Hook API

Evan Sarmiento

September 3, 2001

1 Hooks

Hooks allow you to insert code into various functions. Currently, PRFW only supports hooking into system calls. Kernel function hooks will be added at a later release. PRFW is currently at release 0.1.0, which incorporates mutex locks, clean code *grin*, and completely new names, however, the api is similar.

2 Loading Hooks

You write hooks in a KLD. If you do not know how to write a KLD, see /usr/share/examples/kld.

3 API

3.1 hook_config

hook_config takes three arguments:

```
int sl
pid_t pid
int flags
```

sl denotes the securelevel you want the hooks to run at, this is a required field. If you wish to hook only a particular process and its children, specify that processes pid in the argument. However, if you want this hook to affect all processes, use NULL.
flags is NULL if you specify a pid. However, if you hook all processes, you have two options for flags. ALLBR which means this hook affects all processes but root processes, or ALL, which specifies that all processes are affected by the hook.

3.2 hook_add

Once you've used hook_config, you may add a hook of your choice. hook_add takes four arguments.

```
int sl
int synum
pid_t pid
int (*fp)(p, uap)
```

int sl denotes the securelevel you want that hook to run at, synum is the number of the system call you need to hook into, you can get these numbers from syscall.h. SYS_SETUID is a macro defined in syscall.h, obviously, it is the number for SETUID.
pid can be a NULL argument if you want this hook to affect all processes.
the last argument is a function pointer to a function that takes a process pointer and a user area pointer struct. It is suppost to mimic the system call. Here is an example.

```
int setuid_handler(struct proc *p, struct setuid_args *uap)
{
return (EPERM);
}
...
hook_add(sl, SYS_SETUID, NULL, &setuid_handler);
```

3.3 hook_get_all

This takes zero arguments, it returns a pointer to a structure of the type hook_all_t.

3.4 hook_free

hook_free takes three arguments.

```
pid_t pid
struct hook_all_t *prfw_a
int flags
```

If you wish to free a hook from a particular process specify pid (BROKEN - someone fix this. Strange ucred mutex lock problem I don't want to get into at this moment.) Can be NULL if you want to free hooks that affect the whole system. If you want to free ALL hooks, the second argument to hook_free must not be null. In order to get a pointer to a hook_all_t struct use the hook_get_all function referenced above. flags is important. If you want to free hooks from

all processes use ALLPR. The ALL flag does nothing.
Someone fix this. There are numerous grammer and spelling errors. I need a technical writer. ;-)