FreeBSD Developer Summit
TrustedBSD: Audit + priv(9)

10 November 2006

Robert Watson

FreeBSD Project

Computer Laboratory
University of Cambridge

UNIVERSITY OF
CAMBRIDGE

TrustedBSD

- Audit
  - Quick audit tutorial
  - Adding audit support to new kernel features
  - Userland audit work
- Privileges
  - Priv(9) API

Introduction to Audit

- Log of security-relevant events
  - Secure
  - Reliable
  - Fine-grained
  - Configurable
- A variety of uses including
  - Post-mortem analysis
  - Intrusion detection
  - Live system monitoring, debugging

Audit Basics

- Audit records describe individual events
  - Attributable (to an authenticated user)
  - Non-attributable (no authenticated user)
  - Selected (configured to be audited)
- Most audit events fall into three classes
  - Access control
  - Authentication
  - Security management
- Audit log files are called “trails”

Auditable Events

- Access control
  - System calls checking for super user privilege
  - System calls with file system access control checks
    - Including path name lookup!
  - Login access control decisions
- Authentication, Account Management
  - Password changes, successful authentication, failed authentication, user administration
- Audit related events

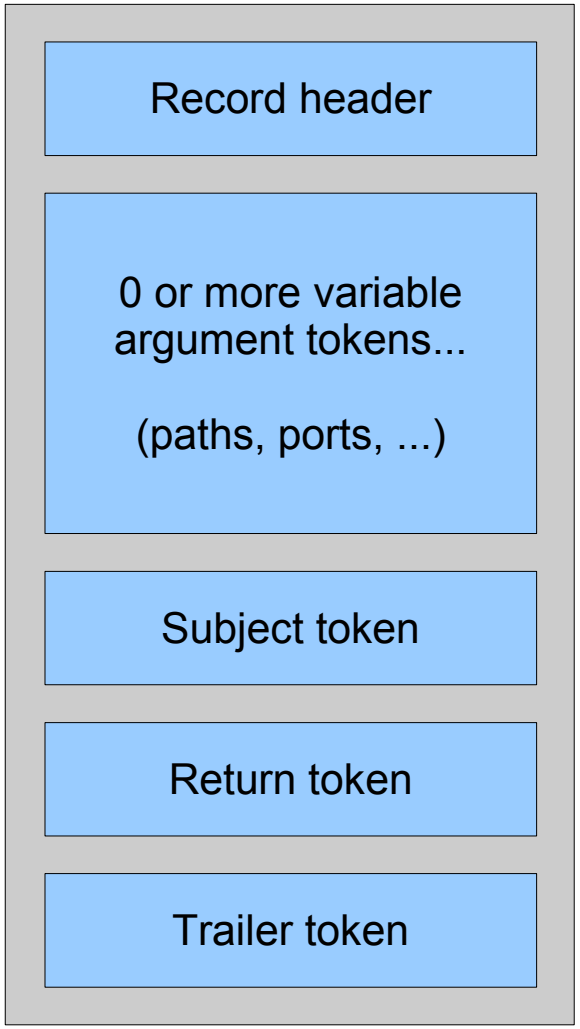
Audit Software Components

- Kernel audit event engine
  - Event allocation, preselection, argument gathering, event commit, queuing, worker thread, pipe system
- Kernel event gathering
  - System calls, argument gathering
- OpenBSM
  - Userland tools, library, configuration files
- Userland integration
  - Login(1), su(1), sshd(8), ...

BSM API and File Format

- Sun's Basic Security Module (BSM) de facto industry standard
  - File formats
    - Token-oriented audit trail format (almost TLV)
    - Audit configuration and databases
  - APIs
    - Construct, parse, process audit record streams
    - Manage audit state, pre-selection model
- Compatibility with many existing libraries and tools for free

Record Format




```
header,129,1,AUE_OPEN_R,0,Tue Feb 21 00:12:23 2006, +
253 msec
argument,2,0,flags
path,/lib/libc.so.6
attribute,444,root,wheel,16842497,11663267,46706288
subject,-1,root,wheel,root,wheel,319,0,0,0.0.0.0
return,success,6
trailer,129

header,108,1,AUE_CLOSE,0,Tue Feb 21 00:12:23 2006, +
255 msec
argument,2,0x6,fd
attribute,444,root,wheel,16842497,11663267,46706288
subject,-1,root,wheel,root,wheel,319,0,0,0.0.0.0
return,success,0
trailer,108
```

Audit Configuration: Pre-Selection

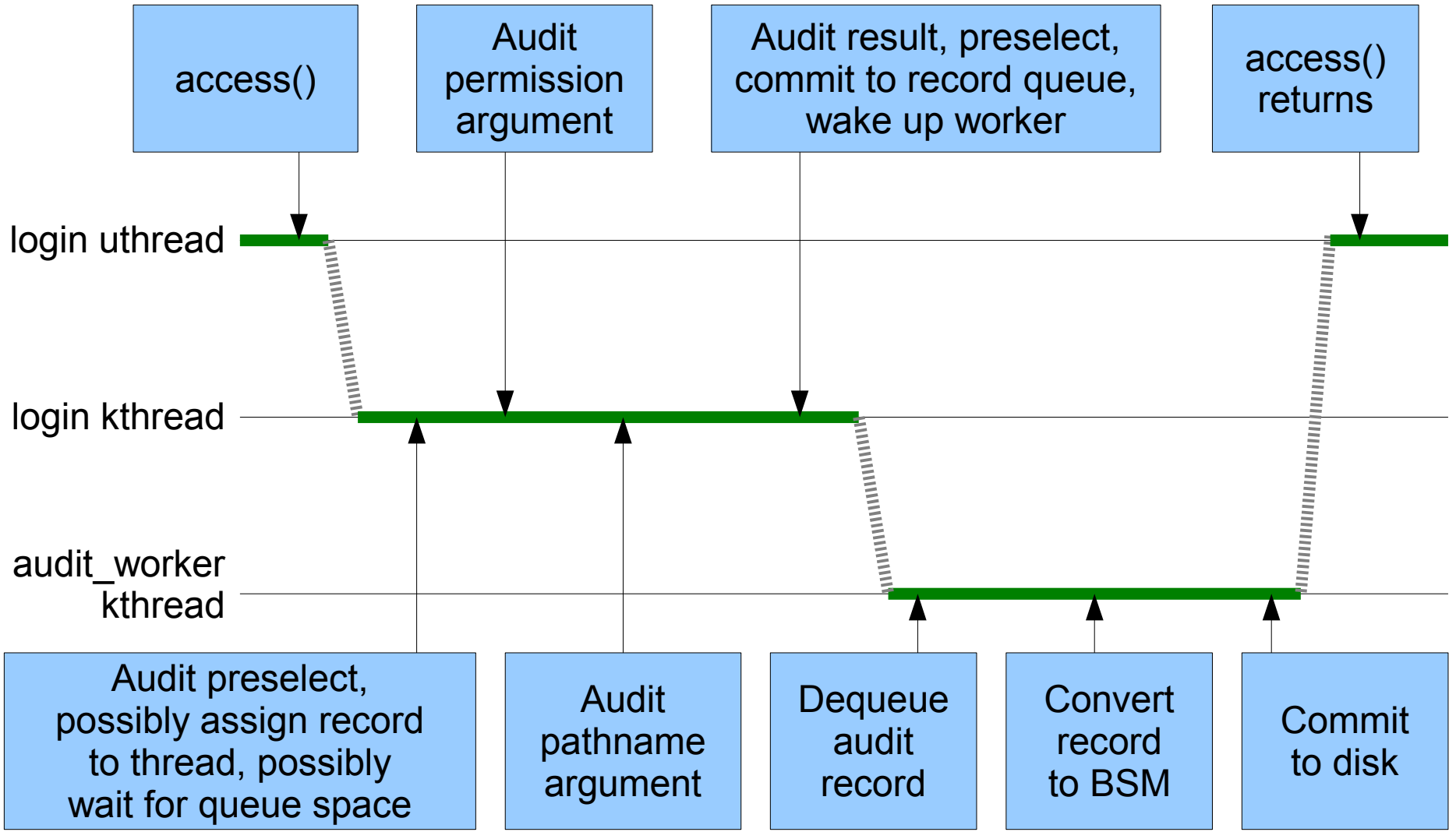
- Over 350 event types
  - Most of them meaningless individually
- Each event assigned to one or more classes
- Class masks assigned to users

```
0:AUE_NULL:indir system call:no
1:AUE_EXIT:exit(2):pc
2:AUE_FORK:fork(2):pc
3:AUE_OPEN:open(2) - attr only:fa
4:AUE_CREAT:creat(2):fc
5:AUE_LINK:link(2):fc
6:AUE_UNLINK:unlink(2):fd
7:AUE_EXEC:exec(2):pc,ex
8:AUE_CHDIR:chdir(2):pc
...
```

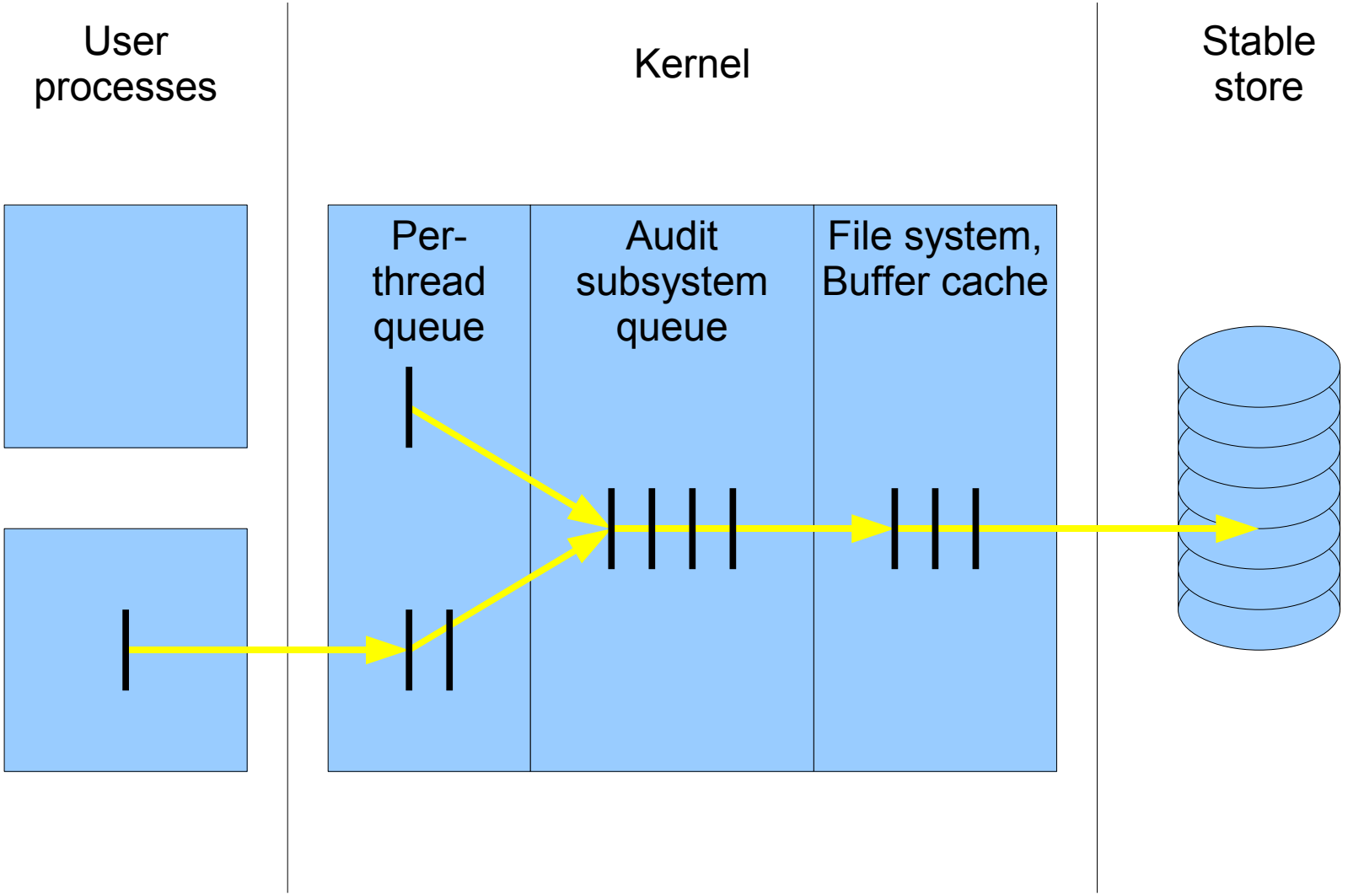
```
0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
0x00000004:fa:file attribute access
0x00000008:fm:file attribute modify
0x00000010:fc:file create
0x00000020:fd:file delete
0x00000040:cl:file close
0x00000080:pc:process
0x00000100:nt:network
...
```

```
root:lo:no
audit:lo:no
test:all:no
www:fr,nt,ip:no
...
```

Sample Audit Control Flow

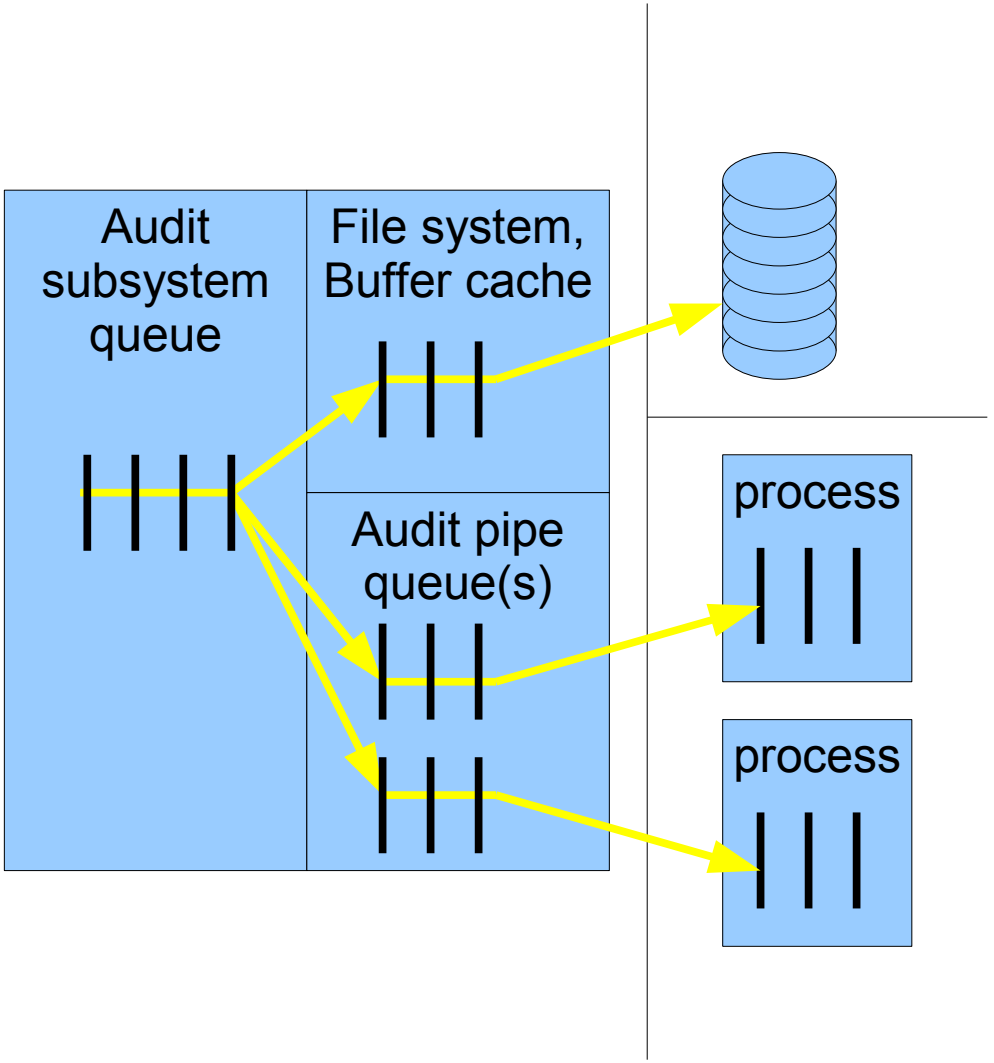


Audit Queuing



Audit Pipes

- Historically, audit for post-mortem analysis
- Today, for intrusion detection / monitoring
- Live record feed
  - Lossy queue
  - Discrete audit records
  - Independent streams
  - Interest model



Tools, Setup, Etc

- Setup
  - Compile in “options AUDIT”
  - Set auditd_enable=”YES” in rc.conf
  - Global settings: /etc/security/audit_control
  - Per-user settings: /etc/security/audit_user
- Management
  - Print audit trails: praudit
  - Redice audit trails: auditreduce
- See handbook chapter, man pages for details

Some Kernel Details

- Global kernel queue, worker thread
- System call code allocates, commits record
- System calls capture arguments
- New system calls
- New proc/thread state

```
struct thread {
    ...
    struct kaudit_record *td_ar;
    ...
}

struct proc {
    ...
    struct auditinfo p_au;
    ...
}
```

Sample System Call: chmod(2)

```
14     AUE_MKNOD        STD      { int mknod(char *path, int mode, int dev); }
15     AUE_CHMOD        STD      { int chmod(char *path, int mode); }
16     AUE_CHOWN        STD      { int chown(char *path, int uid, int gid); }
```

```
int
kern_chmod(struct thread *td, char *path, enum uio_seg pathseg, int mode)
{
    int error;
    struct nameidata nd;
    int vfslocked;

    AUDIT_ARG(mode, mode);
    NDINIT(&nd, LOOKUP, FOLLOW | MPSAFE | AUDITVNODE1, pathseg, path, td);
    if ((error = namei(&nd)) != 0)
        return (error);
    vfslocked = NDHASGIANT(&nd);
    NDFREE(&nd, NDF_ONLY_PNBUF);
    error = setfmode(td, nd.ni_vp, mode);
    vrele(nd.ni_vp);
    VFS_UNLOCK_GIANT(vfslocked);
    return (error);
}
```

System Call Audit Principles

- Assign audit event type in syscalls.master (etc)
- Argument data stored in thread's kaudit_record
  - AUDIT_ARG(type/entry, value)
  - NDINIT() flags
  - kaudit_record has storage for various types
  - Bitmask flags for each entry
  - New types may have to be added
- Converted to BSM in audit worker thread

Coordinating OpenBSM/Solaris/...

- Desirable to remain compatible with Solaris, Mac OS X if possible
- OpenBSM in contrib, maintained in p4
  - Event number allocation
  - Selection of arguments to audit
  - Changes in token stream format
  - New user space APIs

Userland Auditing

```
if (audit_submit(AUE_su, auid, 1, EPERM,
    "bad su %s to %s on %s", username, user, mytty))
        errx(1, "Permission denied");
```

- Security-relevant tools should audit
  - Currently, login, su, sshd, and some others do
  - Requires root privilege
- Two API choices
  - Constructed audit records using audit_open(3)
  - Use audit_submit(3) to generate a generically structured audit record

TODO

- Finish syscall assignments, especially for ABIs
- Flesh out arguments
- Audit + NSS
- Userland sweep
- Ports/packages
- Language bindings

- Enhance pipe preselection
- Multiple pipelines
- IDS/monitoring
- Distributed audit
- New parsing API

Kernel Privilege API priv(9)

- Decompose UNIX security model
  - UNIX process model
  - Mandatory inter-user protections
  - Discretionary access control
  - Privilege model
  - User model layered over kernel protections
- Privilege is the right to violate other policies
  - Historically granted to processes with effective uid 0
  - Scoped by secureleval, jail, MAC, etc.

Replacing the Privilege API

- Existing privilege checks in the kernel
  - error = suser(td)
  - error = suser_cred(cred, flags)
- Reasons to replace suser(9)
  - Offer finer granularity to decision code
  - Allow auditing of privilege by type
  - Virtual image privilege masks
  - Centralize jail, securelevel policies
  - Allow flexibility to configure, extend

priv(9)

- Replace all instances of suser(9) with priv(9)
  - priv_check(td, priv)
  - priv_check_cred(cred, priv, flags)
- Priv is a named privilege
  - PRIV_VFS_READ, PRIV_NETINET_RESERVEDPORT, ...
- Long list of named privileges by subsystem
- Eventual goal of removing jail privilege flag
- MAC can now instrument privilege decision

Where Do We Go From Here?

- Do not add new calls to suser(9) or suser_cred(9)
- Where it makes sense, use an existing privilege
- Where it doesn't, add a new privilege
- When allowing or disallowing in jail, for now use both SUSER_ALLOWJAIL and add to the kern_jail.c:prison_priv_check() switch
- Help sweep up remaining calls to suser(), look at the XXX comments first