BI.ZONE

НИЯУ МИФИ

Введение в PWN 0x5

Спикер: Павел Блинников

Руководитель группы исследования уязвимостей BI.ZONE
Капитан SPRUSH
Админ MEPhI CTF

@pturtle

Что мы узнали на прошлых занятиях?

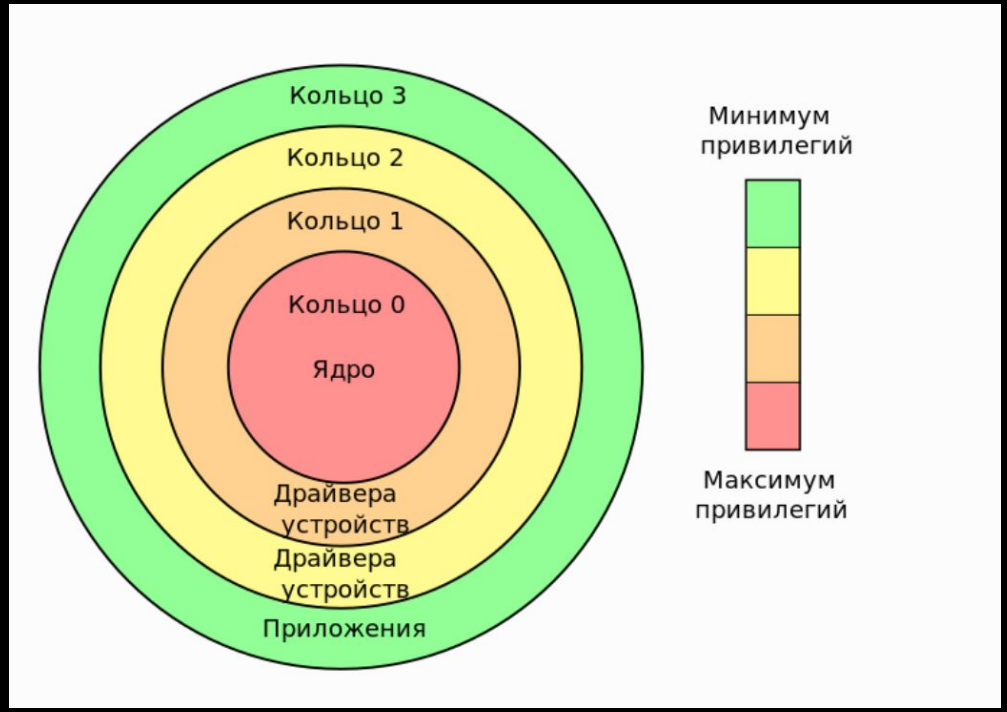
1. Всевозможные аспекты эксплуатации и уязвимости user-space приложений на Linux
2. Отладка приложений
3. Основы Си

Что мы узнаем сегодня?

1. Почему атаковать ядро – это круто
2. Основные примитивы
3. По верхам посмотрим атаки

Ядро ОС

Ядро работает в более привилегированном режиме, чем userspace



Обычный таск на CTF

Таск на ядро

Взаимодействие с ядром

1. Прямой вызов системных вызовов ядра
2. Взаимодействие с девайсами в /dev, /sys и /proc
3. Взаимодействия с ФС, сетью: системные вызовы в модулях ядра (драйверах)

Два основных участка памяти

- kernel base – базовый адрес ядра :)
  при выключенном kaslr обычно равен 0xffffffff81000000
- physmap – прямое отображение физической памяти
  при выключенном kaslr обычно равен 0xffff888000000000

Митигации

- KASLR – ASLR в ядре
- SMEP – запрет на выполнение пользовательского кода из ядра
- SMAP – запрет на обращение к пользовательским страницам памяти из ядра
- KPTI – защита от атак Meltdown/Spectre
- Огромное количество разных митигаций конкретных техник эксплуатации, постоянно вводятся новые

Какие могут быть уязвимости?

Какие могут быть уязвимости?

Все, которые возникают в userspace!

Переполнения буфера, форматные строки, UAF, double free, race condition

Какие могут быть уязвимости?

Все, которые возникают в userspace!

Переполнения буфера, форматные строки, UAF, double free, race condition

Но бывают и специфичные: например TOCTOU (time-of-check to time-of-use)

Ways to win

1. Установка uid в структуре cred у своего процесса равным нулю (arb_read, arb_write)
2. Перезапись modprobe_path на свой путь (arb_write)
3. Чтение флага напрямую из physmap (lol) (arb_read)
4. Вызов commit_creds(prepare_kernel_cred(0)) из потока своей программы, находящемся в ядре

task_struct

```
struct task_struct {
    ...
    struct list_head   tasks;
    ...
    pid_t              pid  ;
    ...
    struct cred*       cred ;
    char comm[TASK_COMM_LEN];
    ...
}
```

cred

```
struct cred {
    atomic_long_t  usage;
    kuid_t          uid;        /* real UID of the task */
    kgid_t          gid;        /* real GID of the task */
    kuid_t         suid;        /* saved UID of the task */
    kgid_t         sgid;        /* saved GID of the task */
    kuid_t         euid;        /* effective UID of the task */
    kgid_t         egid;        /* effective GID of the task */
    ...
}
```

Идея эксплуатации №1

1. Указатель на первый task_struct лежит в глобальной переменной init_task
2. Пробегаемся по списку тасков в list_head, проверяем pid, сравнивая со своим
3. Когда находим свой task_struct, получаем указатель на cred
4. Обнуляем поле uid
5. Вызываем в своем бинаре setresuid(0,0,0) и вызываем шелл через execve(“/bin/sh”, 0, 0)
6. We are root!

utils #1

В тасках нам дают bzImage, а не vmlinux.

bzImage – сжатая версия vmlinux

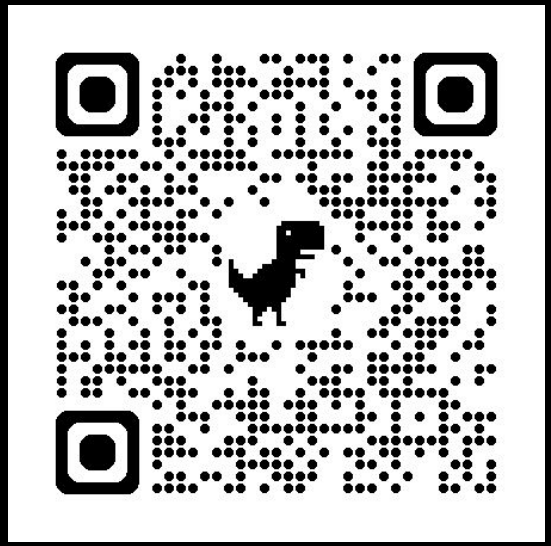
Чтобы разжать используем

github.com/marin-m/vmlinux-to-elf

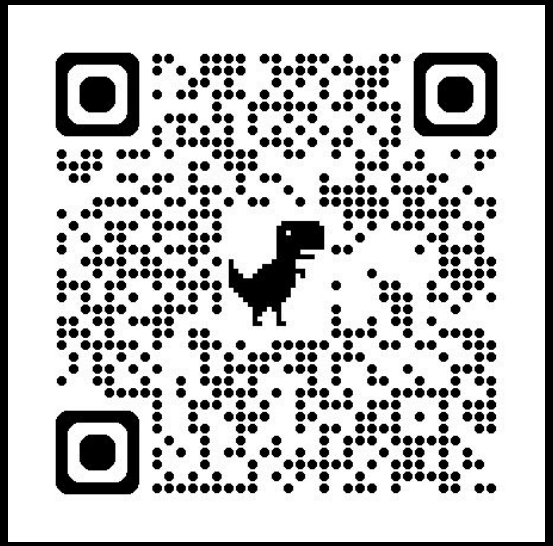
utils #2

Чаще всего в тасках нам не дают отладочную инфу.

Она нужна, чтобы знать оффсеты внутри task_struct



eBPF или история одного провала



Мой модуль gdb для получения оффсетов

modprobe_path, идея эксплуатации №2

char[] modprobe_path – глобальная переменная, в которой лежит путь на ФС до бинаря modprobe

Если перепишем на свой путь, а потом заставим ядро вызвать modprobe (всегда вызывается от root), то получим повышение привилегий

```
pwndbg> x/s &modprobe_path

0xffffffff82d39ec0 <modprobe_path>:     "/sbin/modprobe"
pwndbg>
```

modprobe_path

Как заставить ядро вызвать modprobe?

```
echo -ne '\xff\xff\xff\xff' > /tmp/lol
chmod +x /tmp/lol
/tmp/lol
```

Идея эксплуатации №3

На CTF в тасках на ядро чаще всего используется ФС, которая полностью хранится в ОЗУ

Можно просто прочитать флаг из physmap!

Как узнать KASLR?

- На реальной системе EntryBleed
- На CTF проверьте утечку через /sys/kernel/notes
- Если есть ядерный oracle, который говорит валидна ли страница памяти, то можно за очень быстрый перебор найти базу ядра

Таск mov cr3, rax с cr3.mov CTF

```
for (kaslr = 0xffffffff81000000;;kaslr+=0x10000) {
    p_d.src = kaslr;
    p_d.dst = &check;
    p_d.n = 8;
    ioctl(p_fd, 4097, &p_d);
    if (check) {
        break;
    }
}
```

Как сплойтить кучу в ядре?

В ядре много разных аллокаторов: SLUB, SLAB, SLOB.

Стандартный аллокатор сейчас SLUB.

Основная идея: под объекты одного размера аллокатор создаёт отдельную кучу.

Кроме того есть отдельные кучи для специальных объектов, т.н. dedicated cache.

Slub spraying

1. Ищем “интересные” объекты: с function pointer, индексами/размерами массивов и просто адресами. Мы должны уметь вызывать создание объекта из непривилегированного userspace.
2. Спреим этим объектом в куче с уязвимым объектом
3. Получаем примитивы: arb/rel read/write, code execution

Интересные объекты

Прекрасная статья ptr-yudai с перечислением всех интересных объектов, которые использовались в 2020 году

Список таких объектов постоянно пополняется, лучший способ находить новые – проверять эксплойты на kCTF, исследователи постоянно придумывают там новые техники.

Stack pivoting, идея эксплуатации №4

1. Если мы можем выполнить гаджет, то мы можем применить stack pivoting – технику, которая позволяет переместить стек в контролируемую нами память.
2. Ставим стек куда нам надо, после чего выполняем ROP на уже известные вам примитивы.

DEMO

НИЯУ МИФИ

BI.ZONE


0x41