# softraid boot

Stefan Sperling <stsp@openbsd.org>

EuroBSDcon 2015

# Introduction to softraid

OpenBSD's softraid(4) device

- emulates a host controller which provides a virtual SCSI bus
- uses *disciplines* to perform I/O on underlying disks:  
  RAID 0, RAID 1, RAID 5, CRYPTO, CONCAT
- borrows the bioctl(8) configuration utility from the bio(4) hardware RAID abstraction layer

```
softraid0 at root
scsibus4 at softraid0: 256 targets
sd9 at scsibus4 targ 1 lun 0: <OPENBSD, SR RAID 1, 005> SCSI2 0/direct fixed
sd9: 1430796MB, 512 bytes/sector, 2930271472 sectors
```

(RAID 1 softraid volume appearing as disk sd9)

# Introduction to softraid

OpenBSD's softraid(4) device

- uses *chunks* (disklabel slices of type RAID) for storage
- records meta data at the start of each chunk:

  format version, UUID, volume ID, no. of chunks, chunk ID,
  RAID type and size, and other optional meta data

```
# disklabel -pm sd2
[...]
#                size           offset  fstype [fsize bsize  cpg]
  c:       1430799.4M                0  unused
  d:       1430796.9M               64    RAID
# bioctl sd9
Volume      Status               Size Device
softraid0 0 Online      1500298993664 sd9     RAID1
          0 Online      1500298993664 0:0.0   noencl <sd2d>
          1 Online      1500298993664 0:1.0   noencl <sd3d>
```

(RAID 1 softraid volume using sd2d and sd3d for storage)

# Introduction to softraid

softraid volumes can be assembled manually with bioctl(8) or automatically during boot

- softraid UUID ties volumes and chunks together

  disk device names and disklabel UUIDs are **irrelevant** when softraid volumes are auto-assembled
- volume IDs are used to attach volumes in a predictable order

  stable disk device names unless disks are added/removed
- chunk IDs make chunks appear in a predictable order

  important for e.g. CONCAT discipline

# softraid disciplines overview

Currently available disciplines:

- RAID 0, RAID 1, RAID 5

  spread/copy data across 2 or more chunks
- CRYPTO

  encrypt data, protected by a passphrase or a key disk
- CONCAT

  concatenate disks for more space

Disciplines cannot be nested yet!
So no CRYPTO on top of RAID 1, for instance.

# RAID 1 discipline

The RAID 1 discipline

- auto-assembles by default
- can be used as a boot disk on i386, amd64, sparc64

  bootloader loads kernel image from any available chunk

RAID 1 boot disk install on i386/amd64:

```
Welcome to the OpenBSD/amd64 5.8 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? s

# fdisk -iy sd0
# fdisk -iy sd1
# echo -n "d\n\n\n\nRAID\nw\nq\n\n" | disklabel -E sd0
# echo -n "d\n\n\n\nRAID\nw\nq\n\n" | disklabel -E sd1
# bioctl -c 1 -l /dev/sd0d,/dev/sd1d softraid0
sd2 at scsibus2 targ 1 lun 0: <OPENBSD, SR RAID 1, 005> SCSI2 0/direct fixed
```

Now exit the shell and install as usual, using sd2 as root disk.

# CRYPTO discipline

The CRYPTO discipline

- encrypts data with AES XTS 256

  algorithm fixed (except in meta data), knobs are for knobs
- supports AES-NI for hardware crypto

  unnoticeable overhead on modern laptops
- supports full disk encryption on i386, amd64, sparc64

  bootloader decrypts kernel image
- encrypts AES XTS key with AES ECB 256

  AES ECB “mask key” can be a user passphrase or key disk
  key disk: chunk containing fixed random data used as mask key

# CRYPTO discipline

Fully encrypted disk install on i386/amd64:

```
Welcome to the OpenBSD/amd64 5.8 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? s

# fdisk -iy sd0
# disklabel -E sd0
> a
partition: [a] d
offset: [64]
size: [16777216]
FS type: [4.2BSD] RAID
> w
> q
# bioctl -c C -l /dev/sd0d softraid0
New passphrase:
Re-type passphrase:
sd1 at scsibus2 targ 1 lun 0: <OPENBSD, SR CRYPTO, 005> SCSI2 0/direct fixed
```

Now exit the shell and install as usual, using sd1 as root disk.

# CRYPTO discipline

softraid key disks

- can be put onto any disk device

  tiny USB sticks, SD cards, ...
- auto-assemble at boot if disk device is reported by the bios

  check with `machine diskinfo` at the `boot>` prompt
- can be backed up and restored using `dd(1)`

Backup:

```
dd bs=8192 skip=1 if=/dev/rsd1d of=backup-keydisk.img
```

Restore:

```
dd bs=8192 seek=1 if=backup-keydisk.img of=/dev/rsd1d
```

# CRYPTO discipline

softraid key disks

- store softraid meta data and nothing else

  1 MB is more than enough

- can share one physical disk to unlock multiple crypto volumes

```
# disklabel sd1
[...]
#                size           offset  fstype [fsize bsize  cpg]
  c:         15669248                0  unused
  d:            10192         15621053    RAID
  e:            16065         15631245    RAID
  i:         15615148               32   MSDOS
```

(key disk configuration where sd1d unlocks the root disk and sd1e unlocks the home partition on a separate drive; unused space is FAT-32 formatted)

# CRYPTO discipline

Fully encrypted disk install on i386/amd64 with a key disk (sd1d):

```
Welcome to the OpenBSD/amd64 5.8 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? s

# fdisk -iy sd0
# fdisk -iy sd1
# echo -n "d\n\n\n\nRAID\nw\nq\n\n" | disklabel -E sd0
# disklabel -E sd1
> a
partition: [a] d
offset: [64]
size: [16777216] 1M
FS type: [4.2BSD] RAID
> w
> q
# bioctl -c C -l /dev/sd0d -k /dev/sd1d softraid0
sd2 at scsibus2 targ 1 lun 0: <OPENBSD, SR CRYPTO, 005> SCSI2 0/direct fixed
```

Now exit the shell and install as usual, using sd2 as root disk.

# Booting from softraid

System components involved when booting from softraid:

- installboot(8)

  place boot loaders into softraid meta data area
  i386[1]: MBR loads first-stage boot loader from there
  sparc64: OpenFirmware loads first-stage from superblock
- second-stage boot loaders

  assemble softraid volumes, load kernel from the right volume,
  tell kernel that it was booted from softraid
- OpenBSD kernel

  assemble softraid volumes, detect root filesystem on softraid

[1] amd64 boots the same way

# installboot(8)

`installboot(8)` writes boot loader and boot blocks

|  | skip | SR_META_DATA | SR_BOOT_LOADER | SR_BOOT_BLOCKS |
|---|---|---|---|---|
| blocks | 16 | 64 | 320 | 128 |

- SR_BOOT_LOADER:

  i386: single-inode FFS filesystem containing `boot(8)`
  sparc64: copy of ofwboot (see `boot_sparc64(8)`)

- SR_BOOT_BLOCKS:

  i386: `biosboot(8)`, reads /boot from SR_BOOT_LOADER
  sparc64: unused – first stage resides in superblock

# installboot(8)

installboot(8) also

- adds meta data option to indicate bootable chunk
- saves disklabel UID of root and boot disks there

```
#define SR_MAX_BOOT_DISKS 16
struct sr_meta_boot {
        struct sr_meta_opt_hdr   sbm_hdr;
        u_int32_t                sbm_bootblk_size;
        u_int32_t                sbm_bootldr_size;
        u_char                   sbm_root_duid[8];
        u_char                   sbm_boot_duid[SR_MAX_BOOT_DISKS][8];
} __packed;
```

# i386: boot(8)

i386 second-stage boot loader

- assembles softraid volumes

  RAID 1: load kernel from any online chunk
  CRYPTO: unlock with passphrase or keydisk, load kernel
- has softraid support in disk I/O strategy() function
- passes additional arguments to the kernel

  boot softraid volume UUID
  and mask key in case of CRYPTO

# i386: boot argument passing

i386 uses a linked list of variable-sized boot arguments

```
typedef struct _boot_args {
        int ba_type;                    /* e.g. BOOTARG_BOOTSR */
        size_t ba_size;                 /* e.g. sizeof(bios_bootsr_t) */
        struct _boot_args *ba_next; /* next argument in list */
        int ba_arg[1];                  /* pointer to argument data */
} bootarg_t;

extern bootarg_t *bootargp;             /* list head address known to
                                           boot loader and kernel */

#define BOOTARG_BOOTSR 10               /* softraid volume UUID and mask key */
#define BOOTSR_UUID_MAX 16
#define BOOTSR_CRYPTO_MAXKEYBYTES 32
typedef struct _bios_bootsr {
        u_int8_t        uuid[BOOTSR_UUID_MAX];
        u_int8_t        maskkey[BOOTSR_CRYPTO_MAXKEYBYTES];
} __packed bios_bootsr_t;
```

# i386: kernel

- assembles softraid volumes
- detects softraid boot via hints from boot loader

  boot disklabel UID in a softraid volume? booted from softraid!
  as usual, ‘a’ partition in disklabel is the root partition
- uses CRYPTO mask key provided by boot loader

  no need to enter passphrase twice
  may unplug key disk while kernel boots (unless it unlocks additional volumes during boot)

# sparc64: bootblock.fth

sparc64 first stage boot loader

- runs in OpenFirmware environment

  written in Forth<br>
  softraid support added by jsing@, thanks!!!
- looks for RAID partition **with letter ‘a’** in disklabel
- reads second-stage ofwboot program from softraid meta data

```
\ Are we booting from a softraid volume?
is-bootable-softraid? if
   sr_boot_offset sr_boot_size dev_bsize *
   softraid-boot ( blockno size -- load-base )
else
   " /ofwboot" load-file ( -- load-base )
then
```

# sparc64: ofwboot

sparc64 second-stage boot loader has differences from i386

- walks OpenFirmware device tree to find all disks
- problem: arguments not passed via shared memory

  arguments come from OpenFirmware “bootline”
  contains whatever the user typed at ok> prompt
- how to pass softraid UUID and mask key?

  considered using OF_setprop()
  but mask key might end up in persistent NVRAM...

```
ok setenv boot-file sr0a:/bsd
```

(configure a sparc64 machine to boot from softraid by default)

# sparc64: boot argument passing

Solution: Added a new ELF section to sparc64 kernel image.

```
ld.script: openbsd_bootdata  0x65a41be6; /* PT_OPENBSD_BOOTDATA */

/* MD boot data in .openbsd.bootdata ELF segment */
struct openbsd_bootdata {
u_int64_t  version;
u_int64_t  len; /* of structure */

u_int8_t sr_uuid[BOOTSR_UUID_MAX];
u_int8_t sr_maskkey[BOOTSR_CRYPTO_MAXKEYBYTES];
} __packed;

#define BOOTDATA_VERSION 1
```

# sparc64: kernel

sparc64 kernel gets softraid info from bootdata ELF section

```
struct openbsd_bootdata obd __attribute__((section(".openbsd.bootdata")));

if (obd.version == BOOTDATA_VERSION &&
    obd.len == sizeof(struct openbsd_bootdata)) {
#if NSOFTRAID > 0
        memcpy(sr_bootuuid.sui_id, obd.sr_uuid,
            sizeof(sr_bootuuid.sui_id));
        memcpy(sr_bootkey, obd.sr_maskkey, sizeof(sr_bootkey));
#endif
        explicit_bzero(obd.sr_maskkey, sizeof(obd.sr_maskkey));
}
```

Otherwise same as i386.

# Thank you!

Any questions?