

ZFS

Enhancing the Open Source
Storage System (and the Kernel)

Who am I?

- Soy Christian Kendi
- I do ...
 - IT-Security Consultant
 - (Kernel)-Developer
 - Penetration tester
 - Exploit coder
 - CEO & Founder of Iron Software

What is this->talk about?

- ZFS (Zetabyte File System)
- Open Solaris Gate (Kernel)

What is this->not about?

- Further explanation on how file systems work in general
- Deeper insight into the design and development of ZFS (raidz, allocator, etc.)
- Rootkits (well 😊), we are not too far away from a rootkit

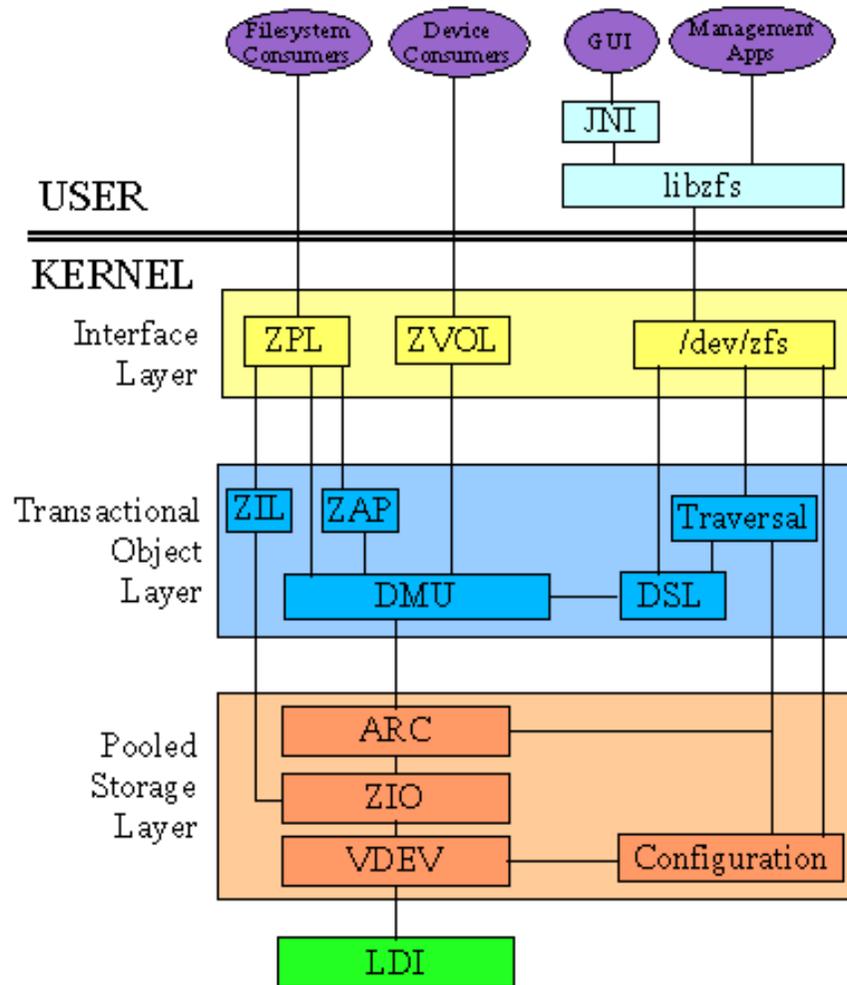
What is ZFS?

- Revolutionary Open Source Storage System
- 128-bit file system
- Capable of storing 16 EiB (1,024 Pebibytes)
- Transparent Compression, Encryption, etc...
- Ported to multiple Operating Systems (Mac OS X, BSD, Linux)

ZFS features

- Storage pools
- Snapshots
- (Incremental) Backups between Snapshots
- Variable block-size up to 128-kilobyte
- On-the-fly compression (LZJB, gzip[1-9])
- 256-bit block checksums (fletcher2/4 or SHA-256)
- Self Healing (On-the-fly Error Correction)
- Open Source (yes its a feature ;)

ZFS internal overview



ZPL (ZFS POSIX Layer)

ZVOL (ZFS Emulated Volume)

DMU (Data Management Unit)

DSL (Dataset and Snapshot Layer)

ZAP (ZFS Attribute Processor)

ZIL (ZFS Intent Log)

ARC (Adaptive Replacement Cache)

Pool Configuration (SPA)

ZIO (ZFS I/O Pipeline)

DEMO

Create a pool, filesystem and
work with snapshots

Security aspects about ZFS

- Offline honey pot analysis
- Backup's of Mission Critical Systems
- Embedded Antivirus support for blocking infected files
- Revert a hacked host back to installation state within seconds
- Forensics by differential FS analysis
- ACLs

Storage Security Concerns

- The most valuable information is stored in databases and storage Systems
- Having access to the company's storage equals having the company
- More to come later...

ZFS enhancing, how?

- A file system is always kernel based
- Open Solaris ON NV (Gate) Source Code
- Building a kernel module
- Hooking internal ZFS functions
- Provide a separate FS-Layer for the “enhances”

ZFS enhancing, helpers?

- kmdb is incredible!
- dtrace & truss

```
modules::list "struct modctl" mod_next | ::print "struct modctl"  
{  
  mod_next = 0xfec479e0  
  mod_prev = 0xda0564c8  
  mod_id = 0  
  mod_mp = 0xfec42d90  
  mod_inprogress_thread = 0  
  mod_modinfo = 0  
  mod_linkage = 0  
  mod_filename = 0xfec42d68 "/platform/i86pc/kernel//unix"  
  mod_modname = 0xfec42d80 "unix"
```

ZFS enhancing, how? #2

- `movl $add, %eax; jmp *%eax`
- Assembly code injection

```
[01] 0xf9e7635c::dis
0xf9e7635c:      movl    $0xfa122f50,%eax <zfs`zfs_mkdir>
0xf9e76361:      jmp     *%eax
0xf9e76363:      inb    (%dx)
0xf9e76364:      decl   %esp
0xf9e76365:      movl   0x8(%ebp),%eax
0xf9e76368:      movl   0x10(%eax),%esi
0xf9e7636b:      movl   (%esi),%ebx
```

ZFS enhancing, how? #2

- But of course we don't want to rewrite the entire ZFS code.

```
[01] > *orig_zfs_dirlook::dis
0xf9e6821c:    movl    $0xfa2813a0,%eax <zfs`zfs_dirlook>
0xf9e68221:    jmp     *%eax
0xf9e68223:    addb   %al,(%eax)
0xf9e68225:    addb   %al,(%eax)
0xf9e68227:    addb   %cl,(%edi)
0xf9e68229:    movl   $0xfc08502,%esi
0xf9e6822e:    testb  %bh,0x83000001(%ebp)
0xf9e68234:    clc
0xf9e68235:    jne    +0x1b    <0xf9e68253>
0xf9e68238:    movsbl 0x1(%edx),%eax
0xf9e6823c:    testl  %eax,%eax
```

- First bytes are restored when executing from the orig_handler within the hook.

ZFS enhancing, what?

- dtrace and truss are your friends
- Find the desired functions

```
-> getdents64
-> getf
  -> set_active_fd
  <- set_active_fd
<- getf
-> fop_rwlock
  -> fs_rwlock
  <- fs_rwlock
<- fop_rwlock
-> fop_readdir
  -> crgetmapped
  <- crgetmapped
  -> zfs_readdir
  -> rrw_enter
  -> rrw_enter_read
  <- rrw_enter_read
  <- rrw_enter
  -> zap_cursor_init_serialized
  <- zap_cursor_init_serialized
  -> kmem_atloc
  -> kmem_cache_alloc
  <- kmem_cache_alloc
  <- kmem_alloc
```

```
ioctl(3, ZFS_IOC_OBJSET_STATS, 0x080450C0) = 0
brk(0x080B4000) = 0
ioctl(3, ZFS_IOC_POOL_STATS, 0x08045020) = 0
ioctl(3, ZFS_IOC_POOL_GET_PROPS, 0x08046080) = 0
ioctl(3, ZFS_IOC_DATASET_LIST_NEXT, 0x080460E0) Err#3 ESRCH
ioctl(3, ZFS_IOC_OBJSET_STATS, 0x080450C0) = 0
ioctl(3, ZFS_IOC_POOL_STATS, 0x08045020) = 0
ioctl(3, ZFS_IOC_POOL_GET_PROPS, 0x08046080) = 0
ioctl(3, ZFS_IOC_DATASET_LIST_NEXT, 0x080460E0) = 0
ioctl(3, ZFS_IOC_DATASET_LIST_NEXT, 0x080460E0) = 0
ioctl(3, ZFS_IOC_DATASET_LIST_NEXT, 0x080460E0) = 0
ioctl(3, ZFS_IOC_OBJSET_STATS, 0x08044020) = 0
ioctl(3, ZFS_IOC_DATASET_LIST_NEXT, 0x080460E0) = 0
ioctl(3, ZFS_IOC_OBJSET_STATS, 0x08044020) = 0
ioctl(3, ZFS_IOC_DATASET_LIST_NEXT, 0x08045040) = 0
ioctl(3, ZFS_IOC_OBJSET_STATS, 0x08042F80) Err#12 ENOMEM
ioctl(3, ZFS_IOC_OBJSET_STATS, 0x08042F80) = 0
```

ZFS enhancing, syscalls?

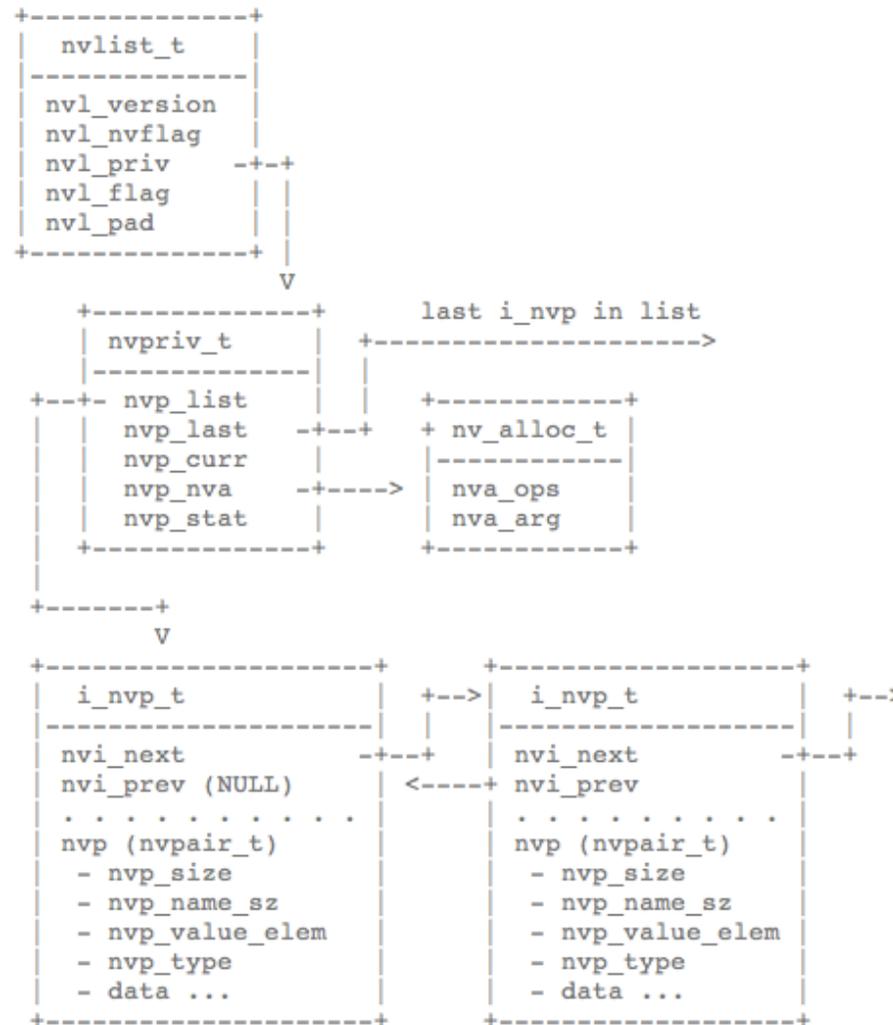
- Okay, admitted. Old but nice. Why?
- Crypto Gate ;)
- Just to be flexible

```
switch (zc->zc_crypto.zic_cmd) {  
case ZFS_IOC_CRYPT_LOAD_KEY_SPA:  
    error = spa_crypto_key_load(spa, &zc->zc_crypto);  
    break;  
case ZFS_IOC_CRYPT_UNLOAD_KEY_SPA:  
    error = spa_crypto_key_unload(spa);  
    break;  
case ZFS_IOC_CRYPT_CHANGE_KEY_SPA:  
    error = spa_crypt_key_change(spa, &zc->zc_crypto);  
    break;  
}
```

ZFS enhancing, functions?

- All userland <-> kernel communication is in `zfs_ioctl.c`
- `zfs_ioc_pool_configs()` will deliver all available pools
 - Or not
- Solaris handles dynamic data with nvlists
- Dynamic means DYNAMIC.

ZFS enhancing, nvlists



ZFS enhancing, functions?

```
/* VOFS HERE */
dsym(int,          zfs_dirlook,          vnode_t *dzp, char *name, vnode_t **vpp, int flags, int *deflg, pathname_t *rpn);
dsym(int,          zfs_mkdir,            vnode_t *dvp, char *dirname, vattr_t *vap, vnode_t **vpp, cred_t *cr);
dsym(int,          zfs_zaccess,          znode_t *zp, int mode, int flags, boolean_t skipaclchk, cred_t *cr);
dsym(int,          zfs_read,             vnode_t *vp, uio_t *uio, int ioflag, cred_t *cr, caller_context_t *ct);
dsym(int,          zfs_write,            vnode_t *vp, uio_t *uio, int ioflag, cred_t *cr, caller_context_t *ct);
dsym(int,          zfs_ioc_pool_get_history, zfs_cmd_t *zc);
dsym(void,         zfs_log_history,      zfs_cmd_t *zc);
dsym(int,          zfs_ioc_pool_configs, zfs_cmd_t *zc);

/* ZFS INTERNALS */
dsym(int,          zfs_ioc_pool_stats,    zfs_cmd_t *zc);
dsym(int,          zfs_ioc_dataset_list_next, zfs_cmd_t *zc);
dsym(int,          dsl_dir_is_private,     dsl_dir_t *dd);
dsym(int,          dataset_namecheck,     const char *path, namecheck_err_t *why, char *what);

/* ZFS IMPORTS - ONLY - */
isym(int,          spa_get_stats,         const char *name, nvlist_t **config, char *altroot, size_t buflen);
isym(nvlist_t *,  spa_all_configs,       uint64_t *generation);
isym(int,          put_nvlist,            zfs_cmd_t *zc, nvlist_t *nvl);
isym(int,          dataset_name_hidden,   const char *name);

/* VOFS IMPORTS - ONLY - */
isym(int,          zfs_open,              vnode_t **vpp, int flag, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_close,             vnode_t *vp, int flag, int count, offset_t offset, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_ioctl,            vnode_t *vp, int com, intptr_t data, int flag, cred_t *cred, int *rvalp, caller_context_t *ct);
isym(int,          zfs_access,            vnode_t *vp, int mode, int flag, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_lookup,           vnode_t *dvp, char *nm, vnode_t **vpp, struct pathname *pnp, int flags,
                                                vnode_t *rdir, cred_t *cr, caller_context_t *ct, int *direntflags, pathname_t *realpnp);
isym(int,          zfs_create,            vnode_t *dvp, char *name, vattr_t *vap, vxexcl_t excl, int mode, vnode_t **vpp,
                                                cred_t *cr, int flag, caller_context_t *ct, vsecattr_t *vsecp);
isym(int,          zfs_remove,            vnode_t *dvp, char *name, cred_t *cr, caller_context_t *ct, int flags);
isym(int,          zfs_rmdir,             vnode_t *dvp, char *name, vnode_t *cwd, cred_t *cr, caller_context_t *ct, int flags);
isym(int,          zfs_readdir,           vnode_t *vp, uio_t *uio, cred_t *cr, int *eofp, caller_context_t *ct, int flags);
isym(int,          zfs_fsync,             vnode_t *vp, int syncflag, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_inactive,          vnode_t *vp, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_getattr,           vnode_t *vp, vattr_t *vap, int flags, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_setattr,           vnode_t *vp, vattr_t *vap, int flags, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_rename,            vnode_t *sdvp, char *snm, vnode_t *tdvp, char *tnm, cred_t *cr, caller_context_t *ct, int flags);
isym(int,          zfs_symlink,           vnode_t *dvp, char *name, vattr_t *vap, char *link, cred_t *cr, caller_context_t *ct, int flags);
isym(int,          zfs_readlink,          vnode_t *vp, uio_t *uio, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_link,              vnode_t *tdvp, vnode_t *svp, char *name, cred_t *cr, caller_context_t *ct, int flags);
isym(int,          zfs_seek,              vnode_t *vp, offset_t off, offset_t *hoffp, caller_context_t *ct);
isym(int,          zfs_fid,               vnode_t *vp, fid_t *fidp, caller_context_t *ct);
isym(int,          zfs_pathconf,          vnode_t *vp, int cmd, ulong_t *valp, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_getsecattr,        vnode_t *vp, vsecattr_t *vsecp, int flag, cred_t *cr, caller_context_t *ct);
isym(int,          zfs_setsecattr,        vnode_t *vp, vsecattr_t *vsecp, int flag, cred_t *cr, caller_context_t *ct);
```

ZFS, The Hackers point of View

- Hide “something”
- Anti-forensics against unloading the module
- + Hide data in a way that offline analysis is hard
- Yes, Crypto is a solution, but....
 - the key must be stored somewhere

ZFS, The Hackers point of View #2

- Some ideas...
 - a private storage pool
 - mirror the companys pool over the internet.
(iSCSI, zfs send)

ZFS, The Hackers point of View #3

- Interesting ioctl's

ZFS_IOC_SEND

ZFS_IOC_RECV

ZFS_IOC_SNAPSHOT

ZFS_IOC_POOL_STATS

ZFS_IOC_POOL_GET_PROPS

ZFS_IOC_POOL_CONFIGS

ZFS_IOC_SNAPSHOT_LIST_NEXT

ZFS_IOC_DATASET_LIST_NEXT

ZFS enhancing, hide something?

- It's all there by it-self. “.zfs” is invisible
- Analysis and code reading/auditing revealed interesting stuff

```
#include "zfs_namecheck.h"

int
dataset_name_hidden(const char *name)
{
    if (strchr(name, '$') != NULL)
        return (1);

    return (0);
}
```

ZFS enhancing, hide something?

#2

- .zfs is a VFS (Virtual File System) layer by itself
- With ZFS we don't just hide directories or files, we hide entire file systems or storage pools
 - Each hidden FS/pool has its own VFS entry
 - VFS controls all FS specific operations
VOPNAME_LOOKUP, { .vop_lookup = ksh_root_lookup },
 - > Have different ZFS revisions in a single kernel
 - ZFS Crypto Gate

ZFS enhancing, hide something?

#3

```
in
zfs_dirlook(znode_t *dzp, char *name, vnode_t **vpp, int flags,
            int *deflg, pathname_t *rpn)
{
    zfs_dirlock_t *dl;
    znode_t *zp;
    int error = 0;

    if (name[0] == 0 || (name[0] == '.' && name[1] == 0)) {
        *vpp = ZTOV(dzp);
        VN_HOLD(*vpp);
    } else if (name[0] == '.' && name[1] == '.' && name[2] == 0) {
        zfsvfs_t *zfsvfs = dzp->z_zfsvfs;
        /*
         * If we are a snapshot mounted under .zfs, return
         * the vp for the snapshot directory.
         */
        if (dzp->z_phys->zp_parent == dzp->z_id &&
            zfsvfs->z_parent != zfsvfs) {
            error = zfsctl_root_lookup(zfsvfs->z_parent->z_ctldir,
                                      "snapshot", vpp, NULL, 0, NULL, kcred,
                                      NULL, NULL, NULL);
            return (error);
        }
        rw_enter(&dzp->z_parent_lock, RW_READER);
        error = zfs_zget(zfsvfs, dzp->z_phys->zp_parent, &zp);
        if (error == 0)
            *vpp = ZTOV(zp);
        rw_exit(&dzp->z_parent_lock);
    } else if (zfs_has_ctldir(dzp) && strcmp(name, ZFS_CTLDIR_NAME) == 0) {
        *vpp = zfsctl_root(dzp);
    }
}
```

ZFS enhancing, Anti-forensics

- ZFS binary and kernel module contain checks for invalid datasets, i.e. internal datasets
- Built-in support for hiding Storage Pools and ZFSs across Systems.

```
Apr  9 13:06:16 opensolaris-vm winnipu: [ID 181094 kern.warning]  
WARNING: hook_zfs_ioc_dataset_list_next(): zc_name: rpool/$MOS cookie:  
133e8aad
```

```
Apr  9 13:06:17 opensolaris-vm winnipu: [ID 181094  
kern.warning] WARNING: hook_zfs_ioc_dataset_list_next():  
zc_name: rpool/$ORIGIN cookie: 13763f21
```

- Module independent, 0day? 😊
- zfs send independent
- Snapshot resistant
- Pools and ZFS's wont show up even if module is not loaded
- Takes advanced personnel to find the pool

ZFS enhancing, Anti-forensics #2

- Patch zfs binary to allow ,,\$‘
- Hook dataset_namecheck()
 - Allow “all” characters to special PIDs
- LD_PRELOAD recompiled libzfs.so.1 with new zfs binary
- list, create, snapshot, send/rcv, etc...
considered internal datasets

Anti-debugging?

- Because, the code is all mine.
- Symbol relocation is done in the Elf header
- The Module pointer holds `mp->symtbl`
- `sp = (Sym*)(mp->symtbl + i * mp->symhdr->sh_entsize); & sp->st_value = 0x????????`
is your friend
- `kobj_sync()` refresh's the module symtab
- Have fun debugging `0xfe??????` and `m0e3asd`

DEMO

Let's make some magic

Poopool

- What is “poopool”?

```
$ ./new_zfs.sh list poopool
sending request for PID 806... done!
NAME      USED  AVAIL  REFER  MOUNTPOINT
poopool   40.3M  123M  40.1M  none
```

```
$ zpool status poopool
  pool: poopool
  state: ONLINE
  scrub: none requested
config:
```

NAME	STATE	READ	WRITE	CKSUM
poopool	ONLINE	0	0	0
/root/poopool.raw	ONLINE	0	0	0

```
errors: No known data errors
```

Hidden subpools with “\$”

- poopool/\$bleh
- Won't automount (nice)

NAME	USED	AVAIL	REFER	MOUNTPOINT
poopool/\$bleh	18K	123M	18K	/system/.zfs/asdf

- Everything about ZFS can be logged

```
Jan  4 18:24:21 opensolaris-vm ksh_zfs: WARNING: hook_ioctl(): ZFS_IOC_POOL_GET_PROPS
Jan  4 18:24:21 opensolaris-vm ksh_zfs: WARNING: hook_zfs_log_history(): log query: poopool
Jan  4 18:24:21 opensolaris-vm ksh_zfs: WARNING: hook_zfs_log_history(): zc_name: poopool
Jan  4 18:24:21 opensolaris-vm ksh_zfs: WARNING: hook_zfs_log_history(): denying history log on pool: poopool
```

Let's sum it up

- Kernel hacking
- Some ZFS internals
- VFS Layers
- Dynamic Symbol Relocation

Outlook

- Hot-patching mission critical systems
- Implementing new (desired) features into a running system
- Adapting a second protection layer
- ZFS Crypto gate in code review (still)

Questions?

Thanks for listening

Have fun!