IOActive Security Advisory

Title	Pointer dereference in OpenSolaris
Severity	Important
Date Discovered	September 2008
Date Reported	September 29, 2008
Date Disclosed	February 4, 2009
Author	Ilja van Sprundel

Affected Products

OpenSolaris 2008.05—earlier versions are likely to be vulnerable; however, Solaris versions 8, 9, and 10 are not.

Description

The OpenSolaris kernel exhibits a vulnerability around a *userland*[1] pointer dereference, and allows both reading from and writing to the kernel. The bug is located in a small API—`ctmpl_set()` and `ctmpl_get()`—that is applied only to data originating in userland; this API does not perform a `copyin()` or `copyout()` and also directly dereferences the pointer.

Technical Details

This section provides some context around the bug's location and demonstrates how easy it is to trigger. For starters, the `ctmpl_set()` and `ctmpl_get()` APIs can be reached from a `/proc ioctl` handler; specifically, `prioctl()`, as shown here:

```
uts/common/fs/proc/prvnops.c
const fs_operation_def_t pr_vnodeops_template[] = {
...
      VOPNAME_IOCTL,          { .vop_ioctl = prioctl },
...
}
```

The handler `prioctl()` is defined in `prioctl.c`:

[1] Anywhere outside kernel space.

```
uts/common/fs/proc/prioctl.c
int
prioctl(
      struct vnode *vp,
      int cmd,
      intptr_t arg,
      int flag,
      cred_t *cr,
      int *rvalp,
      caller_context_t *ct)
```

```
{
      switch (curproc->p_model) {
      case DATAMODEL_ILP32:
        return (prioctl32(vp, cmd, arg, flag, cr, rvalp, ct));
      case DATAMODEL_LP64:
        return (prioctl64(vp, cmd, arg, flag, cr, rvalp, ct));
      default:
            return (ENOSYS);
      }
}
...
static int
prioctl32(
      struct vnode *vp,
      int cmd,
      intptr_t arg,
      int flag,
      cred_t *cr,
      int *rvalp,
      caller_context_t *ct)
{
...
      prnode_t *pnp = VTOP(vp);
...
      if (pnp->pr_type == PR_TMPL)
            return (prctioctl(pnp, cmd, arg, flag, cr));
...
}
...
static int
prctioctl(
       prnode_t *pnp,
       int cmd,
       intptr_t arg,
       int flag,
       cred_t *cr)
{
      int error = 0;
      ct_param_t param;
...
      if (copyin((void *)arg, &param, sizeof (ct_param_t)))
            return (EFAULT);
...
      if (cmd == CT_TSET)
            error = ctmpl_set(tmpl, &param, cr);
      else
            error = ctmpl_get(tmpl, &param);
...
}
...
uts/common/sys/contract.h
typedef struct ct_param {
      uint32_t ctpm_id;
      uint32_t ctpm_size;
      void   *ctpm_value;
} ct_param_t;
```

The APIs `ctmpl_set()` and `ctmpl_get()` are used in the following manner:

```
uts/common/os/contract.c
int
ctmpl_set(ct_template_t *template, ct_param_t *param, const
cred_t *cr)
{
	int result = 0;
	uint64_t param_value;

	if (param->ctpm_id == CTP_COOKIE ||
	    param->ctpm_id == CTP_EV_INFO ||
	    param->ctpm_id == CTP_EV_CRITICAL) {
	      if (param->ctpm_size < sizeof (uint64_t)) {
			return (EINVAL);
	      } else {
			param_value = *(uint64_t *)param->ctpm_value;
	      }
	}
...
}
...
int
ctmpl_get(ct_template_t *template, ct_param_t *param)
{
	int result = 0;
	uint64_t *param_value;

	if (param->ctpm_id == CTP_COOKIE ||
	    param->ctpm_id == CTP_EV_INFO ||
	    param->ctpm_id == CTP_EV_CRITICAL) {
	      if (param->ctpm_size < sizeof (uint64_t)) {
			return (EINVAL);
	      } else {
			param_value = param->ctpm_value;
			param->ctpm_size = sizeof (uint64_t);
	      }
	}

	mutex_enter(&template->ctmpl_lock);
	switch (param->ctpm_id) {
	case CTP_COOKIE:
		*param_value = template->ctmpl_cookie;
		break;
	case CTP_EV_INFO:
		*param_value = template->ctmpl_ev_info;
		break;
	case CTP_EV_CRITICAL:
		*param_value = template->ctmpl_ev_crit;
		break;
...
}
```

In order to access `ctmpl_get()` and/or `ctmpl_set()`, you must issue an `ioctl()` on a file in `/proc` of type `PR_TMPL`; these files can be found in `/proc/<pid>/lwp/<lwpid>/templates/<id>`. Also refer to `prioctl32()`.

The handler for these files is `prctioctl()`, which examines the `ioctl` command (`cmd`) to determine its next action:

- `CT_TSET` triggers a call to `ctmpl_set()`
- `CT_TGET` triggers a call to `ctmpl_get()`

The structure `ct_param_t` (obtained from userland) is then copied and passed onto these two APIs. This structure contains the pointer `ctpm_value`, which is dereferenced directly in `ctmpl_set()` and/or `ctmpl_get()` instead of using `copyin()` and/or `copyout()`—this allows an attacker to read/write from and in kernel memory.

Remediation

This vulnerability has been fixed in OpenSolaris 2008.11; the full patch can be obtained from:

<http://hg.genunix.org/onnv-gate.hg/rev/3160250b7dc5>