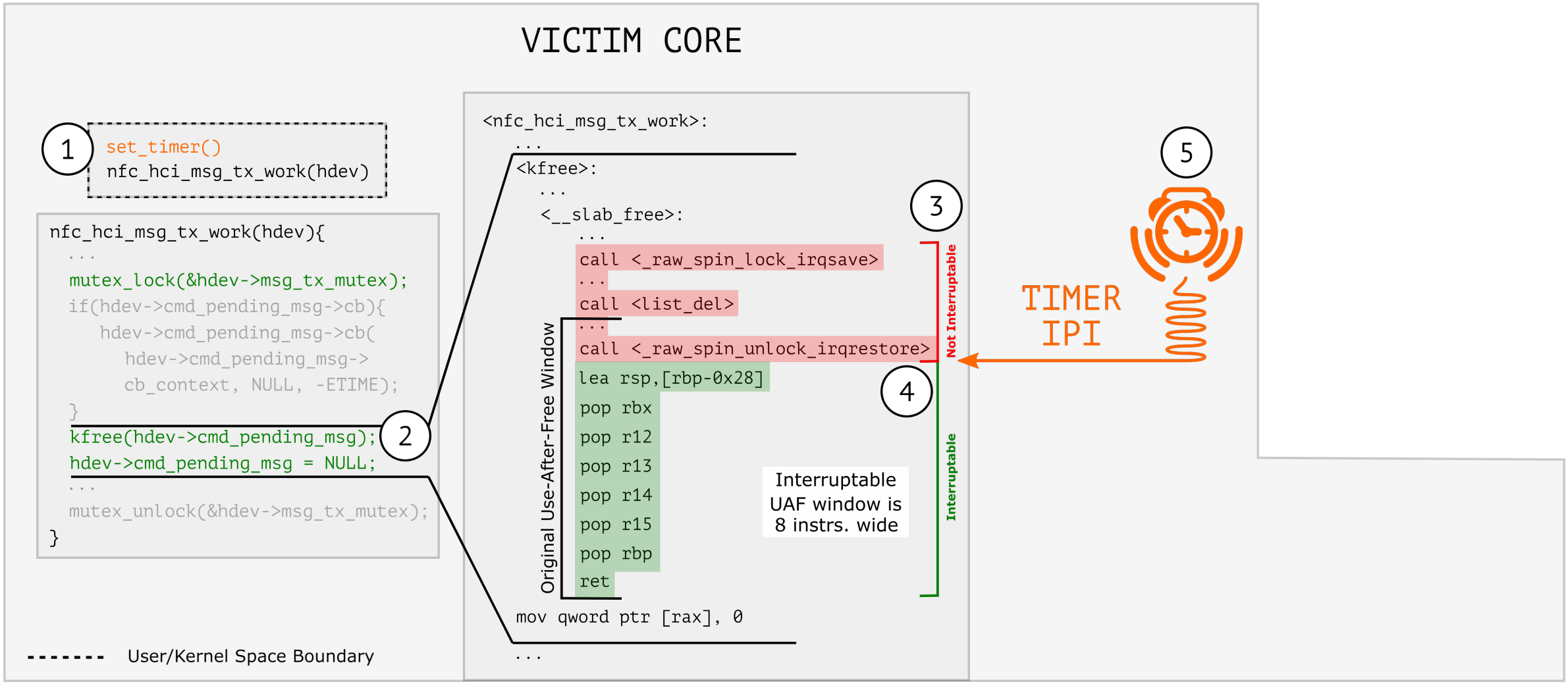
Creating UAF Exploitation Window



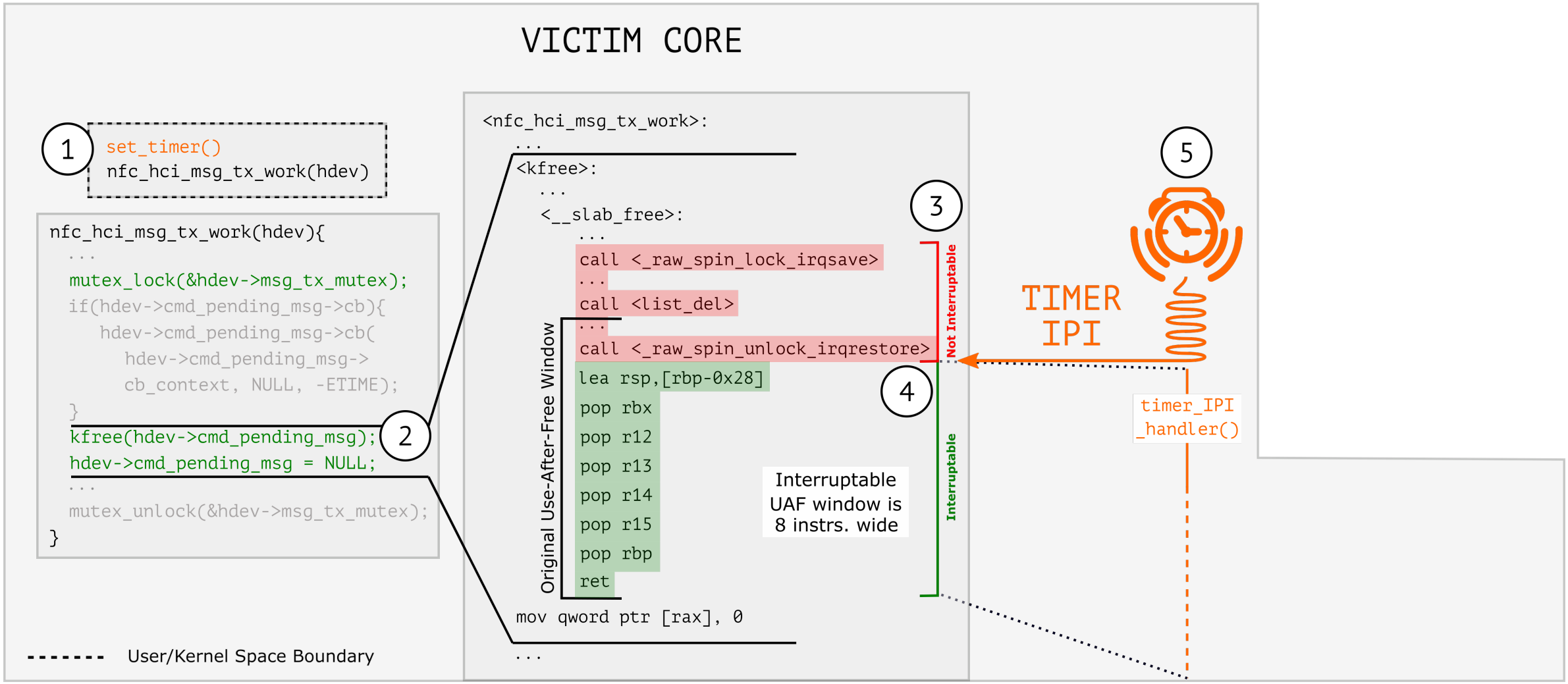
Creating UAF Exploitation Window



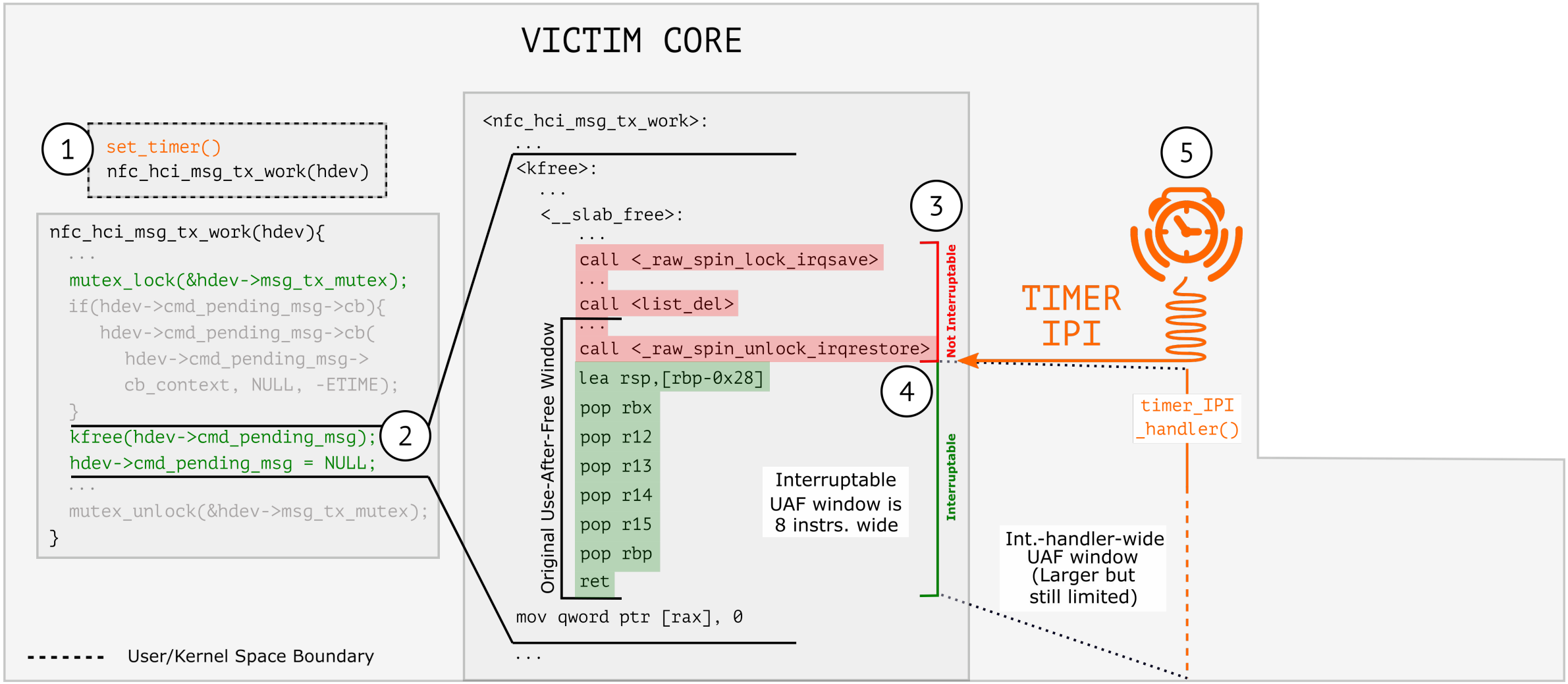
Creating UAF Exploitation Window



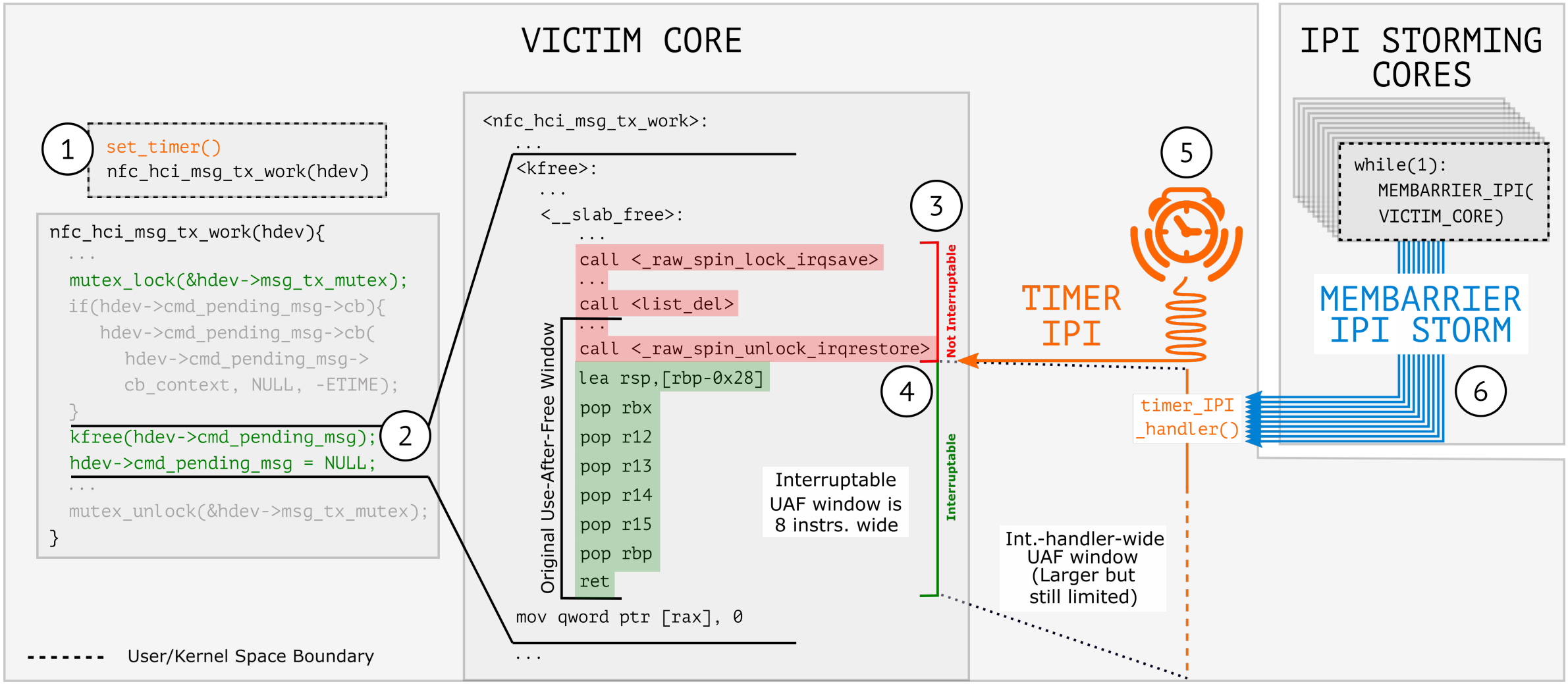
Creating UAF Exploitation Window



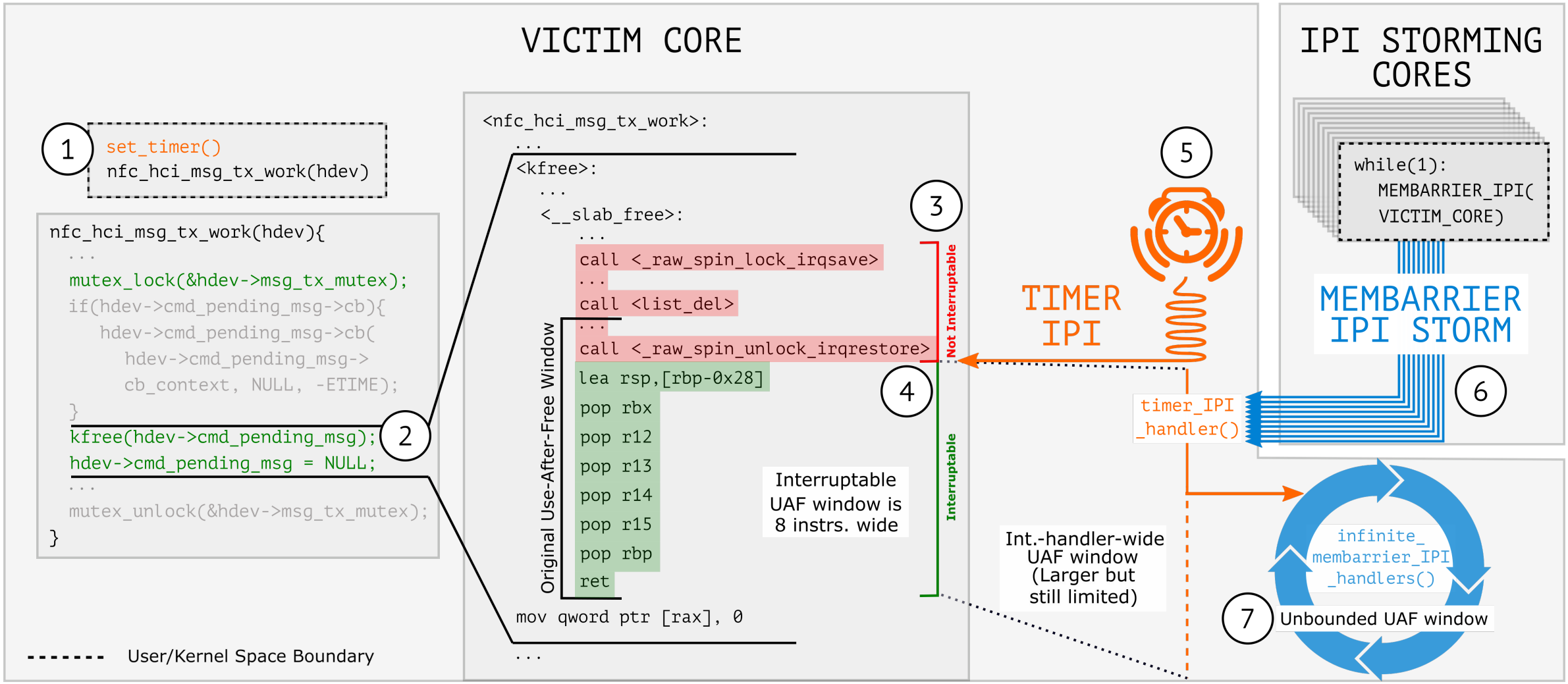
Creating UAF Exploitation Window



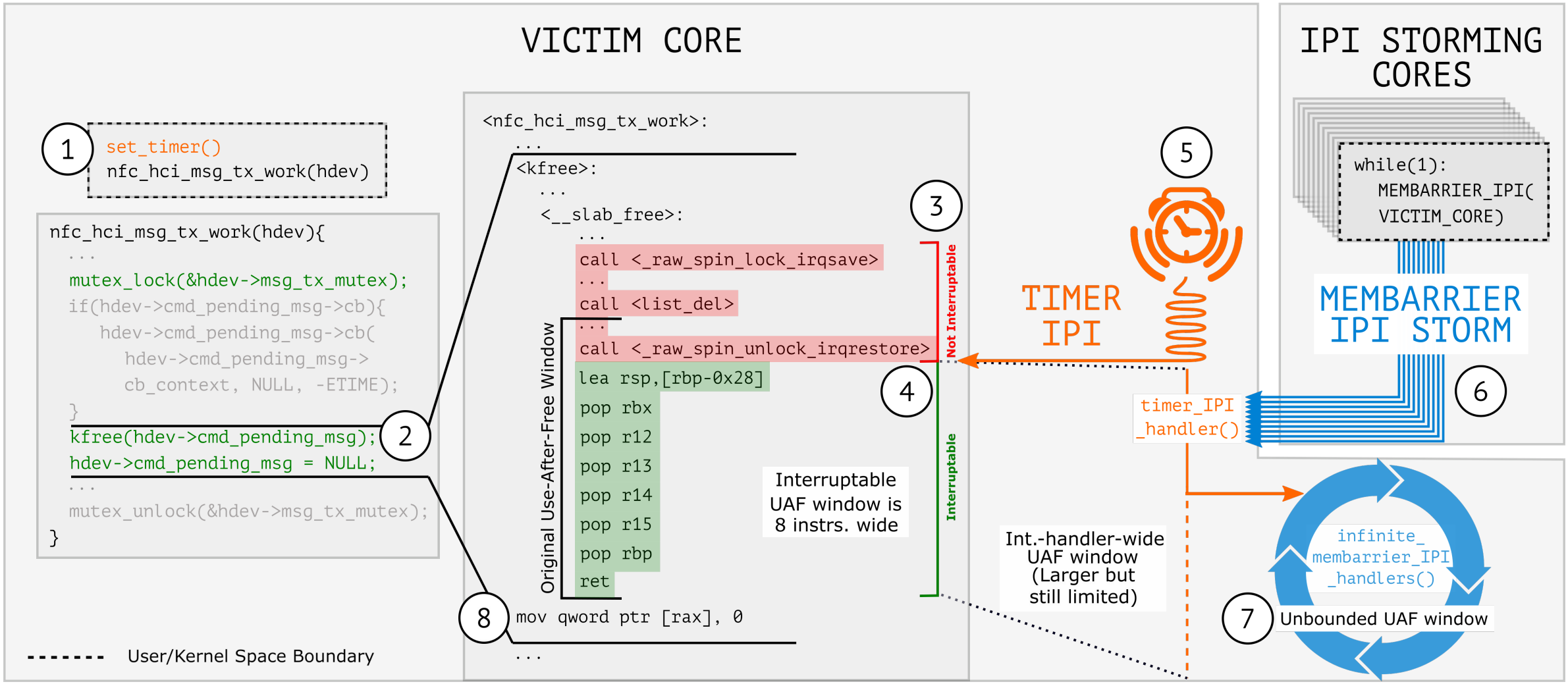
Creating UAF Exploitation Window



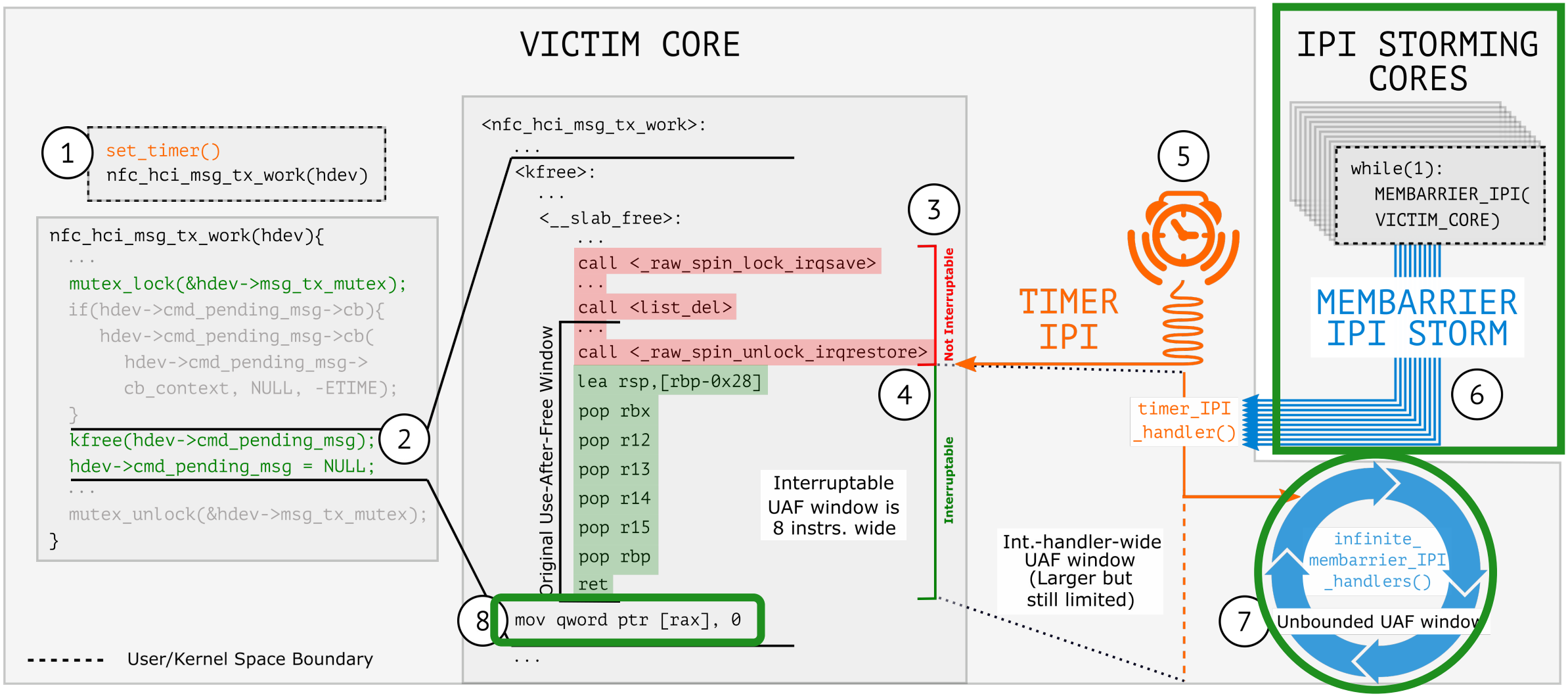
Creating UAF Exploitation Window



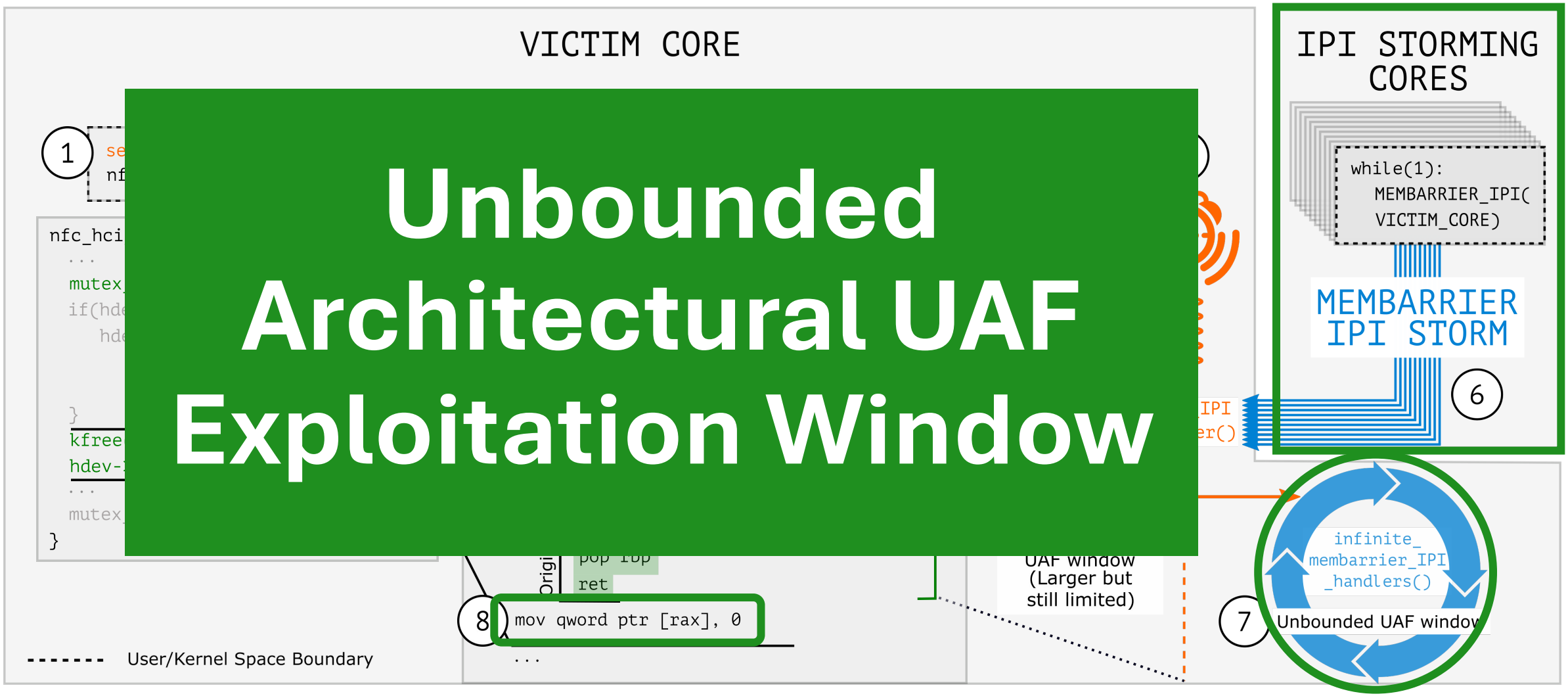
Creating UAF Exploitation Window



Creating UAF Exploitation Window



Creating UAF Exploitation Window



UAF Attack Challenges

1. Bypass The Sync. Primitive:
Speculative Race Condition

2. Create an Exploitation Window:
IPI Storming

SHARED DATA

```
struct nfc_hci_dev {
   ...
   struct mutex msg_tx_mutex;
   ...
   struct hci_msg* cmd_pending_msg;
   ...
}
```

```
struct hci_
   ...
   void (*cb
   void *cb_
   ...
}
```

THREAD 1

```
i_msg_tx_work(hdev){

x_lock(&hdev->msg_tx_mutex);
dev->cmd_pending_msg->cb){
dev->cmd_pending_msg->cb(
  hdev->cmd_pending_msg->
  cb_context, NULL, -ETIME);
 }
 kfree(hdev->cmd_pending_msg);
 hdev->cmd_pending_msg = NULL;
 ...
 mutex_unlock(&hdev->msg_tx_mutex);
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
 ...
 mutex_lock(&hdev->msg_tx_mutex);
 if(hdev->cmd_pending_msg->cb){
    hdev->cmd_pending_msg->cb(
      hdev->cmd_pending_msg->
      cb_context, NULL, -ETIME);
 }
 kfree(hdev->cmd_pending_msg);
 hdev->cmd_pending_msg = NULL;
 ...
 mutex_unlock(&hdev->msg_tx_mutex);
}
```

UAF Attack Challenges

1. Bypass The Sync. Primitive:
Speculative Race Condition

3. Reallocate The Freed Memory:
msgsnd syscall

2. Create an Exploitation Window:
IPI Storming

4. Hijack The Control-Flow:
Speculatively Execute A Controlled Disclosure Gadget

```
struct nfc_
   ...
   struct m
   ...
   struct h
   ...
}
```

```
t hci_
id (*cb
id *cb_
```

THREAD 1

```
i_msg_tx_work(hdev){

x_lock(&hdev->msg_tx_mutex);
dev->cmd_pending_msg->cb){
dev->cmd_pending_msg->cb(
  hdev->cmd_pending_msg->
  cb_context, NULL, -ETIME);
}
kfree(hdev->cmd_pending_msg
hdev->cmd_pending_msg = NUL
...
mutex_unlock(&hdev->msg_tx_
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
...
mutex_lock(&hdev->msg_tx_mutex);
if(hdev->cmd_pending_msg->cb){
   hdev->cmd_pending_msg->cb(
     hdev->cmd_pending_msg->
     cb_context, NULL, -ETIME);
```

Speculative Concurrent Use-After-Free Attack

1. Bypass The Sync. Primitive:
Speculative Race Condition

3. Reallocate The Freed Memory:
msgsnd syscall

2. Create an Exploitation Window:
IPI Storming

4. Hijack The Control-Flow:
Speculatively Execute A Controlled Disclosure Gadget

```
struct nfc_
   ...
   struct m
   ...
   struct h
   ...
}
```

```
t hci_
id (*cb
id *cb_
```

THREAD 1

```
i_msg_tx_work(hdev){
...
x_lock(&hdev->msg_tx_mutex);
dev->cmd_pending_msg->cb){
dev->cmd_pending_msg->cb(
  hdev->cmd_pending_msg->
  cb_context, NULL, -ETIME);
}
kfree(hdev->cmd_pending_msg
hdev->cmd_pending_msg = NUL
...
mutex_unlock(&hdev->msg_tx_
}
```

THREAD 2

```
nfc_hci_msg_tx_work(hdev){
...
mutex_lock(&hdev->msg_tx_mutex);
if(hdev->cmd_pending_msg->cb){
  hdev->cmd_pending_msg->cb(
    hdev->cmd_pending_msg->
    cb_context, NULL, -ETIME);
```

Linux Kernel Attack Surface

Linux Kernel Attack Surface

- Gadget scanner using the Coccinelle patching engine

Linux Kernel Attack Surface

- Gadget scanner using the Coccinelle patching engine
- Statically analyzed Linux v5.15.83

Linux Kernel Attack Surface

- Gadget scanner using the Coccinelle patching engine
- Statically analyzed Linux v5.15.83

Identified 1283 Speculative Concurrent Use-After-Free Gadgets

Mitigating Speculative Race Conditions

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
        ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
            ↳arch_atomic_try_cmpxchg(&lock, , ...)
              ↳arch_try_cmpxchg((&lock, , ...)
                ↳__raw_try_cmpxchg(ptr, ...)({
                    asm volatile(
                        "lock cmpxchgq %2, %1"
                        : "=a" (ret), "+m" (*ptr)
                        : "r" (new), "0" (old)
                        : "memory"
                        );
                })
      return true;
  ...
}
```

Mitigating Speculative Race Conditions

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
        ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
            ↳arch_atomic_try_cmpxchg(&lock, , ...)
              ↳arch_try_cmpxchg((&lock, , ...)
                ↳__raw_try_cmpxchg(ptr, ...)({
                    asm volatile(
                        "lock cmpxchgq %2, %1"
                        : "=a" (ret), "+m" (*ptr)
                        : "r" (new), "0" (old)
                        : "memory"
                        );
                })
      return true;
  ...
}
```

Mitigating Speculative Race Conditions

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
        ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
            ↳arch_atomic_try_cmpxchg(&lock, , ...)
              ↳arch_try_cmpxchg((&lock, , ...)
                ↳__raw_try_cmpxchg(ptr, ...)({
                    asm volatile(
                        "lock cmpxchgq %2, %1"

                            lfence

                })
      return true;
  ...
}
```

Mitigating Speculative Race Conditions

LMBench:
Geomean
≈5%

```
void mutex_lock(struct mutex *lock){
  ...
  if (!__mutex_trylock_fast(lock))
    if (atomic_long_try_cmpxchg_acquire(&lock, ...))
        ↳arch_atomic_long_try_cmpxchg_acquire(&lock, , ...)
          ↳arch_atomic_try_cmpxchg_acquire(&lock, , ...)
            ↳arch_atomic_try_cmpxchg(&lock, , ...)
              ↳arch_try_cmpxchg((&lock, , ...)
                ↳__raw_try_cmpxchg(ptr, ...)({
                    asm volatile(
                        "lock cmpxchgq %2, %1"

                            lfence

                })
      return true;
  ...
}
```

GhostRace: Exploiting and Mitigating Speculative Race Conditions

GhostRace: Exploiting and Mitigating Speculative Race Conditions

- Speculative Race Condition

GhostRace: Exploiting and Mitigating Speculative Race Conditions

- Speculative Race Condition

- Inter-Process Interrupt Storming

GhostRace: Exploiting and Mitigating Speculative Race Conditions

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)

GhostRace: Exploiting and Mitigating Speculative Race Conditions

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)
- PoC Exploit (12KB/s)




GhostRace: Exploiting and Mitigating Speculative Race Conditions

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)
- PoC Exploit (12KB/s)
- Mitigation (5%)





GhostRace: Exploiting and Mitigating Speculative Race Conditions

- Speculative Race Condition
- Inter-Process Interrupt Storming
- Gadget Scanner (1200+)
- PoC Exploit (12KB/s)
- Mitigation (5%)





VUSec

IBM Research | Zurich