Native Host Intrusion Detection with RHEL6 and the Audit Subsystem

Steve Grubb
Red Hat
Introduction

- How the audit system works
- How we can layer an IDS/IPS system on top of it
Introduction

- Designed to meet or exceed audit requirements of:
  - CAPP, LSPP, RSBAC, NISPOM, FISMA, PCI-DSS, STIG
- Evaluated by NIAP and BSI
- Certified to CAPP/EAL4+ on RHEL4
- Certified to LSPP/CAPP/RSBAC/EAL4+ on RHEL5
- Under evaluation for OSPP/EAL4+ on RHEL6
Introduction

Some of the requirements for the audit system:

- Shall be able to record at least the following
  - Date and time of event, type of event, subject identity, outcome
  - Sensitivity labels of subjects and objects
  - Be able to associate event with identity and login of user causing it
  - All modifications to audit configuration and attempted access to logs
  - All use of authentication mechanisms
  - Changes to any trusted database
  - Attempts to import/export information
  - Be able to include/exclude events based on user identity, subject/object labels, other attributes
Audit User Tracking
Audit User Tracking

Init
Uid 0
Auid -1
Ses -1

Sshd
Uid 0
Auid -1
Ses -1

Sshd
Uid 0
Auid -1
Ses -1

pam_loginuid.so

Kernel

CAP_AUDIT_CTL && same PID
Audit User Tracking

- **Init**
  - Uid 0
  - Auid -1
  - Ses -1

- **Sshd**
  - Uid 0
  - Auid -1
  - Ses -1

- **Sshd**
  - Uid 0
  - Auid 500
  - Ses 1

- **pam_loginuid.so**

- **Kernel**

- `CAP_AUDIT_CTL && same PID`
Audit User Tracking
Kernel

- Designed to minimize the performance impact as little as possible
- Relies on a flag, TIF_SYSCALL_AUDIT, which is part of the thread’s information flags variable.
- Flag is inherited at fork when audit_enabled is true
- Flag is never reset
- If you need audit of all processes, you must use audit=1 as a boot parameter.
Kernel – audit flag inheritance

- Fork
- audit_enabled
- Task Filter
- Audit state != DISABLED
- Create context
- Copy known attributes
- Pid, Auid, Uid, Session, Subject, Comm
- Arch, Syscall, Time, Key, Ppid
Kernel – syscall entry

- Kernel syscall entry
- TIF_SYSCALL_AUDIT
- current->audit_context
- audit_enabled
- Collect audit info
- Arch
  SySCALL number
  SySCALL arguments
  Time
Kernel

- Need to decide if the syscall excursion is of interest
- Audit context has a state variable: DISABLED, BUILD, and RECORD
- Filters decide if event is interesting
  - Exit
  - Task
  - User
  - Exclude
The diagram illustrates the flow of communication between an application (App) and a kernel (KERNEL). The application sends requests to the kernel via the 'App' block, which are processed by the 'User' node, followed by the 'Entry' node. These requests are then handled by the 'syscall magic' module before being routed to the 'Task' node and finally to the 'Exclude' node. The 'Exclude' node sends messages to the 'Audit Daemon', which likely performs auditing or logging functions.
Kernel

• Syscall Exit
  - If context marked auditable emit event
  - Event can be multi-part
    • Ex. Message Queue attributes, IPC attributes, execve args, socket addr, socket call args, file paths, and current working directory.
  - All are tied together with time stamp and serial number
  - Free allocated resources
Audit rules are stored at `/etc/audit/audit.rules`

Audit rules are loaded by `auditctl`

Auditctl can control the kernel settings:

- `-e 0/1/2` disable/enable/enabled and immutable
- `-f 0/1/2` failure mode silent/printk/panic
- `-b 320` backlog (default too low for production use)
- `-r 0` event rate limit
- `-s` get status
- `-l` list all rules
- `-D` delete all rules
Syscall Rules

Follows the general form:
-a filter,action -S syscall -F field=value

Example to see failed opens for user 500:
-a exit,always -S open -S openat -F exit=-EPERM -F auid=500

-F can be one of: a0, a1, a2, a3, arch, auid, devmajor, devminor, dir, user/group ids, file type, inode, msgtype, object/subject context parts, path, personality, pid, ppid, or success.

Label can be applied with -F key=name

“and” created by adding more “-F” name/value pairs.
“or” is created by adding a new rule with same key value.
Per Task Audit Context

- Opaque pointer in task structure
- Contains
  - Time, serial number, syscall number, first 4 syscall arguments, exit code, array of file paths, credentials, arch, and data for auxiliary records, and internal house keeping data.
Kernel Filter

Filter Hook

Not the audit daemon

There are rules

For each Rule

Match

For each Field in Rule

Field \(!=\) context

Output Event

cont

brk
Audit Event

type=PATH msg=audit(10/11/2011 17:10:48.489:63) : item=0
name=/var/run/ inode=14909478 dev=08:07 mode=dir,755 ouid=root
ogid=root rdev=00:00 obj=system_u:object_r:var_run_t:s0


type=SYSCALL msg=audit(10/11/2011 17:10:48.489:63) : arch=x86_64
syscall=unlink success=no exit=-13(Permission denied) a0=439928 a1=0
a2=3f29b96600 a3=0 items=2 ppid=1 pid=1280 auid=unset uid=haldaemon
gid=haldaemon euid=haldaemon suid=haldaemon fsuid=haldaemon
egid=haldaemon sgid=haldaemon fsgid=haldaemon tty=(none) ses=unset
comm=halld exe=/usr/sbin/hald subj=system_u:system_r:halld_t:s0
key=delete
Kernel – File System Auditing

- Syscall auditing presents us with a problem when we need to monitor files
- Audit system does collect devmajor/minor information and inode
- But many interesting files are edited as temp copy and then replace original file
- This causes the inode to change
Kernel – File System Auditing

- Audit rules specified as a path and permission
- Kernel translates into inode rule
- When something replaces a watched file, inode rule updated in kernel
- Reconciliation is done by syscall exit filter
- Audit on directory is recursive to bottom of tree
- Mounted subtrees need additional rule added to include subtree in watch
- Limitations:
  - No wildcards for paths
  - If path specifies directory, it audits changes to dir entries
File System Audit Rules

File system audit rules take the general form of:
-w /full/path-to-file -p wrxa -k rule-note

Can also be expressed as syscall audit rule:
-a exit,always -F path=/full/path-to-file -F perm=wrxa -F key=rule-note

The perm field selects the syscalls that are involved in file writing, reading, execution, or attribute change.

Recursive directory audit for writes:
-a exit,always -F dir=/etc -F perm=wa -F key=rule-note
Trusted App Events

- Trusted apps can send events
- Must have CAP_AUDIT_WRITE
- Automatically included in audit trail, no rules needed
- Can be trimmed a little with USER or EXCLUDE filters.
TTY Auditing

- Security requirements ask for super user usage of the system
- Shell or tty based can be defeated or escaped
- Only good place to do this was from kernel
- Enable by adding pam_tty_audit.so to entry point's pam stack
- Both keystrokes and bash completions can be recorded
  - Depends on bash having CAP_AUDIT_WRITE
- Event is hex encoded ASCII – must use ausearch to read
- NOTE: DOES CAPTURE PASSWORDS!
Audit Event Type Classes

1000 - 1099 are for commanding the audit system
1100 - 1199 user space trusted application messages
1200 - 1299 messages internal to the audit daemon
1300 - 1399 kernel audit events (syscall / file system / TTY)
1400 - 1499 kernel SE Linux use
1600 - 1699 kernel crypto events
1700 - 1799 kernel anomaly records
1800 - 1899 kernel integrity labels and related events (IMA)
1900 - 2099 future kernel use
2100 - 2199 user space anomaly records
2200 - 2299 user space actions taken in response to anomalies
2300 - 2399 user space generated MAC events
2400 - 2499 user space crypto events (nss)
2500 - 2599 user space virtualization management events (libvirt)
2600 - 2999 future user space (maybe integrity labels and related events)
Audit Event Record Types

ADD_GROUP
ADD_USER
ANOM_ABEND
ANOM_ACCESS_FS
ANOM_ADD_ACCT
ANOM_AMTU_FAIL
ANOM_CRYPTO_FAIL
ANOM_DEL_ACCT
ANOM_EXEC
ANOM_LOGIN_ACCT
ANOM_LOGIN_FAILURES
ANOM_LOGIN_LOCATION
ANOM_LOGIN_SESSIONS
ANOM_LOGIN_TIME
ANOM_MAX_DAC
ANOM_MAX_MAC
ANOM_MK_EXEC
ANOM_MOD_ACCT
ANOM_PROMISCUOUS
ANOM_RBAC_FAIL
ANOM_RBAC_INTEGRITY_FAIL
ANOM_ROOT_TRANS
AVC
AVC_PATH
BPRM_FCAPS
CAPSET
CHGRP_ID
CHUSER_ID
CONFIG_CHANGE
CRED_ACQ
CRED_DISP
CRED_REFR
CRYPTO_FAILURE_USER
CRYPTO_KEY_USER
CRYPTO_LOGIN
CRYPTO_LOGOUT
CRYPTO_PARAM_CHANGE_USER
CRYPTO_REPLAY_USER
CRYPTO_SESSION
CRYPTO_TEST_USER
CWD
DAC_CHECK
DAEMON_ABORT
DAEMON_ACCEPT
DAEMON_CLOSE
DAEMON_CONFIG
DAEMON_END
DAEMON_RESUME
DAEMON_ROTATE
DAEMON_START
DEL_GROUP
DEL_USER
EOE
EXECVE
FD_PAIR
FS_RELABEL
GRP_AUTH
INTEGRITY_DATA
INTEGRITY_HASH
INTEGRITY_METADATA
INTEGRITY_PCR
INTEGRITY_RULE
INTEGRITY_STATUS
IPC
IPC_SET_PERM
KERNEL
KERNEL_OTHER
LABEL_LEVEL_CHANGE
LABEL_OVERRIDE
LOGIN
MAC_CIPSOV4_ADD
MAC_CIPSOV4_DEL
MAC_CONFIG_CHANGE
MAC_IPSEC_ADDSA
MAC_IPSEC_ADDSPD
MAC_IPSEC_DELSA
MAC_IPSEC_DELSPD
MAC_IPSEC_EVENT
MAC_MAP_ADD
MAC_MAP_DEL
MAC_POLICY_LOAD
MAC_STATUS
MAC_UNBBL_STCADD
MAC_UNBBL_STCDEL
MMAP
MQ_GETSETATTR
MQ_NOTIFY
MQ_OPEN
MQ_SENDRECV
NETFILTER_CFG
NETFILTER_PKT
OBJ_PID
PATH
RESP_ACCT_LOCK
RESP_ACCT_LOCK_TIMED
RESP_ACCT_REMOTE
RESP_ACCT_UNLOCK_TIMED
RESP_ALERT
RESP_ANOMALY
RESP_EXEC
RESP_HALT
RESP_KILL_PROC
RESP_SEBOOL
RESP_SINGLE
RESP_TERM_ACCESS
RESP_TERM_LOCK
ROLE_ASSIGN
ROLE_REMOVE
SELINUX_ERR
SERVICE_START
SERVICE_STOP
SOCKADDR
SYSTEM_BOOT
SYSTEM_RUNLEVEL
SYSTEM_SHUTDOWN
TEST
TRUSTED_APP
TTY
USER
USER_ACCT
USER_AUTH
USER_AVC
USER_CHAUTHTO
USER_CMD
USER_END
USER_ERR
USER_LABELED_EXPORT
USER_LOGIN
USER_LOGOUT
USER_MAC_POLICY_LOAD
USER_MGMT
USER_ROLE_CHANGE
USER_SELINUX_ERR
USER_START
USER_TTY
USER_UNLABELED_EXPORT
USYS_CONFIG
Ausearch

- The ausearch program is the preferred way to look at audit logs
- Can do simple queries
- Correlates the individual records to 1 event
- Can interpret some fields from numeric data to human readable form
- Can be used to extract events from audit logs
Ausearch Examples

- Searching for bad logins:
  - `ausearch -m USER_AUTH,USER_ACCT --success no`
- Searching for events on shadow file today
  - `ausearch --start today -f shadow`
- Searching for failed file opens for user acct 500
  - `ausearch -m PATH --success no --syscall open --loginuid 500`
- Extracting logs for 2 days
  - `ausearch --start yesterday --raw > new.log`
- Output can be piped to other audit utilities but requires --raw output
Aureport

- Utility that provides columnar reports on audit data
- Intended to be used for scripting more interesting reports from raw data
- Gives a summary report about what's been happening on your machine
- Each item in summary report leads to a report on that topic where summary or columnar data is given.
- Can read from stdin so that ausearch can pipe data to it
Aureport Output

Summary Report

Selected time for report: 10/09/2011 00:00:00 - 10/14/2011 11:13:01.139
Number of changes in configuration: 360
Number of changes to accounts, groups, or roles: 2
Number of logins: 9
Number of failed logins: 0
Number of authentications: 14
Number of failed authentications: 0
Number of users: 3
Number of terminals: 11
Number of host names: 1
Number of executables: 21
Number of files: 22
Number of AVC's: 8
Number of MAC events: 17
Number of failed syscalls: 111
Number of anomaly events: 0
Number of responses to anomaly events: 0
Number of crypto events: 0
Number of keys: 9
Number of process IDs: 135
Number of events: 660
Session Reporting

- Aulast is designed to give reports on login sessions
- Designed to look and act like the 'last' command
- Based on audit logs rather than utmp
- Proof mode
  - Output what events it used to bound the session
  - Provide the ausearch command to extract the session for further analysis
reboot system boot 2.6.35.14-97.fc1 Fri Oct 14 07:12 - 07:53 (00:40)
sgrubb tty1 ? Fri Oct 14 10:09 - 10:27 (00:17)
reboot system boot 2.6.35.14-97.fc1 Fri Oct 14 10:08 - 10:27 (00:18)
sgrubb tty1 ? Fri Oct 14 12:40 - down
reboot system boot 2.6.35.14-97.fc1 Fri Oct 14 18:06 - 18:35 (00:29)
sgrubb tty1 ? Fri Oct 14 18:08 - 18:35 (00:26)
reboot system boot 2.6.35.14-97.fc1 Sat Oct 15 08:31
sgrubb tty1 ? Sat Oct 15 08:32 still logged in

reboot system boot 2.6.35.14-97.fc1 Fri Oct 14 18:06 - 18:35 (00:29)
audit event proof serial numbers: 5, 0, 173
Session data can be found with this search:
ausearch --start 10/14/2011 18:06:01 --end 10/14/2011 18:35:08
sgrubb tty1 ? Fri Oct 14 18:08 - 18:35 (00:26)
audit event proof serial numbers: 61, 64, 174
Session data can be found with this search:
ausearch --start 10/14/2011 18:08:56 --end 10/14/2011 18:35:08 --session 1
Investigation Tips

- Main idea is to use 'keys' to group events
- Use key summary report of aureport
- Use ausearch --key to grab events with same key
  - Feed those into aureport for summary reports like file, executable, user, host
  - Audit.rules man page has examples
- More examples can be found in issue #5 of http://magazine.hitb.org/hitb-magazine.html
Audit Parsing Library

- Design goals
  - Completely hide the log file format in case it changes over time
  - Abstract all internal data structures to make friendly to other languages
  - Create iterator approach like database libraries
  - Search API so that only records of interest can be found
  - Ability to translate from numeric values to human readable
Auparse Overview

Log
+ events : Event
+ ausearch_setup()
+ ausearch_next_event()
+ next_event()

Event
+ timestamp :
+ serial_number :
+ node_name :
+ records : Record
+ node_compare()
+ timestamp_compare()
+ get_num_records()
+ first_record()
+ next_record()
+ goto_record_num()

Record
+ Type :
+ line_number :
+ fields : Field
+ record_text :
+ first_field()
+ next_field()
+ get_num_fields()
+ find_field()
+ find_field_next()

Field
+ Name :
+ Value :
+ Type :
+ interpret_field()
+ get_field_int()
Audit Parsing Library Example - C

auparse_state_t *au = auparse_init(AUSOURCE_FILE, ".\test.log");
do {
    do {
        do {
            printf("%s=%s (%s)\n", auparse_get_field_name(au),
                    auparse_get_field_str(au), auparse_interpret_field(au));
        } while (auparse_next_field(au) > 0);
    } while (auparse_next_record(au) > 0);
} while (auparse_next_event(au) > 0);
Audit Parsing Library Example - Python

```python
au = auparse.AuParser(auparse.AUSOURCE_FILE, ".//test.log");
while True:
    while True:
        while True:
            print "%s=%s (%s)" % (au.get_field_name(), au.get_field_str(), au.interpret_field())
            if not au.next_field(): break
            if not au.next_record(): break
            if not au.parse_next_event(): break
```
Auparse Feed API
Requirements for IDS/IPS

- The tools shall build upon audit reduction and analysis tools to aid the ISSO or ISSM in the monitoring and detection of suspicious, intrusive, or attack-like behavior patterns.
- The capability of the system to monitor occurrences of, or accumulation of, auditable events that may indicate an imminent violation of security policies.
- The capability of the system to notify the ISSO of suspicious events and taking the least-disruptive action to terminate the suspicious events.
- In real time
Audit Event Data Flow

Kernel

Audit Daemon

Event Dispatcher

Setroubleshoot

Remote Logging

IDS/IPS
Audit Event Dispatcher Plugins

- Programming rules
  - Must read from stdin
  - Must obey signals such as SIGHUP, SIGTERM
  - Must read config information from file
- Types of plugins
  - Output (passes event to something else)
    - Remote logging, af_unix, setroubleshooter
  - Translational (changes event content/format)
    - Event filter, protocol converter, IDMEF
Audit Event Feeds

- Kernel
- Trusted Programs
  - Pam
  - Login, sshd, gdm, sudo, crond
  - Shadow-utils, passwd
  - Semanage, init, libvirt, dbus, nscd, cups
- MAC selinux-policy
- Test Apps
  - Amtu
  - Aide
- (Security Scanning Tool)
IDS/IPS System

Kernel

Audit

Audispd

Detect

React
Attacks – Anomaly Record Types

- Gain Entry to system
  - Login / exploit
    - AUDIT_ANOM_LOGIN_FAILURES - Failed login limit reached
    - AUDIT_ANOM_LOGIN_TIME - Login attempted at bad time
    - AUDIT_ANOM_LOGIN_SESSIONS - Max concurrent sessions reached
    - AUDIT_ANOM_LOGIN_ACCT - Login attempted to watched acct
    - AUDIT_ANOM_LOGIN_LOCATION - Login from forbidden location
    - AUDIT_ANOM_ABEND - Process ended abnormally
    - AUDIT_ANOM_MAX_MAC - Max MAC failures reached
Attacks – Anomaly Record Types

- Access files or resources
  - AUDIT_ANOM_MAX_DAC - Max DAC failures reached
  - AUDIT_ANOM_MAX_MAC - Max MAC failures reached
  - AUDIT_ANOM_ACCESS_FS - Access of file or dir
  - AUDIT_ANOM_EXEC - Execution of program

- Become root
  - AUDIT_ANOM_ROOT_TRANS – Unexpected transition to uid 0

- Change trusted database
  - AUDIT_ANOM_ACCESS_FS - Access of file or dir
  - AUDIT_ANOM_AMTU_FAIL - AMTU failure
Attacks – Anomaly Record Types

- Add or modify account and passwords
  - AUDIT_ANOM_ADD_ACCT - Adding an acct
  - AUDIT_ANOM_DEL_ACCT - Deleting an acct
  - AUDIT_ANOM_MOD_ACCT - Changing an acct
- Install programs
  - AUDIT_ANOM_MK_EXEC - Make an executable
  - Integrity events probably need a mapping to AUDIT_ANOM_
- Start / stop services
  - AUDIT_ANOM_EXEC - Execution of file
- Watch other users
  - AUDIT_ANOM_ACCESS_FS - Access of file or dir
  - AUDIT_ANOM_MK_EXEC - Make an executable
Attacks – Anomaly Record Types

- Kill audit system
  - AUDIT_ANOM_RBAC_FAIL - RBAC self test failure
  - Plugin would also see an audit daemon stop event and the user sending it
- Sniff traffic
  - AUDIT_ANOM_PROMISCUOUS - Device changed promiscuous mode
- Gain entry to other systems
  - We would have to correlate logging from all machines
Attack Reaction Types

- AUDIT_RESP_ANOMALY - Anomaly not reacted to
- AUDIT_RESP_ALERT - Alert email was sent
- AUDIT_RESP_KILL_PROC - Kill program
- AUDIT_RESP_TERM_ACCESS - Terminate session
- AUDIT_RESP_ACCT_REMOTE - Acct locked from remote access
- AUDIT_RESP_ACCT_LOCK_TIMED - User acct locked for time
- AUDIT_RESP_ACCT_UNLOCK_TIMED - User acct unlocked from time
- AUDIT_RESP_ACCT_LOCK - User acct was locked
- AUDIT_RESP_TERM_LOCK - Terminal was locked
- AUDIT_RESP_SEBOOL - Set an SE Linux Boolean
- AUDIT_RESP_EXEC - Execute a script
- AUDIT_RESP_SINGLE - Go to single user mode
- AUDIT_RESP_HALT - take the system down
Questions?

Email: sgrubb@redhat.com
Web Page: http://people.redhat.com/sgrubb/audit