KasIr 内核地址空间布局随机化

目录

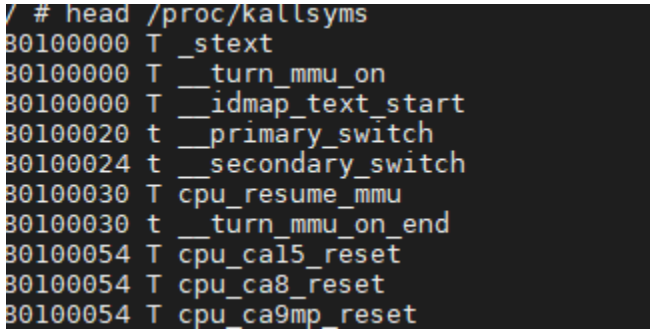
CONTENT

OpenEuler

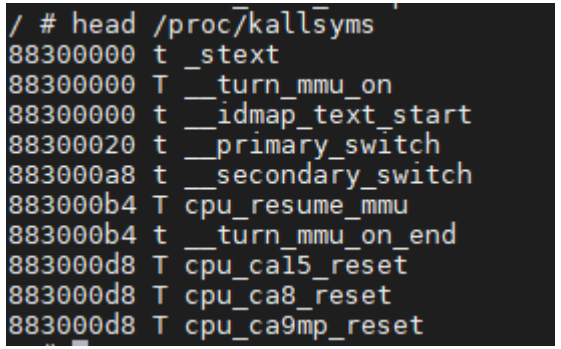
Kaslr介绍

KASLR(kernel address space layout randomization)即内核地址空间布局随机化，KASLR技术在kernel image加载到内存时对其进行偏移和重定位。

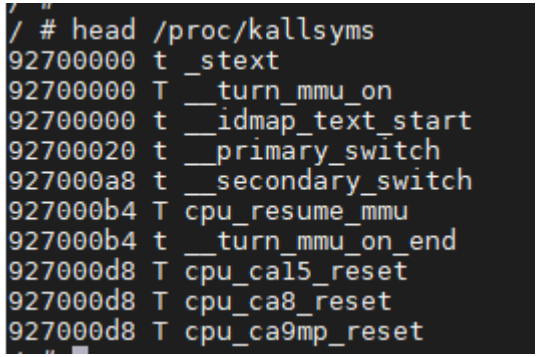
当KASLR关闭的时候，kernel image都会映射到一个固定的链接地址，这对于黑客来说是透明的，因此安全性得不到保证。KASLR技术可以让kernel image映射的地址相对于链接地址有个偏移,偏移地址可以通过dts设置,如果bootloader支持每次开机随机生成偏移数值，那么可以做到每次开机kernel image映射的虚拟地址都不一样,因此，对于开启KASLR的kernel来说，不同的产品的kernel image映射的地址几乎都不一样,因此在安全性上有一定的提升。



nokaslr



kaslr-1



kaslr-2

https://lwn.net/Articles/569635/

OpenEuler

Linux内存地址管理

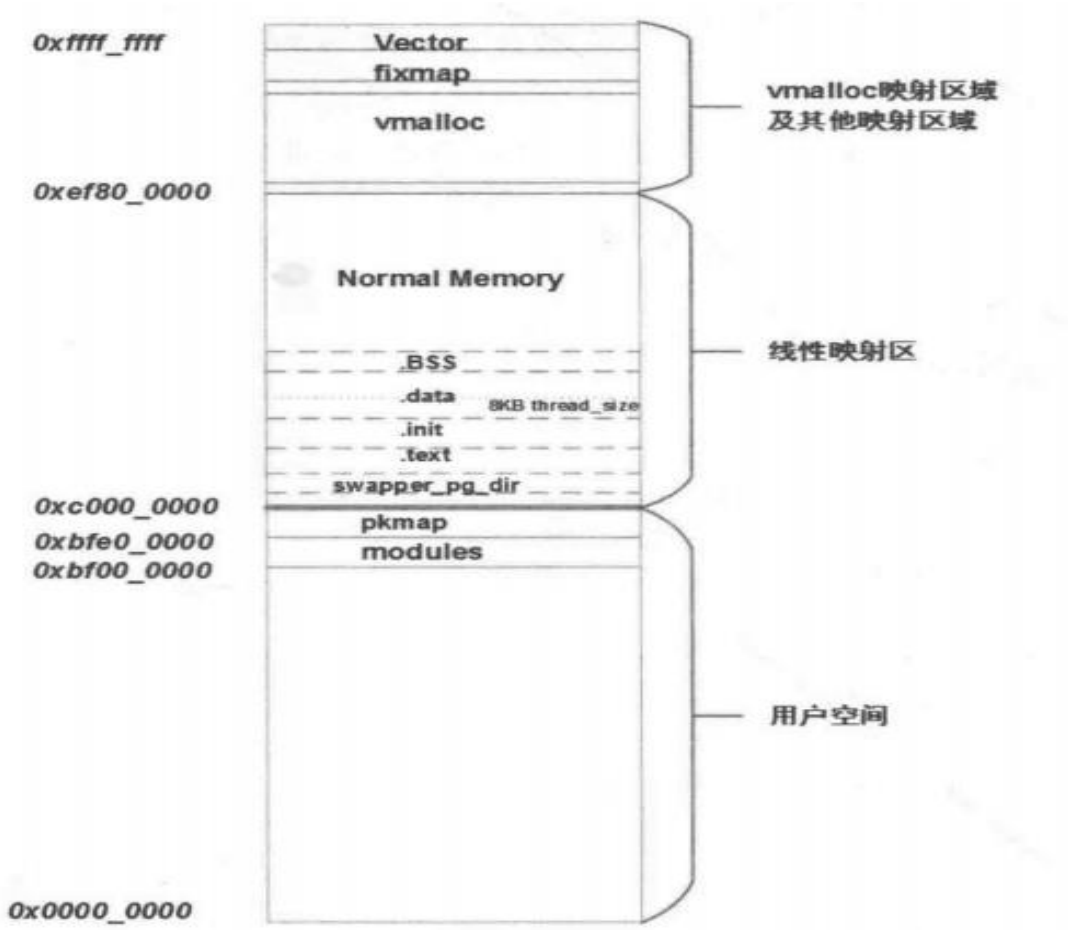


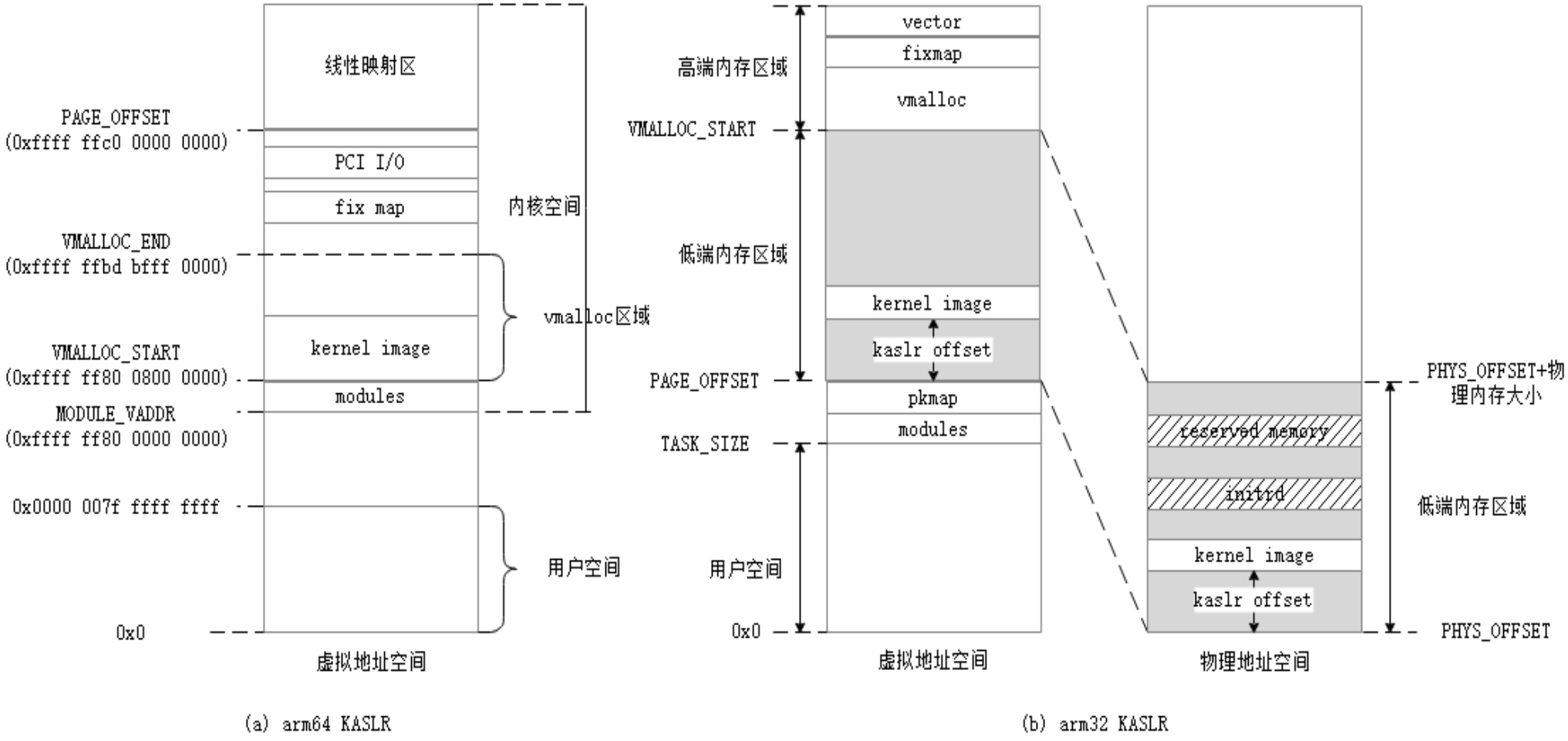
图2.6　ARM32内核内存布局图

内核地址空间有高端内存区域（240MB）和低端内存区域（760MB）的概念，高端内存区域指非线性映射区，其存在的意义主要是进行虚拟地址向物理地址的动态映射和动态扩展，即上图中的vmalloc区域，低端内存区域为线性映射区，其映射原理时将物理地址与虚拟地址进行1：1映射，且地址偏移确定以后，线性映射区的映射关系就不会再发生改变（所以在汇编代码里面，通常就是通过物理地址与虚拟地址的偏移值确定线性区的地址映射关系）

图片来源于网络：https://www.cnblogs.com/linhaostudy/p/12857407.html

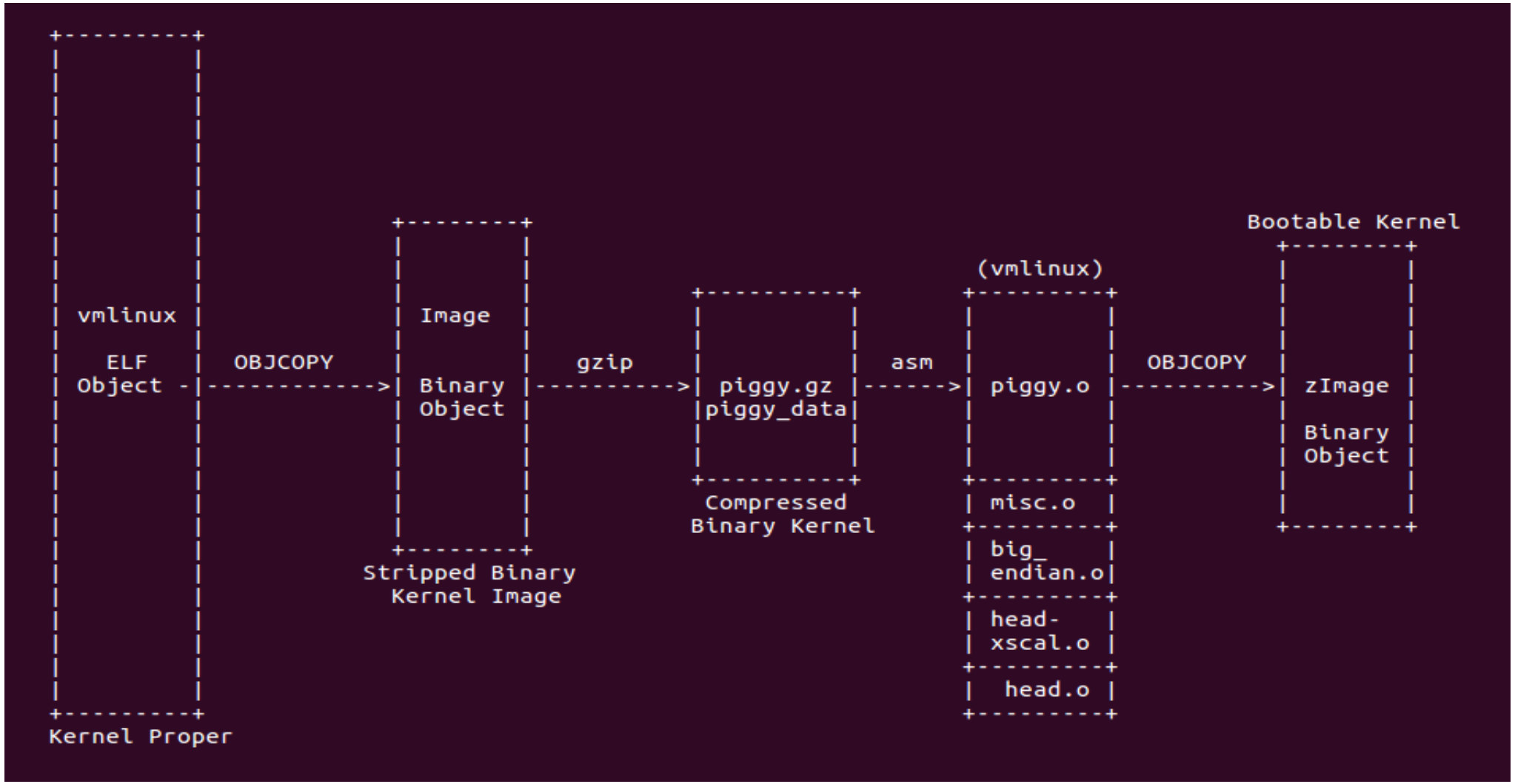
OpenEuler

Arm kaslr与内存的关联



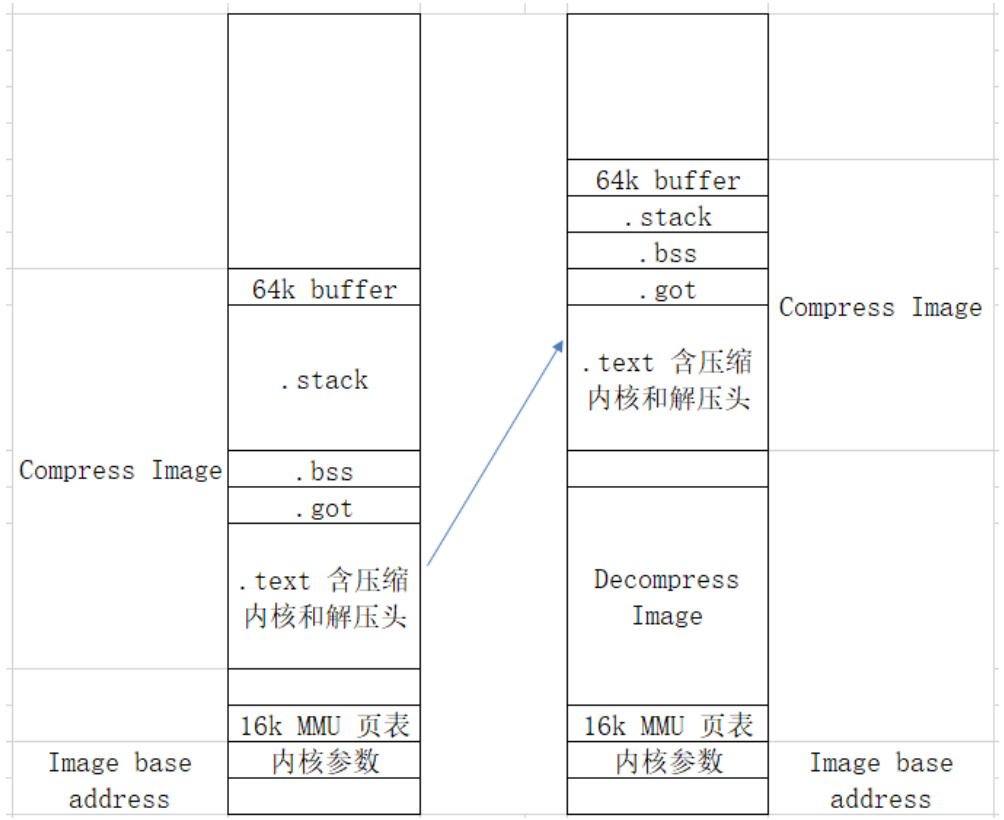
OpenEuler

Linux内核编译构建

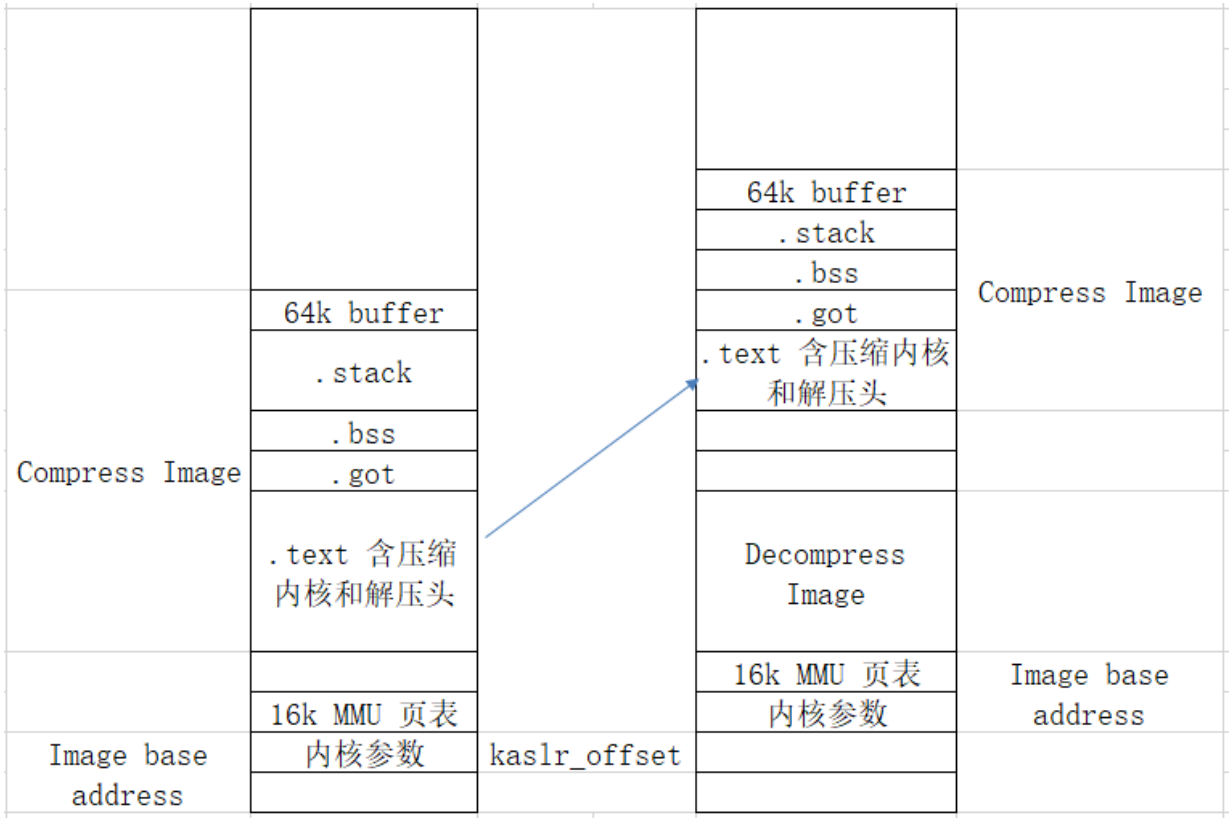


OpenEuler

Linux内核解压过程



no-kaslr



kaslr

OpenEuler

Kaslr实现 –arm架构

Kaslr内核构建阶段：

1. 链接选项中添加-fPIC，-PIE参数，内核编译成地址无关镜像
2. 生成重定位段.rel*,用于内核符号重定位

Kaslr解压缩阶段：

1. kaslr_offset伪随机偏移值计算，解析cmdline，解析dts
2. 镜像解压到指定内存，解压、移动镜像

Kaslr系统启动阶段：

1. kaslr_offset记录内核偏移，内核符号地址重定位

Image内核编译

```
--- a/arch/arm/Makefile
+++ b/arch/arm/Makefile
@@ -48,6 +48,11 @@ CHECKFLAGS    += -D__ARMEL__
 KBUILD_LDFLAGS += -EL
 endif

+ifeq ($(CONFIG_RELOCATABLE),y)
+KBUILD_CFLAGS          += -fpic -include $(srctree)/include/linux/hidden.h
+LDFLAGS_vmlinux                += -pie -shared -Bsymbolic
+endif
+
```

```
--- a/arch/arm/kernel/vmlinux.lds.S
+++ b/arch/arm/kernel/vmlinux.lds.S
@@ -115,6 +115,12 @@ SECTIONS
                __smpalt_end = .;
        }
 #endif
+       .rel.dyn : ALIGN(8) {
+               __rel_begin = .;
+               *(.rel .rel.* .rel.dyn)
+       }
+       __rel_end = ADDR(.rel.dyn) + SIZEOF(.rel.dyn);
+
```

```
Section to Segment mapping:
 Segment Sections...
  00     .head.text
  01     .text
  02     .rodata __ksymtab __ksymtab_gpl __ksymtab_strings __param __modver .notes __ex_table .ARM.unwind_idx .ARM.unwind_tab
  03     .vectors
  04     .stubs
  05     .init.text .exit.text .init.arch.info .init.tagtable .init.smpalt .init.pv_table .init.data .data..percpu
  06     .data __bug_table .bss
  07     .notes
```

```
Section to Segment mapping:
 Segment Sections...
  00     .head.text
  01     .text
  02     .rodata __ksymtab __ksymtab_gpl __ksymtab_strings __param __modver .notes __ex_table .ARM.unwind_idx .ARM.unwind_tab
  03     .vectors
  04     .stubs
  05     .init.text .exit.text .init.arch.info .init.tagtable .init.smpalt .rel.dyn .init.pv_table .init.data .data..percpu
  06     .data __bug_table .data.rel.local .bss
  07     .notes
```

重定义页表目录

```
--- a/arch/arm/kernel/head.S
+++ b/arch/arm/kernel/head.S
@@ -45,14 +45,6 @@
 #define PMD_ORDER      2
 #endif

-       .globl  swapper_pg_dir
-       .equ    swapper_pg_dir, KERNEL_RAM_VADDR - PG_DIR_SIZE
-
-       .macro  pgtbl, rd, phys
-       add     \rd, \phys, #TEXT_OFFSET
-       sub     \rd, \rd, #PG_DIR_SIZE
-       .endm
-
 /*
  * Kernel startup entry point.
  * ---------------------------
@@ -74,6 +66,9 @@
        .arm

        __HEAD
+       .globl  swapper_pg_dir
+       .equ    swapper_pg_dir, . - PG_DIR_SIZE
+
```

OpenEuler

关键随机流程

```
+#ifdef CONFIG_RANDOMIZE_BASE
+                ldr     r1, __kaslr_offset      @ check if the kaslr_offset is
+                cmp     r1, #0                  @ already set
+                bne     1f
+
+                stmfd   sp!, {r0-r3, ip, lr}
+                adr_l   r2, _text               @ start of zImage
+                stmfd   sp!, {r2, r8, r10}      @ pass stack arguments
+
+                ldr_l   r3, __kaslr_seed
+#if defined(CONFIG_CPU_V6) || defined(CONFIG_CPU_V6K) || defined(CONFIG_CPU_V7)
+                /*
+                 * Get some pseudo-entropy from the low bits of the generic
+                 * timer if it is implemented.
+                 */
+                mrc     p15, 0, r1, c0, c1, 1   @ read ID_PFR1 register
+                tst     r1, #0x10000            @ have generic timer?
+                mrrcne  p15, 1, r3, r1, c14     @ read CNTVCT
+#endif
+                adr_l   r0, __kaslr_offset      @ pass &__kaslr_offset in r0
+                mov     r1, r4                  @ pass base address
+                mov     r2, r9                  @ pass decompressed image size
+                eor     r3, r3, r3, ror #16     @ pass pseudorandom seed
+                bl      kaslr_early_init
+                add     sp, sp, #12
+                cmp     r0, #0
+                addne   r4, r4, r0              @ add offset to base address
+                ldmfd   sp!, {r0-r3, ip, lr}
+                bne     restart
+1:
+#endif
```

OpenEuler

随机数的生成 kaslr_early_init

```
+u32 kaslr_early_init(u32 *kaslr_offset, u32 image_base, u32 image_size,
+                     u32 seed, u32 zimage_start, const void *fdt,
+                     u32 zimage_end)
+{
+       static const char __aligned(4) build_id[] = UTS_VERSION UTS_RELEASE;
+       u32 bitmap[(VMALLOC_END - PAGE_OFFSET) / SZ_2M / 32] = {};
```

```
chosen = fdt_path_offset(fdt, "/chosen");
if (chosen < 0)
        return 0;

command_line = fdt_getprop(fdt, chosen, "bootargs", &len);

/* check the command line for the presence of 'nokaslr' */
p = get_cmdline_param(command_line, "nokaslr", sizeof("nokaslr") - 1);
if (p != NULL)
        return 0;

/* check the command line for the presence of 'vmalloc=' */
p = get_cmdline_param(command_line, "vmalloc=", sizeof("vmalloc=") - 1);
if (p != NULL)
        lowmem_top = VMALLOC_END - __memparse(p + 8, NULL) -
                     VMALLOC_OFFSET;
else
        lowmem_top = VMALLOC_DEFAULT_BASE;
```

OpenEuler

随机数的生成 kaslr_early_init

```
+	/* check for a reserved-memory node and record its cell sizes */
+	regions.reserved_mem = fdt_path_offset(fdt, "/reserved-memory");
+	if (regions.reserved_mem >= 0)
+		get_cell_sizes(fdt, regions.reserved_mem,
+			       &regions.reserved_mem_addr_cells,
+			       &regions.reserved_mem_size_cells);
+
+	/*
+	 * Iterate over the physical memory range covered by the lowmem region
+	 * in 2 MB increments, and count each offset at which we don't overlap
+	 * with any of the reserved regions for the zImage itself, the DTB,
+	 * the initrd and any regions described as reserved in the device tree.
+	 * If the region does overlap, set the respective bit in the bitmap[].
+	 * Using this random value, we go over the bitmap and count zero bits
+	 * until we counted enough iterations, and return the offset we ended
+	 * up at.
+	 */
+	count = count_suitable_regions(fdt, &regions, bitmap);
+	puthex32(count);
+
+	num = ((u16)seed * count) >> 16;
+	puthex32(num);
+
+	*kaslr_offset = get_region_number(num, bitmap) * SZ_2M;
+	puthex32(*kaslr_offset);
+
+	return *kaslr_offset;
+}
```

OpenEuler

随机数的生成 count_suitable_regions

```
145 static bool intersects_occupied_region(const void *fdt, u32 start,
146                                         u32 end, struct regions *regions)
147 {
148         if (regions_intersect(start, end, regions->zimage_start,
149                               regions->zimage_start + regions->zimage_size))
150                 return true;
151
152         if (regions_intersect(start, end, regions->initrd_start,
153                               regions->initrd_start + regions->initrd_size))
154                 return true;
155
156         if (regions_intersect(start, end, regions->dtb_start,
157                               regions->dtb_start + regions->dtb_size))
158                 return true;
159
160         return intersects_reserved_region(fdt, start, end, regions);
161 }
162
163 static u32 count_suitable_regions(const void *fdt, struct regions *regions,
164                                   u32 *bitmap)
165 {
166         u32 pa, i = 0, ret = 0;
167
168         for (pa = regions->pa_start; pa < regions->pa_end; pa += SZ_2M, i++) {
169                 if (!intersects_occupied_region(fdt, pa,
170                                                 pa + regions->image_size,
171                                                 regions)) {
172                         ret++;
173                 } else {
174                         /* set 'occupied' bit */
175                         bitmap[i >> 5] |= BIT(i & 0x1f);
176                 }
177         }
178         return ret;
179 }
```

OpenEuler

跳转内核代码入口

```
 /*
  * Check to see if we will overwrite ourselves.
  *   r4  = final kernel address (possibly with LSB set)
@@ -1415,10 +1467,46 @@ __enter_kernel:
                mov     r0, #0                  @ must be 0
                mov     r1, r7                  @ restore architecture number
                mov     r2, r8                  @ restore atags pointer
+#ifdef CONFIG_RANDOMIZE_BASE
+               ldr     r3, __kaslr_offset
+               add     r4, r4, #4              @ skip first instruction
+#endif
  ARM(          mov     pc, r4          )       @ call kernel
  M_CLASS(      add     r4, r4, #1      )       @ enter in Thumb mode for M class
  THUMB(                bx      r4              )       @ entry point is always ARM for A/R classes

+#ifdef CONFIG_RANDOMIZE_BASE
```

OpenEuler

符号重定位

```
+__primary_switch:
+#ifdef CONFIG_RELOCATABLE
+       adr_l   r7, _text                       @ r7 := __pa(_text)
+       sub     r7, r7, #TEXT_OFFSET            @ r7 := PHYS_OFFSET
+
+       adr_l   r5, __rel_begin
+       adr_l   r6, __rel_end
+       sub     r5, r5, r7
+       sub     r6, r6, r7
+
+       add     r5, r5, #PAGE_OFFSET
+       add     r6, r6, #PAGE_OFFSET
+       add     r5, r5, r12
+       add     r6, r6, r12
+
+       adr_l   r3, __stubs_start               @ __pa(__stubs_start)
+       sub     r3, r3, r7                      @ offset of __stubs_start
+       add     r3, r3, #PAGE_OFFSET            @ __va(__stubs_start)
+       sub     r3, r3, #0xffff1000             @ subtract VA of stubs section
+
+0:     cmp     r5, r6
+       bge     1f
+       ldm     r5!, {r7, r8}                   @ load next relocation entry
+       cmp     r8, #23                         @ R_ARM_RELATIVE
+       bne     0b
+       cmp     r7, #0xff000000                 @ vector page?
+       addgt   r7, r7, r3                      @ fix up VA offset
+       ldr     r8, [r7, r12]
+       add     r8, r8, r12
+       str     r8, [r7, r12]
+       b       0b
+1:
+#endif
+       ldr     pc, =__mmap_switched
+ENDPROC(__primary_switch)
+
```

static int val;

static int *ptr = &val;

比如上面这个例子，ptr是一个指向val变量的指针。val相对于二进制的偏移是B，也就是ptr的值是B。但是当内核装载到了A，那么p的值就应该被修正为A + B。先从offset指向的内容中读出A，然后加上B，写回到offset指向的地址中。

OpenEuler

Kaslr特性使用

KASLR 使能条件：

1. Kconfig控制：CONFIG_RANDOMIZE_BASE
2. comline配置：不能存在nokaslr参数

KASLR 随机种子：

1. 基于DTB文件做CRC运算，生成随机种子
2. 通过dts文件指定随机种子，如下：

```
/ {
    chosen {
        kaslr-seed = <0x10000000>;
    };
};
```

KASLR 生效确认：

echo 0 > /proc/sys/kernel/kptr_restrict

cat /proc/kallsyms |grep purge_vmap_area_lazy

Kaslr调试-打印内存布局（vexpress-a9）

make ARCH=arm CROSS_COMPILE=armeb-eabi- vexpress_defconfig

make ARCH=arm CROSS_COMPILE=armeb-eabi- menuconfig

配置Kconfig，配置如下：

```
[*] Kernel low-level debugging functions (read help!)
      Kernel low-level debugging port (Use PL011 UART0 at 0x10009000 (V2P-CA9 core tile))  --->
```

```
                Kernel low-level debugging port
  Use the arrow keys to navigate this window or press the
  hotkey of the item you wish to select followed by the <SPACE
  BAR>. Press <?> for additional information about this
 ┌──────────────────────────────────────────────────────────┐
 │ ( ) Autodetect UART0 on Versatile Express Cortex-A core til │
 │ (X) Use PL011 UART0 at 0x10009000 (V2P-CA9 core tile)    │
 │ ( ) Use PL011 UART0 at 0x1c090000 (RS1 complaint tiles)  │
 │ ( ) Kernel low-level debugging via EmbeddedICE DCC channel │
 │ ( ) Kernel low-level debug output via semihosting I/O    │
 │ ( ) Kernel low-level debugging via 8250 UART             │
 └    ┴(+)                                                 ┘
                    <Select>      < Help >
```

```
Symbol: DEBUG_UNCOMPRESS [=y]
Type  : bool
Defined at arch/arm/Kconfig.debug:1926
  Prompt: Enable decompressor debugging via DEBUG_LL output
  Depends on: (ARCH_MULTIPLATFORM [=y] || PLAT_SAMSUNG [=n] || ARM_SINGLE_ARMV7M [=n]) && DEBU
  Location:
    -> Kernel hacking
(1)   -> arm Debugging
```

OpenEuler

Kaslr调试-打印内存布局（vexpress-a9）

内存地址布局信息：

```
~/linux_dir/bugfix/hulk-5.10/hulk# qemu-system-arm -M vexpress-a9 -m 512M -s -nographic -kernel arch/arm/boot/zImage -dtb
-append "root=/dev/mmcblk0 rw console=ttyAMA0"
WARNING: Image format was not specified for '/home/linux_dir/debug/rootfs.sd' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
regions.image_size:00e08000
regions.pa_start:60000000
regions.pa_end:7f1f8000
regions.zimage_start:60010000
regions.zimage_size:005199f8
regions.dtb_start:68000000
regions.dtb_size:0000bcd6
regions.initrd_start:00000000
regions.initrd_size:00000000
count:000000ee
num:00000036
*kaslr_offset:08200000
Uncompressing Linux... done, booting the kernel.
```

OpenEuler

Kaslr调试-gdb调试内核

1、Qemu启动内核加上 -S参数，表示qemu启动即暂停，等待gdb连接，如下：

```
qemu-system-arm -M vexpress-a9 -m 1024M -S -s -nographic -kernel zImage -dtb vexpress-v2p-ca9.dtb
-sd /home/rootfs.sd -append "root=/dev/mmcblk0 rw console=ttyAMA0"
```

2、使用gdb的add-symbol-file 动态加载vmlinux 的符号表，根据实际 vmlinux 加载偏移设置偏移量，如果不开kaslr，无需设置地址偏移，例如：

arm-linux-gnueabi-gdb vmlinux 0x60010000

add-symbol-file arch/arm/boot/compressed/vmlinux 0x60010000

3、使用gdb连接内核进程，如下：

(gdb) target remote:1234

4、断点调试，如下：

b BS_debug //假设BS_debug是待调试函数，或者自己添加的debug位置标记

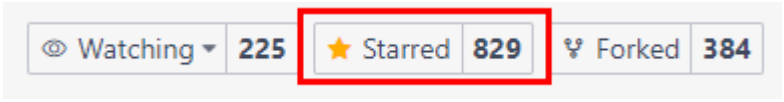
5、enjoy it.

✓ openEuler kernel gitee 仓库

源代码仓库

https://gitee.com/openeuler/kernel

欢迎大家多多 Star，多多参与社区开发，多多贡献补丁。



✓ maillist、issue、bugzilla

可以通过邮件列表、issue、bugzilla 参与社区讨论

欢迎大家多多讨论问题，发现问题多提 issue、bugzilla

https://gitee.com/openeuler/kernel/issues

https://bugzilla.openeuler.org

kernel@openeuler.org

✓ openEuler kernel SIG 微信技术交流群

请扫描右方二维码添加小助手微信

或者直接添加小助手微信（微信号：openeuler-kernel）

备注“交流群”或“技术交流”

加入 openEuler kernel SIG 技术交流群

技术交流

OpenEuler

Thank you

OpenEuler