FreeBSD Advanced Security Features

Robert N. M. Watson

Security Research
Computer Laboratory
University of Cambridge

19 May, 2007
Introduction

• Welcome!
  – Introduction to some of the advanced security features in the FreeBSD operating system

• Background
  – Introduce a series of access control and audit security features used to manage local security
  – Features appeared between FreeBSD 4.0 and FreeBSD 6.2, and build on the UNIX security model
  – To talk about new security features, we must understand the FreeBSD security architecture
Post-UNIX Security Features

- Securelevels
- Pluggable authentication modules (OpenPAM)
- Crypto library and tools (OpenSSL)
- Resource limits
- Jails, jail securelevels
- GBDE, GELI
- IPFW, PF, IPFilter
- KAME IPSEC, FAST_IPSEC
- Access control lists (ACLs)
- Security event audit
- Mandatory access control (MAC)
- 802.11 security
Brief History of the TrustedBSD Project

- TrustedBSD Project founded in April, 2000
  - Goal to provide trusted operating system extensions to FreeBSD
  - DARPA funding began in July, 2001
  - Continuing funding from a variety of government and industry sponsors
  - Work ranges from immediately practical to research
  - While many of these features are production-quality, some are still under development
  - Scope now also includes Apple's Mac OS X
FreeBSD Security Architecture
FreeBSD Security Architecture

- FreeBSD's security architecture is the UNIX security architecture
  - Entirely trusted monolithic kernel
  - UNIX process model
  - Kernel UIDs/GIDs driven by user-space user mode
  - Privileged root user
  - Various forms of access control (permissions, ...)
- Security features discussed here extend this security model in a number of ways
Kernel and User Processes

File system access

system call

Inter-process communication

User Process

User Process

User Process

User Process

...
Security Architecture:
Kernel Access Control Policy

- Objects owned by a user and group
- Mandatory inter-user protections
  - No inter-user process control (debugging, ...)
  - Only owner of an object can control its protections
  - Special protections for setuid, setgid processes
- Discretionary protections
  - File permissions and ACLs allow owner to grant specific rights to other users and groups
  - Used to protect both system and user data
Security Architecture: The User-space Security Model

- Low-level kernel primitives provide foundation:
  - Process isolation
  - Process credentials and privilege
  - Privilege escalation through setuid/setgid
  - Object ownership and access control

- No mention of password files, logging in, remote access, home directories, etc.
  - All implemented as a user-space software layer using kernel primitives
Security Architecture:
Authentication and Remote Access

• Kernel provides low-level networking primitives
  – Concepts such as telnet, SSH entirely in user-space
  – Map network I/O into simulated tty input

• User authentication entirely in user-space
  – Pluggable Authentication Modules (PAM) invoked by remote access daemons
  – Kernel UIDs and GIDs set by daemon at login
  – Hence cache consistency issues between /etc files and running kernel state
Security Architecture: Conclusion

• Layered UNIX security architecture
  – Kernel provides low-level process, process credential, and system services
  – User-space libraries and tools implement users, authentication, remote access

• Security features we discuss will extend this basic functionality
  – Increased functionality
  – Increased flexibility
Access Control Lists
Access Control Lists (ACLs)

- Extend UNIX file permissions
  - Allow flexible assignment of rights by and for users
  - Required by Orange Book C2, CC CAPP
- ACLs supported in most operating systems
  - POSIX.1e ACLs (Solaris, IRIX, FreeBSD, Linux)
  - NT ACLs (Windows, NFSv4, Mac OS X, ZFS)
- FreeBSD UFS implements POSIX.1e ACLs
  - NT ACL mapping provided by Samba
Configuring UFS2 ACLs

- UFS2 ACLs stored in extended attributes
- Compile UFS ACL support into kernel
  - options UFS_ACL
  - Enabled by default in GENERIC kernel
- ACLs must be administratively enabled for each file system they will be used with
  - tunefs -a enable
  - File system must be unmounted or mounted read-only (best done from single-user mode)
UNIX File Permissions

• Permission mask in file mode
  – Assigns rights to file owner, group, and other
  – Possible rights: read, write, execute

• Certain other special bits in file mode
  – setuid: process takes on UID of file when executing
  – setgid: process takes on GID of file when executing
  – sticky bit limits unlink rights in directory (/tmp)

• Expressiveness of file permissions very limited
  – Only administrator can modify group membership
POSIX.1e ACLs

- Allow file owner to assign rights for additional users and groups
  - UNIX permissions for owner, group, other
  - POSIX.1e ACL entries assign rights for for additional users and additional groups
  - Directories have an optional default ACL
    - Set ACLs on new files or sub-directories in subtree
- POSIX.1e provides a mask ACL entry to support file mode compatibility for applications
# Example ACL

<table>
<thead>
<tr>
<th>User</th>
<th>Group</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw-</td>
<td>r--</td>
<td>---</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th>Group</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>[owner] rw-</td>
<td>[owner] r--</td>
<td></td>
</tr>
<tr>
<td>robert rw-</td>
<td>www r--</td>
<td></td>
</tr>
<tr>
<td>Mask rw-</td>
<td>Other ---</td>
<td></td>
</tr>
</tbody>
</table>
Example ACL

• One file with only a basic ACL (UNIX permission mask)
• One file with an extended ACL
  – One additional user
  – One additional group
  – Mask granting at most read/write to groups and additional users

```bash
example
  cinnamon% getfacl without_acl
  #file:without_acl
  #owner:0
  #group:0
  user::rw-
  group::r--
  other::---

example
  cinnamon% getfacl with_acl
  #file:with_acl
  #owner:0
  #group:0
  user::rw-
  user:robert:rw-
  group::r
  group:www:r--
  mask::rw-
  other::---
```
ACLs on Newly Created Files and Directories

- Directories have *access* and *default* ACLs
- If the parent directory has only a basic ACL, UNIX creation rules apply
- If the parent directory has a default ACL, special creation rules apply:
  - Access ACL of child will be default ACL of parent masked by requested creation mode and umask
  - New subdirectories inherit parents' default ACL
ACL Tools

• Modifications to existing commands
  – mount(8) – Show when ACLs are enabled
  – ls(1) – Show when an ACL is present with “+”
  – tar(1) – Back up and restore ACLs on files

• New ACL commands
  – getfacl(1) – Retrieve the ACL on one or more files
  – setfacl(1) – Set the ACL on one or more files
ACL Documentation

• Man pages
  – getfacl(1), setfacl(1)

• FreeBSD Handbook chapter “File System Access Control Lists”
ACL Conclusion

- Access control lists add greater flexibility to UNIX file protection model
  - Users can assign rights to other users and groups
  - Avoid the necessity for administrative involvement with users collaborate
  - Available in all FreeBSD versions with UFS2
  - Backwards compatible with UNIX permissions
  - Portable to other UNIX operating systems
Security Event Auditing
Security Event Auditing

- Auditing logs system security events
  - Secure, reliable, fine-grained, configurable
- A variety of uses including
  - Post-mortem analysis
  - Intrusion detection
  - Live system monitoring, debugging
- Required by Orange Book, Common Criteria CAPP evaluations
- Found in most commercial UNIX systems
Audit Logs, Records, and Events

- Audit log files are called “trails”, contain records
- 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
What events can be audited?

- 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 administration events
FreeBSD Security Event Auditing Architecture

- Audit records describe security events
- Audit records managed by kernel audit engine
- Audit daemon manages trails, configuration
- Sun's BSM audit trail file format and API
- Administrators control granularity of logging
- Kernel and privileged processes may submit records to audit trail
- UNIX DAC permissions protect audit log
BSM Audit Record Format

<record version="10" event="OpenSSH login" modifier="0" time="Fri May 18 04:19:56 2007" msec="274" >
<subject audit-uid="robert" uid="robert" gid="robert" ruid="robert" rgid="robert" pid="44835" sid="44835" tid="42666 24.114.252.226" />
<text>successful login robert</text>
<return errval="success" retval="0" />
</record>

<record version="10" event="execve(2)" modifier="0" time="Fri May 18 07:04:15 2007" msec="933" >
<exec_args><arg>pine</arg></exec_args>
<path>/usr/local/bin/pine</path>
<attribute mode="555" uid="root" gid="wheel" fsid="90"
nodeid="71201" device="336464" />
<subject audit-uid="robert" uid="robert" gid="robert" ruid="robert" rgid="robert" pid="51933" sid="51927" tid="49811 24.114.252.226" />
<return errval="success" retval="0" />
</record>
Audit Selection

- Potential for audit record volume huge
  - Terabytes/hour on busy, fully audited system

- Two key points for audit record selection
  - Audit pre-selection to limit audit records created
  - Audit post-selection, or reduction, to eliminate undesired records after creation

- FreeBSD support both models
  - Administrator can apply filters to users at login time
  - Administrator can use tools to reduce trails later
Audit Trail Reduction

- Reduction selects records from audit trails
  - E.g., for long-term archiving or immediate inspection

- `auditreduce(8)` accepts a trail file as input, and generates a reduced trail stream as output
  - Criteria for record selection include user ID, date or time of event, type of event, or affected object

- Can output to a file or create a pipeline
  - Reducing a large audit trail to just login/logout data
  - Piping output to `praudit(8)` for printing
Audit Pipes

- Historically, audit for post-mortem analysis
- Today, for intrusion detection / monitoring
- Audit pipes provide live record feed
  - Lossy queue
  - Discrete audit records
  - Independent streams
Audit Documentation

- Extensive man pages
  - audit(4), auditpipe(4)
  - audit(8), auditreduce(8), praudit(8), auditd(8)
  - audit.log(5), audit_control(5), audit_user(5), ...


- TrustedBSD audit implementation paper: “The FreeBSD Audit System”
Audit: Conclusion

- Powerful tool for tracking and monitoring system use
- Fine-grained, reliable, and secure logging of user activity
- Now available in FreeBSD 6.2 as an experimental feature
- Will be a production feature in FreeBSD 6.3 as functionality matures
Mandatory Access Control (MAC)
Mandatory Access Control (MAC)

- Administrator defines mandatory rules under which users and processes interact
  - Contrast with Discretionary Access Control (DAC)
  - File ACLs protect files at the discretion of the owner
- Historically, Multi-Level Security (MLS)
  - Data is labelled with sensitivity levels/compartments to indicate what protection is required
- Recently, much more broad definition
  - “Mandatory” as opposed to a specific policy
TrustedBSD MAC Framework

- Kernel framework that allows policy modules to modify the kernel access control policy
  - Add new constraints
  - Track use of resources
  - Attach security labels to objects
- Two general common classes of policies
  - Ubiquitous information labelling policies
  - Hardening policies
MAC Policies for FreeBSD

• FreeBSD has a number of sample policies
  – Labelled: Biba, MLS, LOMAC, partition
  – Hardening: portacl, seeotheruids, ugidfw

• Several open source third party policies
  – Cryptographically signed binaries
  – SEBSD (SELinux FLASK/TE)
  – mac_privs

• Several third-part policies built into products
BSD Extended User/Group File System Firewall

• Rule-based file system protection policy
  – Module name: mac_bsdextended
  – Kernel option: options MAC_BSDEXTENDED

• Implements a file access “firewall” rules
  – ugidfw(8) management tool is similar to ipfw(8)
  – Administrator can use rules to restrict access by user or group
  – Overrides normal file permissions and ACLs

• No data or subject labelling is required
User/Group File System Firewall

- ugidfw(8) command manages a rule list similar to that in network firewalls

- Override permissions that would otherwise grant rights denied by the firewall policy

- ugidfw set 100 subject uid www object uid robert mode rxs
  - Deny any access but read, execute, and stat by user www on objects owned by user robert
MAC Documentation

- FreeBSD man pages
  - mac(4)
  - getfmac(8), setfmac(8)
- FreeBSD Handbook chapter, “Mandatory Access Control (MAC)”
- TrustedBSD implementation papers
  - “The TrustedBSD MAC Framework: Extensible Access Control for FreeBSD 5.0”
  - “Design and Implementation of the TrustedBSD MAC Framework”
Conclusion

- Introduction to FreeBSD Security Architecture
- Several advanced FreeBSD security features
  - ACLs
  - Audit
  - MAC
- Further information can be found in:
  - The FreeBSD Handbook
  - http://www.TrustedBSD.org