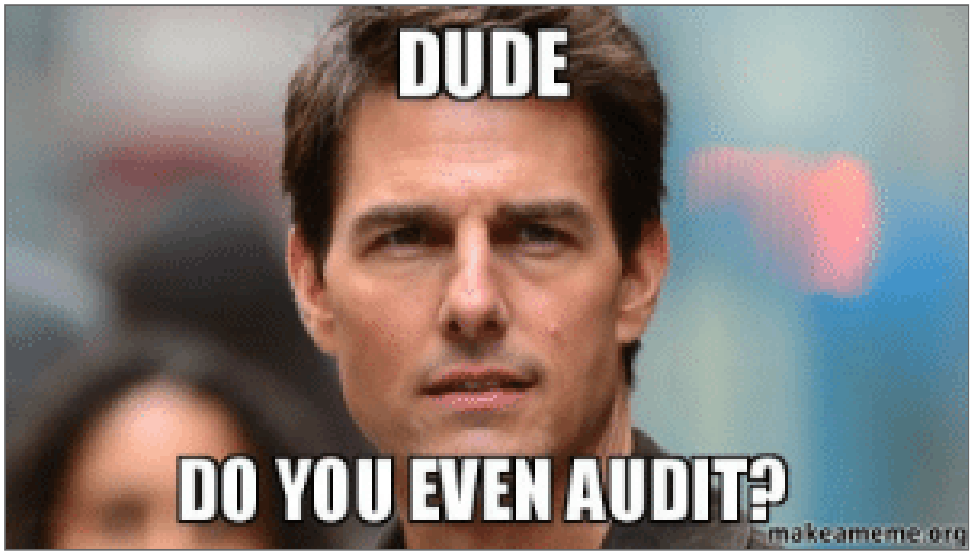
DUDE, DO YOU EVEN AUDIT? 🧐



지금 이 순간, Linux Auditing System 슈퍼 유저 & 내부 구조 잘알!

Paran Lee <p4ranlee@gmail.com>

INDEX

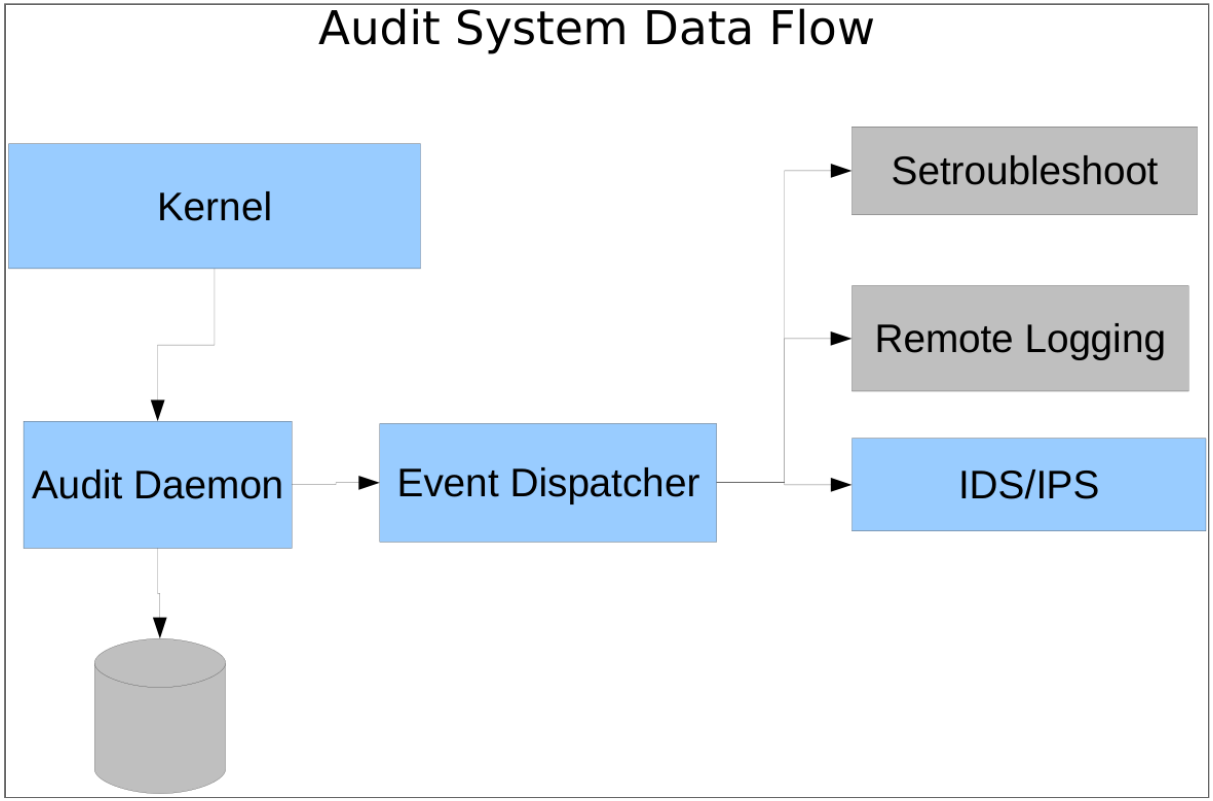
1. AUDIT 을 어떻게 활용할까? 🤔

- 🤔 어떻게 증거(로그)를 남기면 좋을까요?
  - 스토리지 서버 /opt 디렉토리 안에 중요한 파일을 누가 지우셨어욥*!?*
  - 컨테이너 안에 이상한 소프트웨어가 자꾸 설치되요. 누가 설치한거죠*..?*
  - 사내 소스 서버에 자꾸 이상한 *IP* 가 접근합니다. 이러다가 전부 *DRM* 걸리거나 소스 콸콸콸 유출 아니겠죠*?!*

고민하지 말고, **AUDIT** 을 사용해봅시다**!**

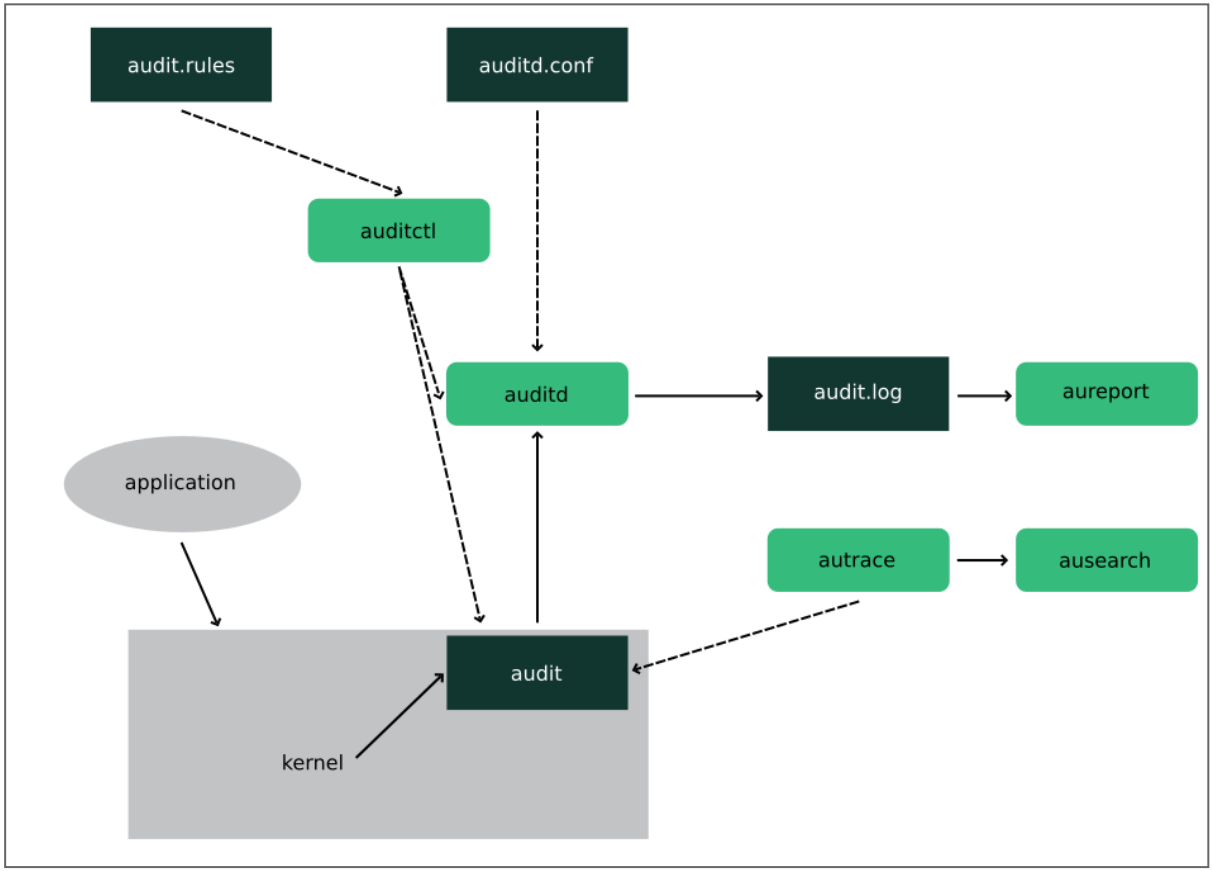
1. AUDIT 을 어떻게 활용할까? 🤔

- **audit** 을 잘 사용하기 위한 사용법, 그리고 커널/유저 동작과 소프트웨어 컴포넌트는 어떻게 구성되어 있는지 살펴봅니다.



1.1 큰 그림

- 유저 프로세스가 하는 행동의 로그를 남기기 위한 커널 프로세스 kaudit 와 유저 libaudit 프레임워크를 활용한 소프트웨어 인프라입니다.



- 커널 스페이스의 audit lsm hook 로 ctx -> audit_context -> skb
- netlink 로 auditd 와 통신하며 행동 로깅 및 새로운 룰을 적용합니다.

1.1 주요 특징

- 커널 프로세스로 상주 중인 kaudit 은 **security/lsm_audit.c** Hook 을 사용하여 ctx 정보를 가져옵니다.
- auditd 가 올라오면서 **audit.rules** 파일을 읽어 정책을 적용합니다.
- auditctl 로 운영 중인 시스템에 적용합니다.
- 아키텍쳐 지원 : arm, x86, s390 (32, 64 bit)
  - **aarch64_table.h**
- 리눅스 커널 시스템 콜 테이블의 새로운 시스템 콜을 팔로우 업 합니다.
  - (예) _S(280, "bpf") audit 3.0 BPF 시스템 콜 감사 정책 지원
  - Add **bpf syscall** command argument interpretation to auparse
  - 호스트 머신에서 virtual machine, container 감시 관련 feature 를 확장 중입니다.
- audit 은 빨간 모자의 안보가 중요한 서버 솔루션에서 침입 탐지 시스템 **(Intrusion Detection System)**으로 활용 중입니다.

1.2 MAN

- auditd 는 유저 공간 Linux Auditing System 입니다.
  - audit records를 디스크에 쓰는 임무를 맡은 친구죠.
- 로그는 ausearch 또는 aureport 를 통하여 편리하게 볼 수 있습니다.
- auditctl 을 통해 운영 중에 audit 설정을 바꾸거나 룰을 변경할 수 있어요!
- augenrules 은 /etc/audit/rules.d/ 안에 있는 룰 파일들을 */etc/audit/audit.rules* 파일로 만들어 줍니다.
- auditd.conf 설정을 바꾸어서 auditd 를 입맛대로 설정할 수 있어요!

1.3 AUDIT 슈퍼 유저가 되기 - 실행 확인하기

```
# service auditd status
Redirecting to /bin/systemctl status auditd.service
● auditd.service - Security Auditing Service
   Loaded: loaded (/usr/lib/systemd/system/auditd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2022-09-28 01:42:53 KST; 54min ago
     Docs: man:auditd(8)
           https://github.com/linux-audit/audit-documentation
 Main PID: 1014 (auditd)
    Tasks: 4 (limit: 49134)
   Memory: 3.6M
   CGroup: /system.slice/auditd.service
           ├─1014 /sbin/auditd
           └─1016 /usr/sbin/sedispatch

 9월 28 01:42:53 localhost.localdomain augenrules[1033]: enabled 1
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: failure 1
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: pid 1014
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: rate_limit 0
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: backlog_limit 8192
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: lost 0
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: backlog 4
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: backlog_wait_time 60000
 9월 28 01:42:53 localhost.localdomain augenrules[1033]: backlog_wait_time_actual 0
```

- 리눅스 커널의 kauditd
- 유저 스페이스의 레드햇/데비안 계열 배포판 auditd 로 활성화 되어있습니다.

1.3 AUDIT 슈퍼 유저가 되기 - 실행 확인하기

```
# 레드햇 계열 # dnf install auditd
# 데비안 계열 # apt install auditd
```

1.3 AUDIT 슈퍼 유저가 되기 - AUREPORT

ssh 접근, 즉 sshd fork 하여 유저가 로그인하는 행위의 로그를 봐 볼까요!

```
#  aureport -l --failed

Login Report
============================================
# date time auid host term exe success event
============================================
1. 2022년 09월 28일 05:33:54 (unknown) 192.168.66.1 ssh /usr/sbin/sshd no 205
2. 2022년 09월 28일 05:33:54 (unknown) 192.168.66.1 ssh /usr/sbin/sshd no 216
3. 2022년 09월 28일 05:34:06 (unknown) 192.168.66.1 ssh /usr/sbin/sshd no 227
4. 2022년 09월 28일 05:35:44 ahnlab 192.168.66.1 ssh /usr/sbin/sshd no 268
5. 2022년 09월 28일 05:35:44 ahnlab 192.168.66.1 ssh /usr/sbin/sshd no 281
```

```
#  aureport -l --success

Login Report
============================================
# date time auid host term exe success event
============================================
1. 2022년 09월 28일 04:47:43 1000 ::1 /dev/pts/1 /usr/sbin/sshd yes 208
2. 2022년 09월 28일 05:35:38 1000 192.168.66.1 /dev/pts/1 /usr/sbin/sshd yes 245
3. 2022년 09월 28일 05:35:46 1000 192.168.66.1 ssh /usr/sbin/sshd yes 299
```

1.3 AUDIT 슈퍼 유저가 되기 - FILE RULE

- 외부에서 /etc/ssh/sshd_config 파일을 읽거나 수정하려는 모든 시도를 남겨볼까요?
- 해당 rule 을 sshd_config 키로 기록해보죠!

```
$ auditctl -w /etc/ssh/sshd_config -p warx -k sshd_config
```

```
$ # ausearch -k sshd_config
----
time->Wed Sep 28 06:04:31 2022
type=SYSCALL msg=audit(1664312671.115:387): arch=c000003e syscall=44 success=yes exit=1088 a0=4 a1=7ffc4fcb9
type=CONFIG_CHANGE msg=audit(1664312671.115:387): auid=1000 ses=3 subj=unconfined_u:unconfined_r:unconfined_
----
time->Wed Sep 28 06:04:46 2022
type=PATH msg=audit(1664312686.595:388): item=0 name="/etc/ssh/sshd_config" inode=103004708 dev=fd:00 mode=0
type=SYSCALL msg=audit(1664312686.595:388): arch=c000003e syscall=257 success=yes exit=3 a0=fffffff9c a1=55db
```

1.3 AUDIT 슈퍼 유저가 되기 - RECORD TYPE

- type=SYSCALL
  - type 필드에는 레코드 유형이 포함됩니다. 이 예제에서 SYSCALL 값은 커널에 대한 시스템 호출에 의해 이 레코드가 트리거되었음을 지정합니다.
- key=“sshd_config”
  - 키 필드는 감사 로그에서 이 이벤트를 생성한 규칙과 관련된 관리자 정의 문자열을 기록합니다.

```
type=SYSCALL msg=audit(1364481363.243:24287): arch=c000003e
syscall=2 success=no exit=-13
a0=7fffd19c5592 a1=0 a2=7fffd19c4b50 a3=a items=1
ppid=2686 pid=3538 auid=1000
uid=1000 gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000
tty=pts0 ses=1 comm="cat" exe="/bin/cat"
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
key="sshd_config"
```

1.3 AUDIT 슈퍼 유저가 되기 - RECORD TYPE

- ppid=2686
  - ppid 필드는 상위 프로세스 ID(PPID)를 기록합니다. 이 경우 2686 은 bash 와 같은 상위 프로세스의 PPID였습니다.
- pid=3538
  - pid 필드는 프로세스 ID(PID)를 기록합니다. 이 경우 3538 은 cat 프로세스의 PID입니다.
- auid=1000
  - auid 필드는 loginuid인 Audit 사용자 ID를 기록합니다. 이 ID는 로그인 시 사용자에게 할당되며, 예를 들어 su - john 명령으로 사용자 계정을 전환하여 사용자의 ID가 변경될 경우에도 모든 프로세스에 상속됩니다.

```
type=SYSCALL msg=audit(1364481363.243:24287): arch=c000003e
syscall=2 success=no exit=-13 a0=7fffd19c5592 a1=0 a2=7fffd19c4b50 a3=a items=1
ppid=2686 pid=3538 auid=1000 uid=1000 gid=1000 euid=1000
suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000
tty=pts0 ses=1 comm="cat" exe="/bin/cat"
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key="sshd_config"
```

1.3 AUDIT 슈퍼 유저가 되기 - NETWORK RULE

- 유입, 유출되는 네트워크 연결을 확인할 수 있습니다.
  - 172.20.14.41:60822 에서 현재 localhost 로 접근을 확인 가능합니다!

```
$ auditctl -a always,exit -F arch=b64 -S accept,connect -F key=external-access
```

```
type=PROCTITLE msg=audit(2022년 08월 02일 09:47:19.345:372385) :
proctitle=/usr/sbin/lighttpd -f /etc/lighttpd/lighttpd.conf
type=SOCKADDR msg=audit(2022년 08월 02일 09:47:19.345:372385) :
saddr={ fam=inet laddr=172.20.14.41 lport=60822 }
type=SYSCALL msg=audit(2022년 08월 02일 09:47:19.345:372385) :
arch=x86_64 syscall=accept success=yes exit=11
a0=0x4 a1=0x7fffffffe870 a2=0x7fffffffe844 a3=0x3e8
items=0 ppid=1 pid=22267 auid=unset uid=lighttpd gid=lighttpd
euid=lighttpd suid=lighttpd fsuid=lighttpd egid=lighttpd sgid=lighttpd
fsgid=lighttpd tty=(none) ses=unset comm=lighttpd
exe=/usr/sbin/lighttpd key=my_accept
```

1.3 AUDIT 슈퍼 유저가 되기 - PROCESS RULE

- 일반 사용자의 root로 권한 상승 시도를 확인해볼까요?

```
$ auditctl -a always,exit -F arch=b64 -S execve -C uid!=euid -F euid=0 -F key=10.2.5.b-elevated-privs-setui
```

```
type=SYSCALL msg=audit(1659428449.377:378871): arch=c000003e
syscall=59 success=yes exit=0 a0=6f4f60 a1=6fa4b0 a2=822900 a3=7fffffffe260
items=2 ppid=14078 pid=15495 auid=1052 uid=1052 gid=1052
euid=0 suid=0 fsuid=0 egid=1052 sgid=1052 fsgid=1052
tty=pts8 ses=26782 comm="su" exe="/usr/bin/su"
key="10.2.5.b-elevated-privs-setuid"
---
type=USER_AUTH msg=audit(1659428456.960:378872): pid=15495 uid=1052 auid=1052 ses=26782
msg='op=PAM:authentication grantors=pam_faillock,pam_unix acct="root" exe="/usr/bin/su" hostname=localhost.l
```

- 이외에도 시스템 룰 예시를 조금 더 살펴볼려면? **30-stig.rules** 참고!
  - Security Technical Implementation (STIG, 미국 국방성의 DISA 보안 구성 표준)에서 요구하는 조건을 충족할 수 있도록 구성된 Audit 규칙입니다.

1.3 AUDIT 슈퍼 유저가 되기 - CONFIG

기본 설정은 아래와 같이 확인 할 수 있습니다.

- event buffer 사이즈 8192
- burst of events 시에 60000 만큼 기다린다.

```
# cat /etc/audit/rules.d/audit.rules
## First rule - delete all
-D

## Increase the buffers to survive stress events.
## Make this bigger for busy systems
-b 8192

## This determine how long to wait in burst of events
--backlog_wait_time 60000

## Set failure mode to syslog
-f 1
```

```
# cat /etc/audit/audit.rules
## This file is automatically generated from /etc/audit/rules.d
-D
-b 8192
-f 1
--backlog_wait_time 60000
```

2. 리눅스 커널 **KAUDITD** 내부 구조 분석!

- kaudit 의 중요한 부분을 콕 찍먹해볼 시간입니다.

2.1. KADUIT 언제 어떻게 초기화 될까요?

```
-> arch_call_rest_init()
  -> rest_init()
    -> pid = kernel_thread(kernel_init, NULL, CLONE_FS);
      -> kernel_init()
        -> kernel_init_freeable()
          -> do_basic_setup()
            -> do_initcalls()
```

- do_initcalls() 내의 Linux 커널 부팅 중 초기화 호출 상대적 순서를 살펴보면, 3번째에 해당하는 것을 볼 수 있습니다.
  - early_initcall(), core_initcall()
  - **postcore_initcall() → postcore_initcall(audit_init);**
  - arch_initcall(), subsys_initcall(), fs_initcall(), device_initcall()

2.1. KADUIT 언제 어떻게 초기화 될까요?

```
# dmesg | grep audit
[    0.215458] audit: initializing netlink subsys (disabled)
[    0.215500] audit: type=2000 audit(1664301355.215:1): state=initialized audit_enabled=0 res=1
[    7.430702] audit: type=1404 audit(1664301363.005:2): enforcing=1 old_enforcing=0 auid=4294967295 ses=429
[    7.786790] audit: type=1403 audit(1664301363.360:3): auid=4294967295 ses=4294967295 lsm=selinux res=1
```

```
$ ps -ef | grep audit
root          69       2  0 09:29 ?        00:00:00 [kauditd]
root        1038       1  0 09:29 ?        00:00:00 /sbin/auditd
```

2.1. KADUIT 언제 어떻게 초기화 될까요?

- **audit_init()** 다음 항목에 주목해서 보면 어떨까요?
  - skb_queue 자료구조 audit_queue
  - kauditd_thread(), audit_log()

```
/* Initialize audit support at boot time. */
static int __init audit_init(void)
{
        int i;

        if (audit_initialized == AUDIT_DISABLED)
                return 0;

        audit_buffer_cache = kmem_cache_create("audit_buffer",
                                               sizeof(struct audit_buffer),
                                               0, SLAB_PANIC, NULL);


        skb_queue_head_init(&audit_queue);
        skb_queue_head_init(&audit_retry_queue);
        skb_queue_head_init(&audit_hold_queue);

        for (i = 0; i < AUDIT_INODE_BUCKETS; i++)
                INIT_LIST_HEAD(&audit_inode_hash[i]);
        // ...
        audit_initialized = AUDIT_INITIALIZED;

        kauditd_task = kthread_run(kauditd_thread, NULL, "kauditd");
        // ...
```

- 큐 자료구조 초기화 및 kauditd 커널 스레드의 생성을 확인할 수 있습니다.

2.2. AUDIT.LOG 로그에 찍히기까지

- syscall_trace_enter -> audit_syscall_entry 공통 훅
  - **arch/arm64/kernel/ptrace.c**

```
int syscall_trace_enter(struct pt_regs *regs)
{
        unsigned long flags = read_thread_flags();

        if (flags & (_TIF_SYSCALL_EMU | _TIF_SYSCALL_TRACE)) {
                report_syscall(regs, PTRACE_SYSCALL_ENTER);
                if (flags & _TIF_SYSCALL_EMU)
                        return NO_SYSCALL;
        }

        /* Do the secure computing after ptrace; failures should be fast. */
        if (secure_computing() == -1)
                return NO_SYSCALL;

        if (test_thread_flag(TIF_SYSCALL_TRACEPOINT))
                trace_sys_enter(regs, regs->syscallno);

        audit_syscall_entry(regs->syscallno, regs->orig_x0, regs->regs[1],
                            regs->regs[2], regs->regs[3]);

        return regs->syscallno;
}
```

2.2. AUDIT.LOG 로그에 찍히기까지

- syscall_trace_exit -> audit_syscall_exit 공통 훅
  - **arch/arm64/kernel/ptrace.c**

```
void syscall_trace_exit(struct pt_regs *regs)
{
        unsigned long flags = read_thread_flags();

        audit_syscall_exit(regs);

        if (flags & _TIF_SYSCALL_TRACEPOINT)
                trace_sys_exit(regs, syscall_get_return_value(current, regs));

        if (flags & (_TIF_SYSCALL_TRACE | _TIF_SINGLESTEP))
                report_syscall(regs, PTRACE_SYSCALL_EXIT);

        rseq_syscall(regs);
}
```

2.2. AUDIT.LOG 로그에 찍히기까지

- **struct common_audit_data** : 공통 lsm audit log 을 위한 data 구조체

```
/* Auxiliary data to use in generating the audit record. */
struct common_audit_data {
        union   {
                struct path path;
                struct dentry *dentry;
                struct inode *inode;
                struct lsm_network_audit *net;
                int cap;
                int ipc_id;
                struct task_struct *tsk;
                char *kmod_name;
                struct lsm_ioctlop_audit *op;
                struct file *file;
                struct lsm_ibpkey_audit *ibpkey;
                struct lsm_ibendport_audit *ibendport;
                int reason;
                const char *anonclass;
        } u;
}
```

2.2. AUDIT.LOG 로그에 찍히기까지

- **common_lsm_audit()** : Hook 에서 audit 하기 위해 사용할 함수

```
/**
 * common_lsm_audit - generic LSM auditing function
 * @a:  auxiliary audit data
 * @pre_audit: lsm-specific pre-audit callback
 * @post_audit: lsm-specific post-audit callback
 *
 * setup the audit buffer for common security information
 * uses callback to print LSM specific information
 */
void common_lsm_audit(struct common_audit_data *a,
        void (*pre_audit)(struct audit_buffer *, void *),
        void (*post_audit)(struct audit_buffer *, void *))
{
        struct audit_buffer *ab;

        if (a == NULL)
                return;
        /* we use GFP_ATOMIC so we won't sleep */
        ab = audit_log_start(audit_context(), GFP_ATOMIC | __GFP_NOWARN,
                             AUDIT_AVC);

        if (ab == NULL)
                return;
```

2.2. AUDIT.LOG 로그에 찍히기까지

- **audit_log_start()** : struct audit_buffer 인스턴스를 만들어줍니다.
- **audit_log_end()** : 만든 버퍼를 큐잉합니다.

```
/* The audit_buffer is used when formatting an audit record.  The caller
 * locks briefly to get the record off the freelist or to allocate the
 * buffer, and locks briefly to send the buffer to the netlink layer or
 * to place it on a transmit queue.  Multiple audit_buffers can be in
 * use simultaneously. */
struct audit_buffer {
        struct sk_buff       *skb;      /* formatted skb ready to send */
        struct audit_context *ctx;      /* NULL or associated context */
        gfp_t                gfp_mask;
};
```

```
/**
 * audit_log - Log an audit record
 * @ctx: audit context
 * @gfp_mask: type of allocation
 * @type: audit message type
 * @fmt: format string to use
 * @...: variable parameters matching the format string
 *
 * This is a convenience function that calls audit_log_start,
 * audit_log_vformat, and audit_log_end.  It may be called
 * in any context.
 */
void audit_log(struct audit_context *ctx, gfp_t gfp_mask, int type,
               const char *fmt, ...)
{
        struct audit_buffer *ab;
        va_list args;
```

```
va_start(args, fmt);
audit_log_vformat(ab, fmt, args);
va_end(args);
```

- 로그를 만들기위한 버퍼를 사용하는 루틴을 확인합니다.

2.2. AUDIT.LOG 로그에 찍히기까지

- **struct audit_context** 를 가지고 **struct audit_buffer** 인스턴스를 만들어줍니다.

```
/**
 * audit_log_start - obtain an audit buffer
 * @ctx: audit_context (may be NULL)
 * @gfp_mask: type of allocation
 * @type: audit message type
 *
 * Returns audit_buffer pointer on success or NULL on error.
 *
 * Obtain an audit buffer.  This routine does locking to obtain the
 * audit buffer, but then no locking is required for calls to
 * audit_log_*format.  If the task (ctx) is a task that is currently in a
 * syscall, then the syscall is marked as auditable and an audit record
 * will be written at syscall exit.  If there is no associated task, then
 * task context (ctx) should be NULL.
 */
struct audit_buffer *audit_log_start(struct audit_context *ctx, gfp_t gfp_mask,
                                     int type)
{
        struct audit_buffer *ab;
        struct timespec64 t;
        unsigned int serial;

        if (audit_initialized != AUDIT_INITIALIZED)
```

- **Hook** 의 **ctx -> audit_context -> audit_buffer**

2.2. AUDIT.LOG 로그에 찍히기까지

- **struct audit_buffer** 를 가지고 넷링크 통신을 위한 sk_buff 를 만들고, audit_queue 테일에 큐잉합니다.

```
/**
 * audit_log_end - end one audit record
 * @ab: the audit_buffer
 *
 * We can not do a netlink send inside an irq context because it blocks (last
 * arg, flags, is not set to MSG_DONTWAIT), so the audit buffer is placed on a
 * queue and a kthread is scheduled to remove them from the queue outside the
 * irq context.  May be called in any context.
 */
void audit_log_end(struct audit_buffer *ab)
{
        struct sk_buff *skb;
        struct nlmsghdr *nlh;

        if (!ab)
                return;

        if (audit_rate_check()) {
                skb = ab->skb;
                ab->skb = NULL;

                /* setup the netlink header, see the comments in
                 * kauditd_send_multicast_skb() for length quirks */
```

- queue the netlink packet and poke the kauditd thread

2.2. AUDIT.LOG 로그에 찍히기까지

- kthread 에서는 irq context 밖에서 audit_queue 처리가 가능합니다!
  - audit buffer 는 audit_queue 에 들어가구요.
  - 프로세스 컨텍스트에서 처리해서 핸들링이 쉬워지죠!

```
/**
 * kauditd_thread - Worker thread to send audit records to userspace
 * @dummy: unused
 */
static int kauditd_thread(void *dummy)
{
        int rc;
        u32 portid = 0;
        struct net *net = NULL;
        struct sock *sk = NULL;
        struct auditd_connection *ac;

#define UNICAST_RETRIES 5

        set_freezable();
        while (!kthread_should_stop()) {
                /* NOTE: see the lock comments in auditd_send_unicast_skb() */
                rcu_read_lock();
                ac = rcu_dereference(auditd_conn);
                if (!ac) {
                        rcu_read_unlock();
                        goto main_queue;
                }
```

- netlink 기반 logger 구현 한다면 참고할 수 있는 좋은 코드네요!

2.2. AUDIT.LOG 로그에 찍히기까지

- **kauditd_thread()** 내부에서 보았던, 유저 스페이스로 netlink 소켓으로 패킷을 전달하는 부분입니다!

```
/**
 * kauditd_send_queue - Helper for kauditd_thread to flush skb queues
 * @sk: the sending sock
 * @portid: the netlink destination
 * @queue: the skb queue to process
 * @retry_limit: limit on number of netlink unicast failures
 * @skb_hook: per-skb hook for additional processing
 * @err_hook: hook called if the skb fails the netlink unicast send
 *
 * Description:
 * Run through the given queue and attempt to send the audit records to auditd,
 * returns zero on success, negative values on failure.  It is up to the caller
 * to ensure that the @sk is valid for the duration of this function.
 *
 */
static int kauditd_send_queue(struct sock *sk, u32 portid,
                              struct sk_buff_head *queue,
                              unsigned int retry_limit,
                              void (*skb_hook)(struct sk_buff *skb),
                              void (*err_hook)(struct sk_buff *skb, int error))
{
        int rc = 0;
        struct sk_buff *skb = NULL;
```

- 큐잉한 audit_queue 에서 skb 를 get 하고, 이를 netlink 로 전달합니다.

2.2. AUDIT.LOG 로그에 찍히기까지

- **struct audit_context** 멤버를 슬쩍 볼까요! 자세한 설명은 생략!

```
/* The per-task audit context. */
struct audit_context {
        int                 dummy;      /* must be the first element */
        enum {
                AUDIT_CTX_UNUSED,       /* audit_context is currently unused */
                AUDIT_CTX_SYSCALL,      /* in use by syscall */
                AUDIT_CTX_URING,        /* in use by io_uring */
        } context;
        enum audit_state    state, current_state;
        unsigned int        serial;     /* serial number for record */
        int                 major;      /* syscall number */
        int                 uring_op;   /* uring operation */
        struct timespec64   ctime;      /* time of syscall entry */
        unsigned long       argv[4];    /* syscall arguments */
        long                return_code;/* syscall return code */
        u64                 prio;
        int                 return_valid; /* return code is valid */
        /*
         * The names_list is the list of all audit_names collected during this
         * syscall.  The first AUDIT_NAMES entries in the names_list will
         * actually be from the preallocated_names array for performance
         * reasons.  Except during allocation they should never be referenced
         * through the preallocated_names array and should only be found/used
```

- The per-task audit context.

```
static inline void audit_set_context(struct task_struct *task, struct audit_context *ctx)
{
        task->audit_context = ctx;
}
```

```
        return current->audit_context;
}
```

– task_struct 에 audit_context 멤버가 있습니다.

2.3. AUDIT RULE 을 어떻게 로드할까?

- 유저 스페이스 : auditd

```
int audit_add_rule_data(int fd, struct audit_rule_data *rule,
                        int flags, int action)
{
        int rc;

        rule->flags  = flags;
        rule->action = action;
        rc = audit_send(fd, AUDIT_ADD_RULE, rule,
                        sizeof(struct audit_rule_data) + rule->buflen);
```

2.3. AUDIT RULE 을 어떻게 로드할까?

- 커널 스페이스 : kaduditd

```
/**
 * audit_receive - receive messages from a netlink control socket
 * @skb: the message buffer
 *
 * Parse the provided skb and deal with any messages that may be present,
 * malformed skbs are discarded.
 */
static void audit_receive(struct sk_buff  *skb)
{
        struct nlmsghdr *nlh;
        /*
         * len MUST be signed for nlmsg_next to be able to dec it below 0
         * if the nlmsg_len was not aligned
         */
        int len;
        int err;

        nlh = nlmsg_hdr(skb);
        len = skb->len;

        audit_ctl_lock();
        while (nlmsg_ok(nlh, len)) {
                err = audit_receive_msg(skb, nlh);
```

- 룰은 위의 로직을 통해서 유저에서 커널로 올라옴을 확인합니다!

2. 유저 스페이스 **AUDITD** 콤포넌트 분석**!**

- auditd 의 중요한 부분을 콕 찍먹해볼 시간입니다.

2.1. AUDITD 언제 어떻게 초기화 할까요?

- **auditd.c - main()** 콕 찍어볼까요?

```
        /* Load the Configuration File */
        if (load_config(&config, TEST_AUDITD))

        /* Init netlink */
        if ((fd = audit_open())

        // Init complete, start event loop
        if (!stop)
                ev_loop (loop, 0);
```

```
int audit_open(void)
{
        int fd = socket(PF_NETLINK, SOCK_RAW, NETLINK_AUDIT);
```

2.2. 감사 정책을 어떻게 로드할까요?

auditd 데몬을 직접 만들어볼까요? libaudit + auditd 내부에서도 사용하는 libev 로 ~

```c
#include <stdio.h>
#include <unistd.h>

#include <libaudit.h>

#include <ev.h>

static int fd;

void monitoring(struct ev_loop *loop, struct ev_io *io, int revents) {
    struct audit_reply reply;

    audit_get_reply(fd, &reply, GET_REPLY_NONBLOCKING, 0);

    if (reply.type != AUDIT_EOE &&
            reply.type != AUDIT_PROCTITLE &&
            reply.type != AUDIT_PATH) {
        printf("Event: Type=%s Message=%.*sWn",
                    audit_msg_type_to_name(reply.type),
                    reply.len,
                    reply.message);
    }
}
```

주요 릴리즈 변경

2020-12-17 audit 3.0 릴리즈부터는 기존 audispd 이벤트 디스패쳐 데몬을 auditd 로 통합했습니다.

This is the long awaited 3.0 major feature release. Most notable item is that audispd is gone.

- Merge auditd and audispd code
- Move all audispd config files under /etc/audit/
- Move audispd.conf settings into auditd.conf

TODO

```
Future roadmap (subject to change):
===================================
3.1
* Basic HIDS based on reactive audit component
* Multi-thread audisp-remote
* Add keywords for time: month-ago, this-hour, last-hour
* If searching user/group doesn't map to uid/gid, do translated string search
* In auditd, look into non-blocking handling of write to plugins
* Support multiple time streams when searching

3.2
* Container support
* Support TLS PSK as remote logging transport
* Add rule verify to detect mismatch between in-kernel and on-disk rules
* audisp-remote, add config to say what home network is so
  laptops don't try if their not on a network that can reach the server.
* Fix audit.pc.in to use Requires.private
* Change ausearch to output name="" unless its a real null.
  (mount) ausearch-report.c, 523. FIXME
* Fix SIGHUP for auditd network settings
* Add ability to filter events in auditd
```

이상입니다. ^^7 고생하셨습니다!

참고

- **Steve Grubb(Redhat) - Linux Audit**
- **Steve Grubb Blog - Security + Data Science**
- **Linux auditd for Threat Hunting [Part 1]**
- **Security hooks for Audit**
- **SLES 15-SP1 - Part VI The Linux Audit Framework**
- **Audit and IDS - Red Hat People**
- **Tracking User and System activity with Oracle Linux Auditing**
- **Stop disabling SELinux**

MISC(AUDIT PACKAGE)

```
# rpm -ql audit
/etc/audit
/etc/audit/audit-stop.rules
/etc/audit/audit.rules
/etc/audit/auditd.conf
/etc/audit/plugins.d
/etc/audit/plugins.d/af_unix.conf
/etc/audit/rules.d
/etc/audit/rules.d/audit.rules
/usr/bin/aulast
/usr/bin/aulastlog
/usr/bin/ausyscall
/usr/bin/auvirt
/usr/lib/.build-id
/usr/lib/.build-id/0a
/usr/lib/.build-id/0a/54a2aeda2ce2f0cc9c789ab94afde974ea3ddf
/usr/lib/.build-id/15
/usr/lib/.build-id/15/be5ccbbd0ba37fea823d161d849a3d48671c64
/usr/lib/.build-id/2d
/usr/lib/.build-id/2d/733d5160c5a1ef08df0c709fb7436df2e3a548
/usr/lib/.build-id/3f
/usr/lib/.build-id/3f/bf084e6e5e599ac11ef7055de93519681e0d78
/usr/lib/.build-id/4a
```

MISC(AUDIT PACKAGE)

```
# dnf search audit
마지막 메타자료 만료확인 0:04:14 이전인: 2022년 09월 28일 (수) 오전 03시 03분 55초.
========================== 이름 & 요약과 일치하는 항목: audit ==========================
audit.x86_64 : User space tools for kernel auditing
audit.src : User space tools for kernel auditing
audit-debuginfo.i686 : Debug information for package audit
audit-debuginfo.x86_64 : Debug information for package audit
audit-debugsource.i686 : Debug sources for package audit
audit-debugsource.x86_64 : Debug sources for package audit
audit-libs.x86_64 : Dynamic library for libaudit
audit-libs.i686 : Dynamic library for libaudit
audit-libs-debuginfo.i686 : Debug information for package audit-libs
audit-libs-debuginfo.x86_64 : Debug information for package audit-libs
audit-libs-devel.i686 : Header files for libaudit
audit-libs-devel.x86_64 : Header files for libaudit
pgaudit-debuginfo.x86_64 : Debug information for package pgaudit
pgaudit-debugsource.x86_64 : Debug sources for package pgaudit
python3-audit.x86_64 : Python3 bindings for libaudit
python3-audit-debuginfo.i686 : Debug information for package python3-audit
python3-audit-debuginfo.x86_64 : Debug information for package python3-audit
rsyslog-mmaudit.x86_64 : Message modification module supporting Linux audit format
rsyslog-mmaudit-debuginfo.x86_64 : Debug information for package rsyslog-mmaudit
sos-audit.noarch : Audit use of some commands for support purposes
```

유저 UID 를 통한 확인 방법

```
# ausearch -ui $UID --interpret
----
type=DAEMON_START msg=audit(2022년 09월 28일 01:42:53.647:2054) : op=start ver=3.0.7 format=enriched kerne
----
type=SERVICE_START msg=audit(2022년 09월 28일 01:42:53.666:5) : pid=1 uid=root auid=unset ses=unset subj=s
----
type=PROCTITLE msg=audit(2022년 09월 28일 01:42:53.714:6) : proctitle=/sbin/auditctl -R /etc/audit/audit.r
type=SYSCALL msg=audit(2022년 09월 28일 01:42:53.714:6) : arch=x86_64 syscall=sendto success=yes exit=60 a
type=CONFIG_CHANGE msg=audit(2022년 09월 28일 01:42:53.714:6) : op=set audit_backlog_limit=8192 old=64 aui
----
type=PROCTITLE msg=audit(2022년 09월 28일 01:42:53.731:7) : proctitle=/sbin/auditctl -R /etc/audit/audit.r
type=SYSCALL msg=audit(2022년 09월 28일 01:42:53.731:7) : arch=x86_64 syscall=sendto success=yes exit=60 a
type=CONFIG_CHANGE msg=audit(2022년 09월 28일 01:42:53.731:7) : op=set audit_failure=1 old=1 auid=unset se
----
type=PROCTITLE msg=audit(2022년 09월 28일 01:42:53.734:8) : proctitle=/sbin/auditctl -R /etc/audit/audit.r
type=SYSCALL msg=audit(2022년 09월 28일 01:42:53.734:8) : arch=x86_64 syscall=sendto success=yes exit=60 a
type=CONFIG_CHANGE msg=audit(2022년 09월 28일 01:42:53.734:8) : op=set audit_backlog_wait_time=60000 old=6
----
type=SERVICE_START msg=audit(2022년 09월 28일 01:42:53.755:9) : pid=1 uid=root auid=unset ses=unset subj=s
----
type=SYSTEM_BOOT msg=audit(2022년 09월 28일 01:42:53.767:10) : pid=1040 uid=root auid=unset ses=unset subj
----
```