Trusted Operating System Architecture

Agenda

1. Introduction to FreeBSD/TrustedBSD
2. Security Event Auditing Framework
3. Host-based intrusion detection
4. Mandatory Access Control Framework
5. Demonstration

What is FreeBSD? (short version)

- Advanced operating system which runs on many hardware architectures
- To name a few: x86, amd64, Sparc64, PPC, ARM
- Derived from BSD, the version of UNIX developed at the University of Berkeley

What is TrustedBSD?

- Set of trusted operating system extensions to the FreeBSD operating system
- Targets Common Criteria (CC) for information technology security evaluation
- Sponsored by a number of organizations:
  - Defense advanced research projects agency (DARPA)
  - National Security Agency (NSA)
  - Apple Computer

TrustedBSD Projects

- Mandatory Access Control Framework and modules:
  - Flask/TE
  - BIBA
  - MLS
- Security Event auditing framework
- Extended file system ACLs (POSIX.1e [spec. deprecated])
- www.trustedbsd.org

Continued…

- Disk encryption for cold storage
- Capability/privilege model
- Intrusion detection
- more…

What We will be Covering

- Audit
- Intrusion detection
- Mandatory Access Control (MAC) framework

First, Some Problems

- Authorization:
  - Most operating systems have two privileges: root/admin and regular users
  - Privilege is often required to do useful things
  - Compromise of the privileged account generally results in the system being completely untrusted

More Problems…

- Security event logging:
  - UNIX syslog is very popular but, ultimately, completely inadequate for security
  - Syslog offers no reliability; in fact in many cases syslog(3) doesn't even return a value
  - Anyone can submit syslog messages at any facility and priority… and as any process
  - Network syslog… don't even get me started (no confidentiality, no integrity, and based on the most *spoofable* protocol [UDP])

Continued…

- Security event logging
  - Granularity in filtering
    (unless you are amazing with grep and regular expressions ☺)

More Problems…

- Security notification engines…
  Wait, do they even exist?
  - Operating systems attempt to give logs when something “relevant” has happen
  - They allow us set permissions (in most cases)
  - But, generally, provide no mechanism for notification if there is a violation of policy

So, Where are We Today?

- As security practitioners, we use potentially forged data for investigations, intrusion detection, etc.
- We accept the fact that under resource pressure, certain events might not be logged
- When systems are compromised, we assume the entire system is un-trusted, format, and reinstall

Security Event Auditing

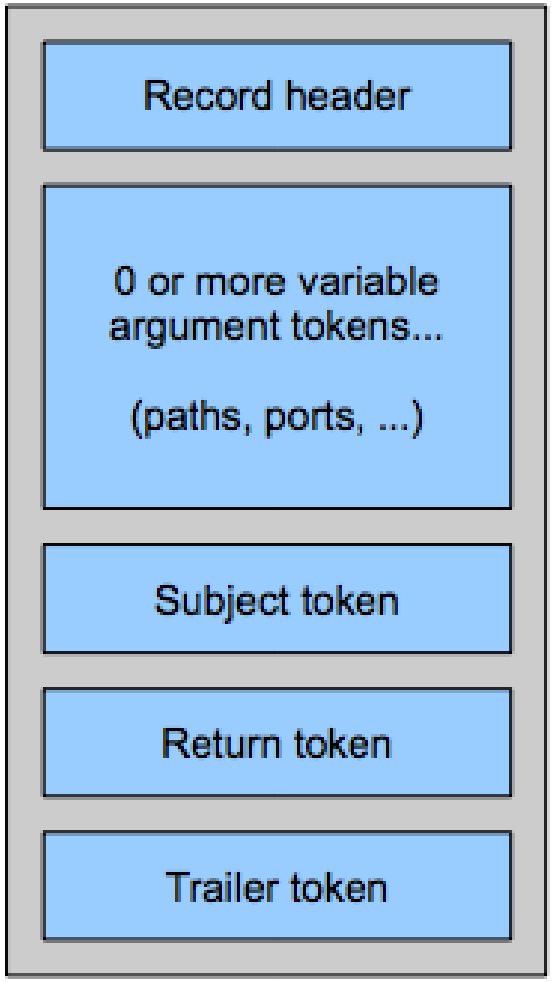
Security Auditing Framework

- Goals (Mostly CC/CAPP driven):
  - Ensure that only privileged or authorized subject can submit audit trails
  - Must be an upper bound on loss, in the event of a power failure
  - If an auditable event can not be audited, then it cannot happen
  - Must be able to select which events get audited

Audit Records

- Security auditing framework produces records
- Modeled the record format after the Sun Basic Security Module (BSM)
- Allows pre-existing tools, APIs, and others to work with the TrustedBSD auditing system
- TrustedBSD implementation adopted by Apple Computer

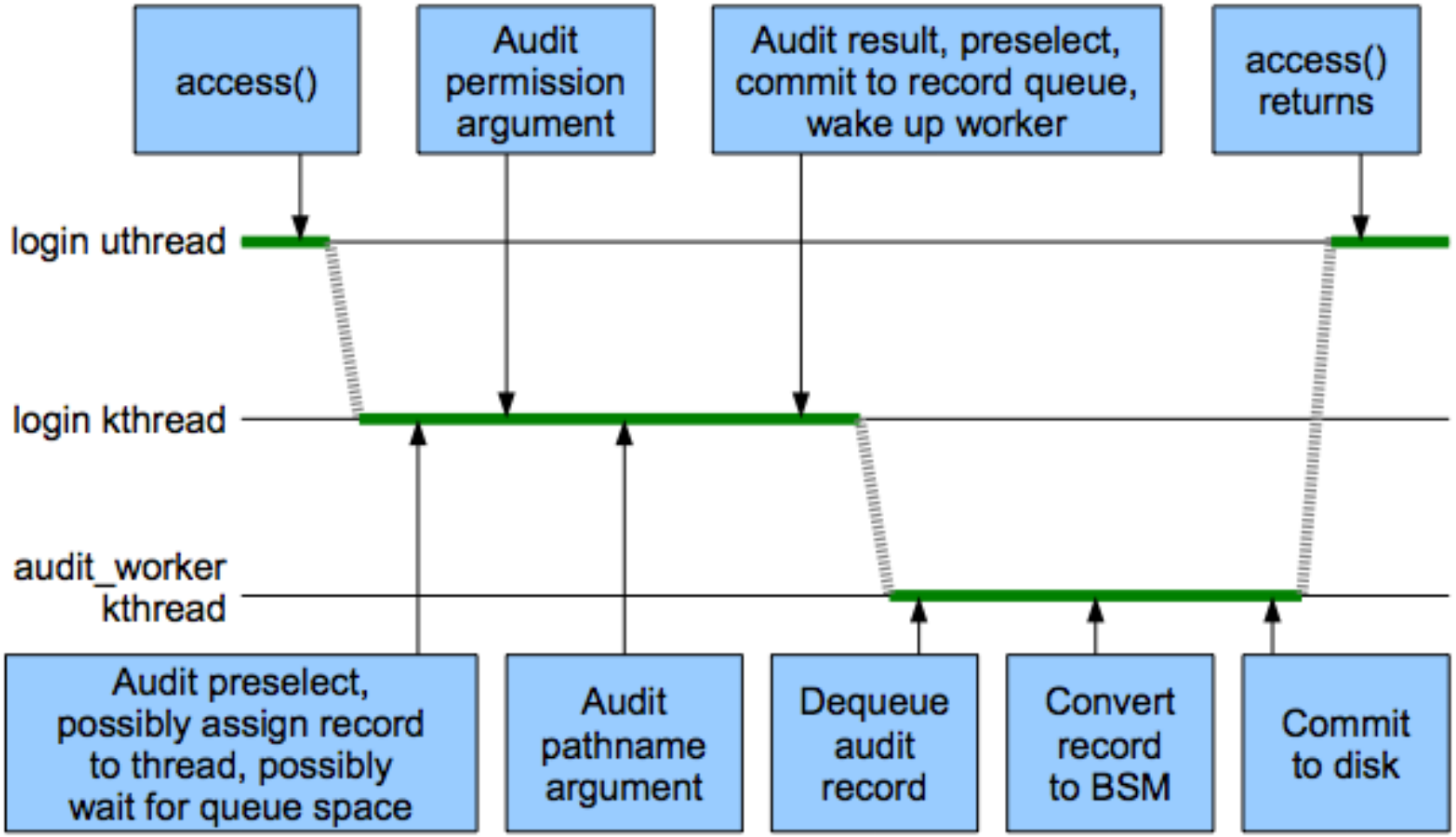
Record Format (BSM)



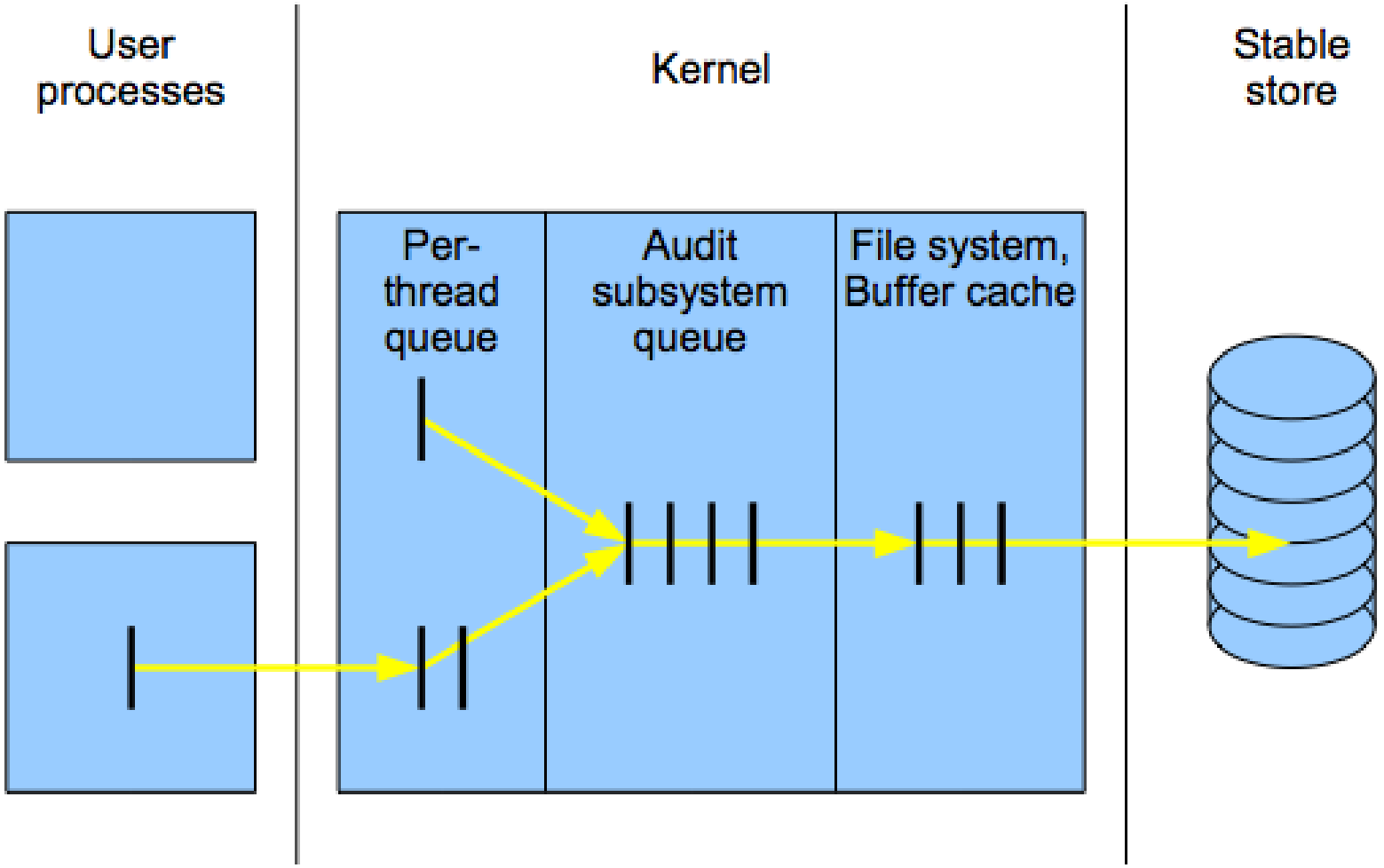

```
header,129,1,AUE_OPEN_R,0,Tue Feb 21 00:12:23 2006, +
253 msec
argument,2,0,flags
path,/lib/libc.so.6
attribute,444,root,wheel,16842497,11663267,46706288
subject,-1,root,wheel,root,wheel,319,0,0,0.0.0.0
return,success,6
trailer,129

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
```

Record Queuing



Audit Queuing



So, What gets Audited?

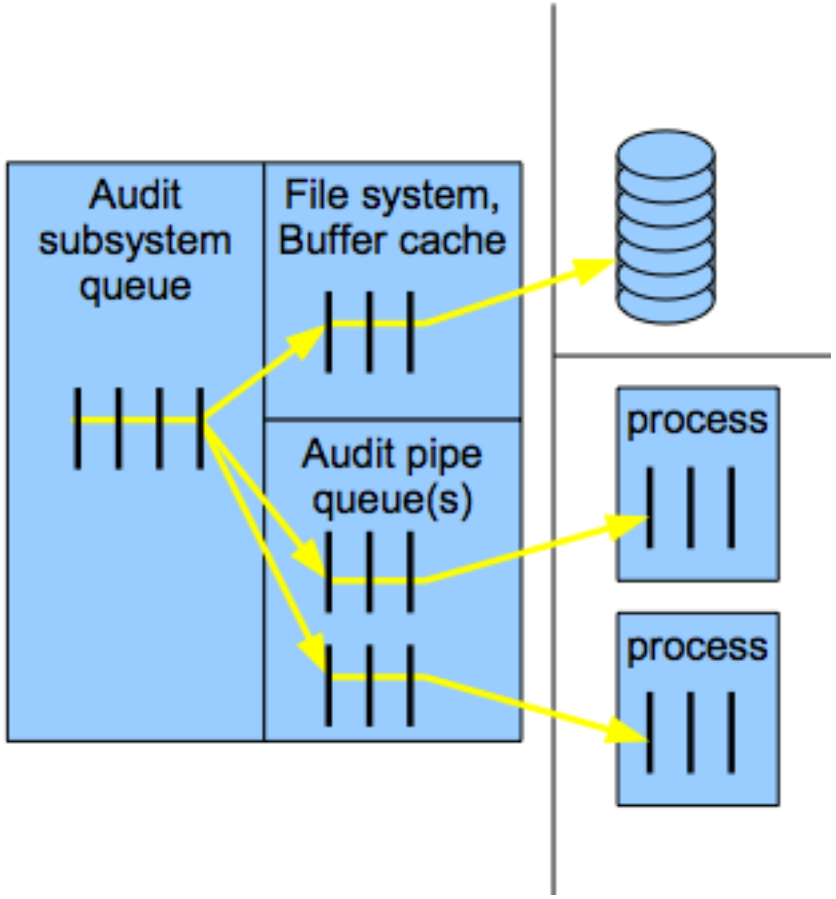
- …a lot. There are over 500 event types.
  - Kernel generated events, file access, execution, etc.
  - Userspace events: manipulating password database, escalating privilege with things like su
  - Audit events are mapped to one or more audit classes: file read, write, administrative, etc.
  - Pre-selection occurs on the class level

Record Selection

- Potential for audit record volume is huge
  - Terabytes per hour on busy, fully-audited systems
- Two points for record selection
  - Pre-selection: before the audit record is created
  - Post-select: reduction on an existing audit trail

Real Time Audit Feed

- Some cases: processes (e.g., IDS) may want real time feeds
- Adjust pre-selection on the fly
- Reduce auditing overhead



Audit Driven Host-Based IDS

Audit-Driven IDS

Goals:

- Reduce false positives
- Ensure intrusion decisions are based on legit data
- Ensure that the IDS is capable of seeing all relevant events
- Enable security administrators to define “sequences” of events which require escalation

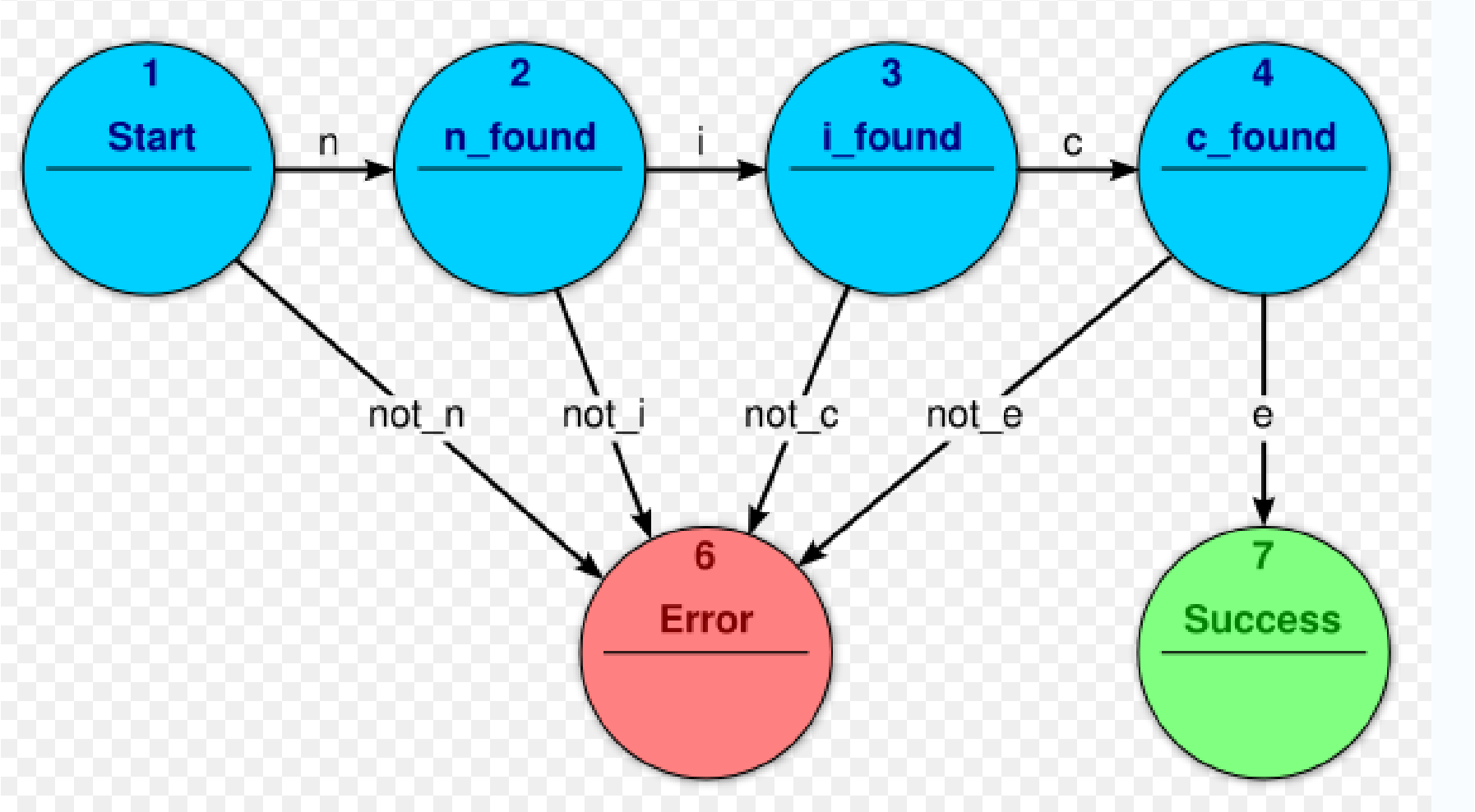
BSMTRACE

- Host-based intrusion detection (HIDS) based on finite state machine principles
- Uses BSM audit records as an input
- Uses audit pipes to tap into real audit record feeds
- Audit trail (files) supported, making it possible to operate on Solaris and Apple OS X

Continued…

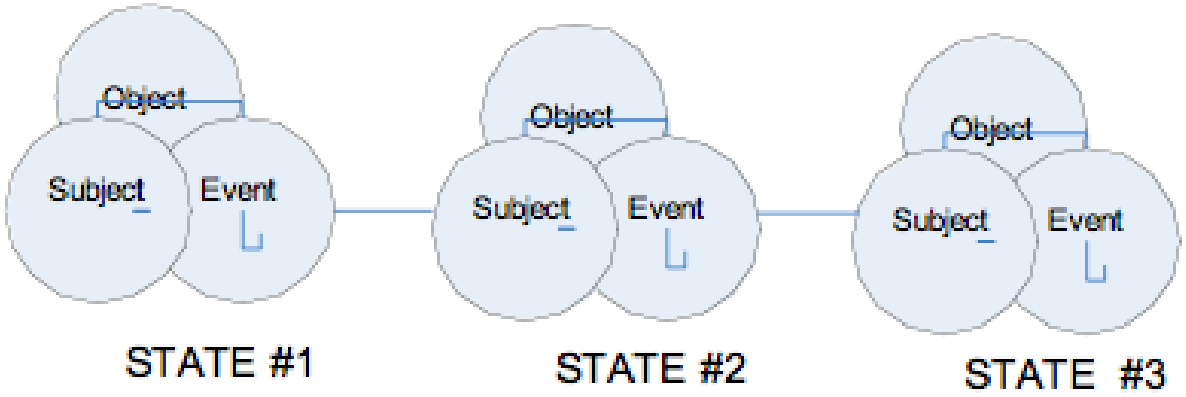
- Observes host behavior and makes decisions on events that have actually occurred
- Fundamental difference:
  NIDS often has no context
- Don’t know which OS/software is running
- Don’t understand vulnerabilities, etc.

Finite State Machines 101

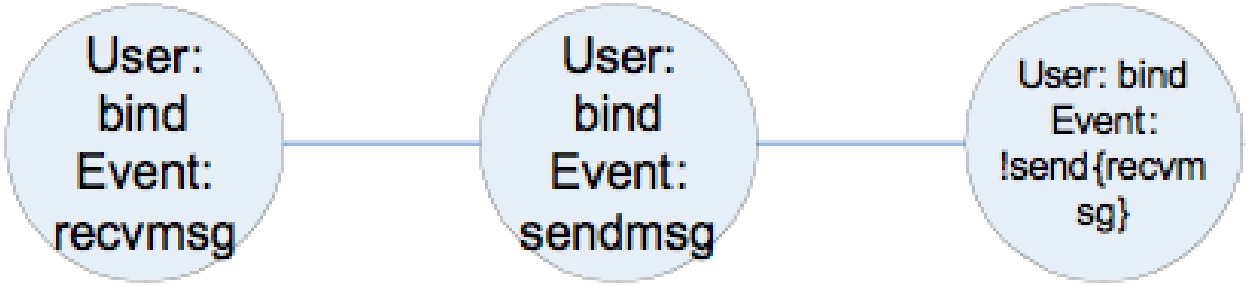


BSMtrace States in Detail

- "States" in detail



- Example sequence:



Sample State Machine

```
sequence named.exec {
        subject <auid> { bind; };
        state {
                event <auditevent> { AUE_SOCKET; };
                status success;
        };
        state {
                event <auditevent> { AUE_BIND; };
                status success;
        };
        state {
                event $execution;
                status any;
        };
};
```

HIDS: Legacy problem

- But wait… Something seems wrong with trusting a machine to tell you when it has been compromised
- What if the HIDS process gets compromised?
- How can we compartmentalize untrusted processes?

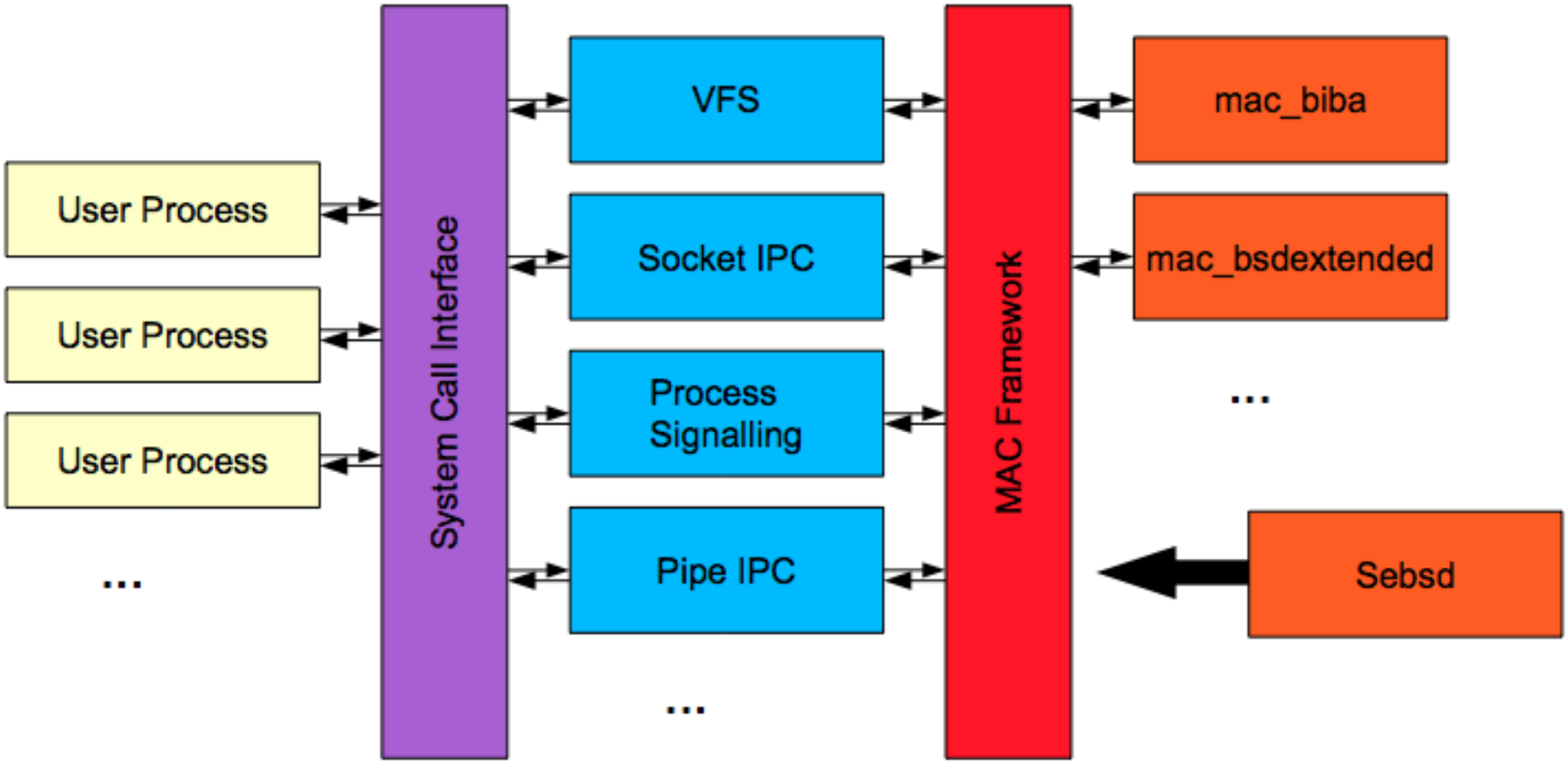
Mandatory Access Control (MAC) Framework

Mandatory Access Control Framework

High-level Goals:

- Develop a policy-independent access control and authorization framework for the OS
- Hook every point in the kernel that a security decision needs to be made
- Ensure security policy components plug in as a module to the framework
- Have the ability to label both subjects and objects in a policy-independent fashion

Implementation Details



Why the Abstraction?

- What is the right policy?
- Other operating systems force their notion of policy on their users
- In some environments, MLS might make more sense then BIBA, and in others Flask/TE (SeLinux)

MAC Entry Points

- Access control checks scattered around the kernel in (roughly) over a hundred places
  - File access, system V IPC creation, etc.
- Other entry points mostly assist in label management: initialization, copying, and destruction of label storage

Why all the Entry Points?

But other security mechanisms hook in the syscall layer and inspect there

- BAD – Most modern operating systems support concurrency
- Race conditions can make TOCTOU issues very serious
- TrustedBSD operates on the vnode not the path, as an example

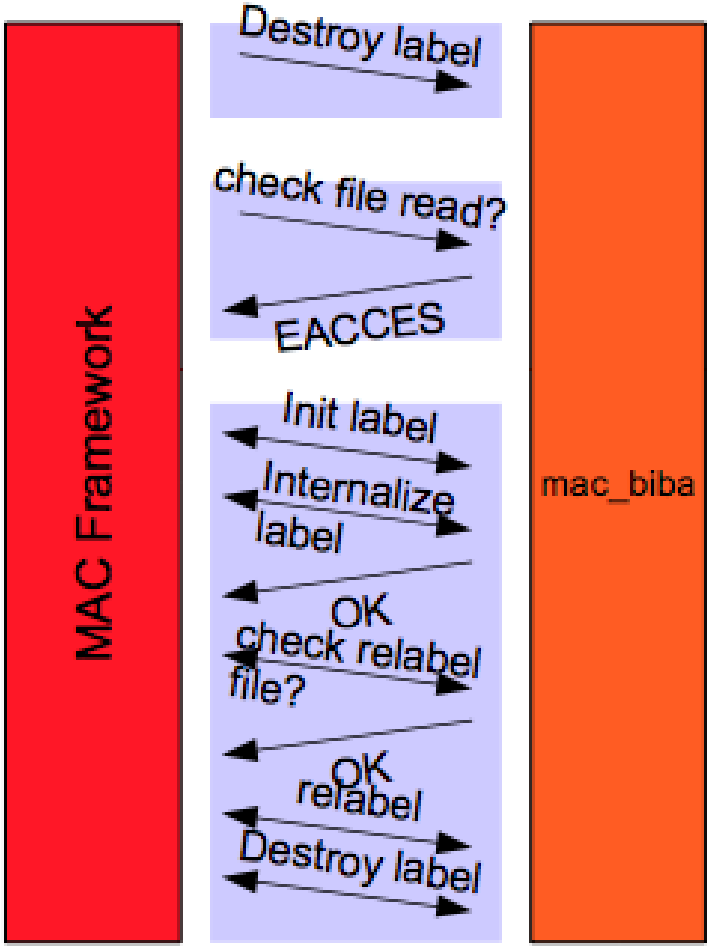
Object Labels

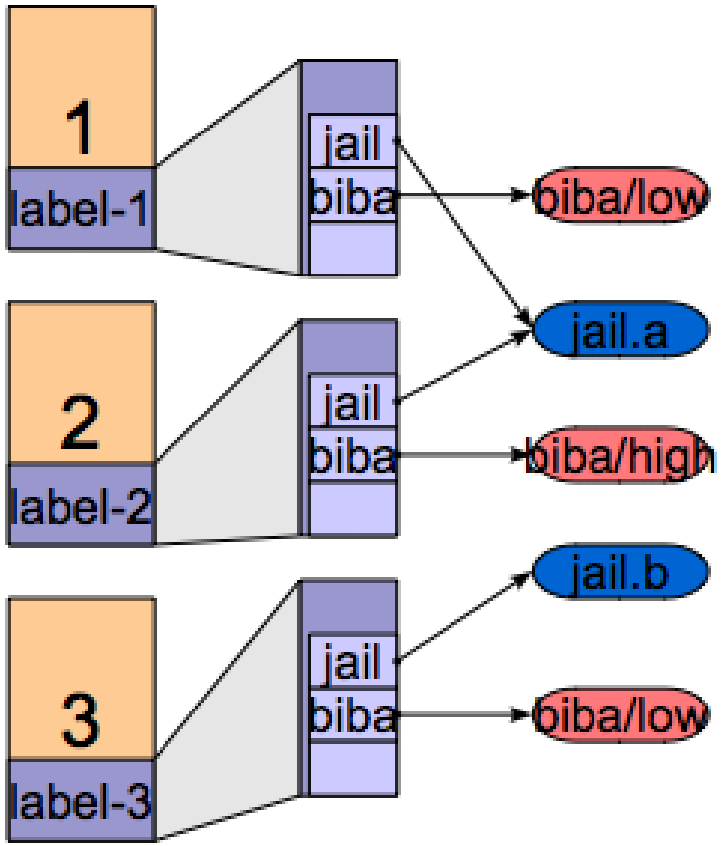
- What is a label?
  It’s something that gets associated to all users (subjects) and objects
- Basically, what the security policy uses to make an authorization decision

Which Objects get Labels?

- Pipes
- Sockets
- Files/vnodes
- Mbufs (packets)
- Network interfaces
- System V IPC objects

Policy Agnostic Labels





Questions?