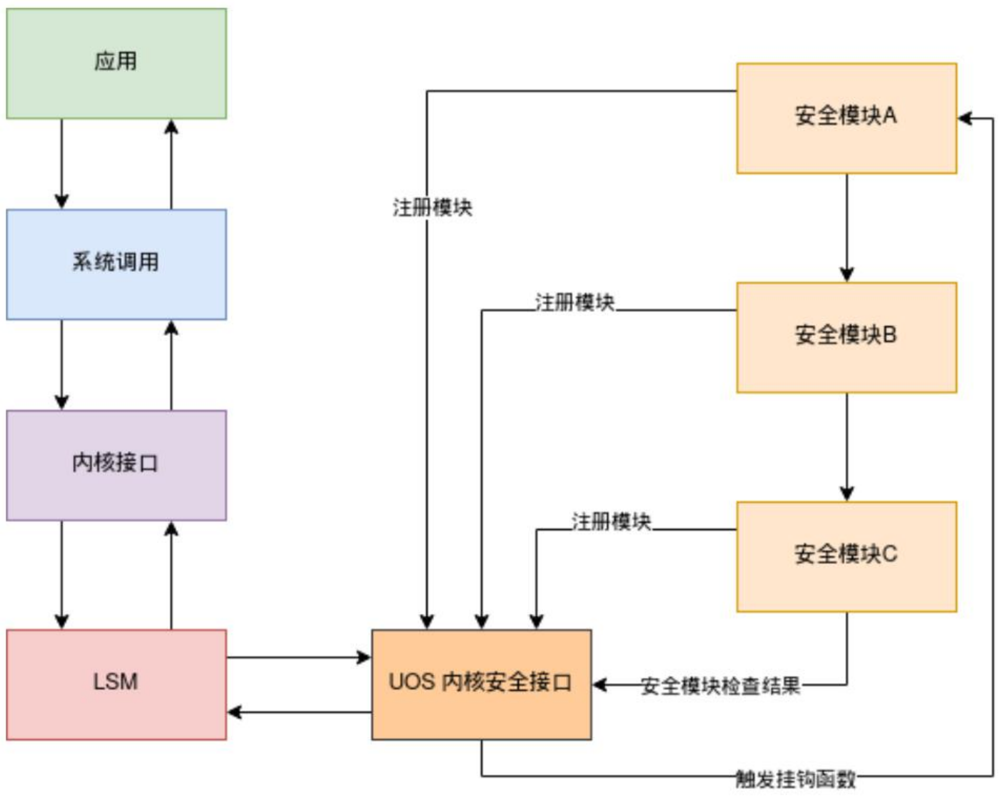
UOS LSM Hook Manager

设计描述

uos 以设计稳定可靠、易于开发者使用的内核安全接口为目标，基于 Linux 安全模块(Linux Security Module,LSM)开发了 hookmanager 模块提供接口给开发者使用，达到动态注册/移除 LSM Hook 的目的，其架构如下图所示：



uos 提供的 hookmanager 通过延用 LSM 提供的安全接口以确保接口的多样性,同时对 LSM 的调用机制进行封装,确保 LSM 的变更对开发者无感,hookmanager 会适配 LSM 接口的变更从而向开发者提供稳定的使用接口，让开发者无需关注于 LSM 底层机制。

总的来说，uos 提供的 hookmanger 安全接口具有如下有点：

1、可同时运行多个安全模块，易于扩充

2、接口简洁，易于理解和使用

3、便于以普通模块方式开发安全模块，易于开发和调试

4、性能开销低，基于 LSM 实现，运行高效。

接口描述

uos 主要提供了以下接口供开发者开发安全模块：

```
int uos_hook_register(enum UOS_HOOK_LIST hook_id, struct uos_hook_cb_entry *entry);
int uos_hook_cancel(enum UOS_HOOK_LIST hook_id, char *owner);
```

这套接口实现了 hook 的注册/注销接口，开发者使用 uos_hook_register 接口进行 hook 注册，该 hook 在对应场景下会自动被触发。开发者使用 uos_hook_cancel 接口注销对应的 hook。

可以通过以下方式判断系统是否启用 hookmanager 功能：

1. 确认/sys/kernel/security/hookmanager/version 文件是否存在,如果存在则说明 hookmanager 功能已启用,否则说明未启用;

2. 确认/boot/config-`uname -r`中 CONFIG_SECURITY_HOOKMANAGER=y 是否配置，已配置表示 hookmanager 已启用,否则说明未启用;

确认 hookmanger 启用后，可通过 hookmanager 提供的 uos_hook_register 接口注册 hook，当前支持动态注册/移除的 hook 列表如下：

```
UOS_AUDIT_RULE_FREE,
UOS_AUDIT_RULE_INIT,
UOS_AUDIT_RULE_KNOWN,
UOS_AUDIT_RULE_MATCH,
UOS_BINDER_SET_CONTEXT_MGR,
UOS_BINDER_TRANSACTION,
UOS_BINDER_TRANSFER_BINDER,
UOS_BINDER_TRANSFER_FILE,
UOS_BPF,
UOS_BPF_MAP,
UOS_BPF_MAP_ALLOC_SECURITY,
UOS_BPF_MAP_FREE_SECURITY,
UOS_BPF_PROG,
UOS_BPF_PROG_ALLOC_SECURITY,
UOS_BPF_PROG_FREE_SECURITY,
UOS_BPRM_CHECK_SECURITY,
UOS_BPRM_COMMITTED_CREDS,
UOS_BPRM_COMMITTING_CREDS,
UOS_BPRM_SET_CREDS,
UOS_CAPABLE,
UOS_CAPGET,
UOS_CAPSET,
UOS_CRED_ALLOC_BLANK,
UOS_CRED_FREE,
UOS_CRED_GETSECID,
UOS_CRED_PREPARE,
UOS_CRED_TRANSFER,
UOS_DENTRY_CREATE_FILES_AS,
UOS_DENTRY_INIT_SECURITY,
UOS_D_INSTANTIATE,
UOS_FILE_ALLOC_SECURITY,
UOS_FILE_FCNTL,
UOS_FILE_FREE_SECURITY,
UOS_FILE_IOCTL,
UOS_FILE_LOCK,
UOS_FILE_MPROTECT,
UOS_FILE_OPEN,
UOS_FILE_PERMISSION,
UOS_FILE_RECEIVE,
UOS_FILE_SEND_SIGIOTASK,
UOS_FILE_SET_FOWNER,
UOS_GETPROCATTR,
UOS_IB_ALLOC_SECURITY,
UOS_IB_ENDPORT_MANAGE_SUBNET,
UOS_IB_FREE_SECURITY,
UOS_IB_PKEY_ACCESS,
UOS_INET_CONN_ESTABLISHED,
UOS_INET_CONN_REQUEST,
UOS_INET_CSK_CLONE,
UOS_INODE_ALLOC_SECURITY,
UOS_INODE_COPY_UP,
UOS_INODE_COPY_UP_XATTR,
UOS_INODE_CREATE,
UOS_INODE_FOLLOW_LINK,
UOS_INODE_FREE,
UOS_INODE_FREE_SECURITY,
UOS_INODE_GETATTR,
UOS_INODE_GETSECCTX,
UOS_INODE_GETSECID,
UOS_INODE_GETSECURITY,
UOS_INODE_GETXATTR,
UOS_INODE_INVALIDATE_SECCTX,
UOS_INODE_KILLPRIV,
UOS_INODE_LINK,
UOS_INODE_LISTSECURITY,
UOS_INODE_LISTXATTR,
UOS_INODE_MKDIR,
UOS_INODE_MKNOD,
UOS_INODE_NEED_KILLPRIV,
UOS_INODE_NOTIFYSECCTX,
UOS_INODE_PERMISSION,
UOS_INODE_POST_SETXATTR,
UOS_INODE_READLINK,
UOS_INODE_REMOVEEXATTR,
UOS_INODE_RENAME,
UOS_INODE_RMDIR,
UOS_INODE_SETATTR,
UOS_INODE_SETSECCTX,
UOS_INODE_SETSECURITY,
UOS_INODE_SETXATTR,
UOS_INODE_SYMLINK,
UOS_INODE_UNLINK,
UOS_IPC_GETSECID,
UOS_IPC_PERMISSION,
UOS_ISMACLABEL,
UOS_KERNEL_ACT_AS,
UOS_KERNEL_CREATE_FILES_AS,
UOS_KERNEL_LOAD_DATA,
UOS_KERNEL_MODULE_REQUEST,
UOS_KERNEL_POST_READ_FILE,
UOS_KERNEL_READ_FILE,
UOS_KEY_ALLOC,
UOS_KEY_FREE,
UOS_KEY_GETSECURITY,
UOS_KEY_PERMISSION,
UOS_LOCKED_DOWN,
UOS_MMAP_ADDR,
UOS_MMAP_FILE,
UOS_MSG_MSG_ALLOC_SECURITY,
UOS_MSG_MSG_FREE_SECURITY,
UOS_MSG_QUEUE_ALLOC_SECURITY,
UOS_MSG_QUEUE_ASSOCIATE,
UOS_MSG_QUEUE_FREE_SECURITY,
UOS_MSG_QUEUE_MSGCTL,
UOS_MSG_QUEUE_MSGRCV,
UOS_MSG_QUEUE_MSGSEND,
UOS_NETLINK_SEND,
UOS_PATH_CHMOD,
UOS_PATH_CHOWN,
UOS_PATH_CHROOT,
UOS_PATH_LINK,
UOS_PATH_MKDIR,
UOS_PATH_MKNOD,
UOS_PATH_RENAME,
UOS_PATH_RMDIR,
UOS_PATH_SYMLINK,
UOS_PATH_TRUNCATE,
UOS_PATH_UNLINK,
UOS_PTRACE_ACCESS_CHECK,
UOS_PTRACE_TRACEME,
UOS_QUOTACTL,
UOS_QUOTA_ON,
UOS_RELEASE_SECCTX,
UOS_REQ_CLASSIFY_FLOW,
UOS_SB_ALLOC_SECURITY,
UOS_SB_CLONE_MNT_OPTS,
UOS_SB_COPY_DATA,
UOS_SB_FREE_SECURITY,
UOS_SB_KERN_MOUNT,
UOS_SB_MOUNT,
UOS_SB_PARSE_OPTS_STR,
UOS_SB_PIVOTROOT,
UOS_SB_REMOUNT,
UOS_SB_SHOW_OPTIONS,
UOS_SB_STATFS,
UOS_SB_UMOUNT,
UOS_SCTP_ASSOC_REQUEST,
UOS_SCTP_BIND_CONNECT,
UOS_SCTP_SK_CLONE,
UOS_SECCTX_TO_SECID,
UOS_SECID_TO_SECCTX,
UOS_SECMARK_REFCOUNT_DEC,
UOS_SECMARK_REFCOUNT_INC,
UOS_SECMARK_RELABEL_PACKET,
UOS_SEM_ALLOC_SECURITY,
UOS_SEM_ASSOCIATE,
UOS_SEM_FREE_SECURITY,
UOS_SEM_SEMCTL,
UOS_SEM_SEMOP,
UOS_SETPROCATTR,
UOS_SETTIME,
UOS_SHM_ALLOC_SECURITY,
UOS_SHM_ASSOCIATE,
UOS_SHM_FREE_SECURITY,
UOS_SHM_SHMAT,
UOS_SHM_SHMCTL,
UOS_SK_ALLOC_SECURITY,
UOS_SK_CLONE_SECURITY,
UOS_SK_FREE_SECURITY,
UOS_SK_GETSECID,
UOS_SOCKET_ACCEPT,
UOS_SOCKET_BIND,
UOS_SOCKET_CONNECT,
UOS_SOCKET_CREATE,
UOS_SOCKET_GETPEERNAME,
UOS_SOCKET_GETPEERSEC_DGRAM,
UOS_SOCKET_GETPEERSEC_STREAM,
UOS_SOCKET_GETSOCKNAME,
UOS_SOCKET_GETSOCKOPT,
UOS_SOCKET_LISTEN,
UOS_SOCKET_POST_CREATE,
UOS_SOCKET_RECVMSG,
UOS_SOCKET_SENDMSG,
UOS_SOCKET_SETSOCKOPT,
UOS_SOCKET_SHUTDOWN,
UOS_SOCKET_SOCKETPAIR,
UOS_SOCKET_SOCK_RCV_SKB,
UOS_SOCK_GRAFT,
UOS_SYSLOG,
UOS_TASK_ALLOC,
UOS_TASK_FIX_SETUID,
UOS_TASK_FREE,
UOS_TASK_GETIOPRIO,
UOS_TASK_GETPGID,
UOS_TASK_GETSCHEDULER,
UOS_TASK_GETSECID,
UOS_TASK_GETSID,
UOS_TASK_KILL,
UOS_TASK_MOVEMEMORY,
UOS_TASK_PRCTL,
UOS_TASK_PRLIMIT,
UOS_TASK_SETIOPRIO,
UOS_TASK_SETNICE,
UOS_TASK_SETPGID,
UOS_TASK_SETRLIMIT,
UOS_TASK_SETSCHEDULER,
UOS_TASK_TO_INODE,
UOS_TUN_DEV_ALLOC_SECURITY,
UOS_TUN_DEV_ATTACH,
UOS_TUN_DEV_ATTACH_QUEUE,
UOS_TUN_DEV_CREATE,
UOS_TUN_DEV_FREE_SECURITY,
UOS_TUN_DEV_OPEN,
UOS_UFILE_CLOSE,
UOS_UNIX_MAY_SEND,
UOS_UNIX_STREAM_CONNECT,
UOS_VM_ENOUGH_MEMORY,
UOS_XFRM_DECODE_SESSION,
UOS_XFRM_POLICY_ALLOC_SECURITY,
UOS_XFRM_POLICY_CLONE_SECURITY,
UOS_XFRM_POLICY_DELETE_SECURITY,
UOS_XFRM_POLICY_FREE_SECURITY,
UOS_XFRM_POLICY_LOOKUP,
UOS_XFRM_STATE_ALLOC,
UOS_XFRM_STATE_ALLOC_ACQUIRE,
UOS_XFRM_STATE_DELETE_SECURITY,
UOS_XFRM_STATE_FREE_SECURITY,
UOS_XFRM_STATE_POL_FLOW_MATCH,

UOS_HOOK_NONE
```

与上述 hook 对应的 LSM Hook 函数原型如下：

```

void (*audit_rule_free)(void *lsmrule);
int (*audit_rule_init)(u32 field, u32 op, char *rulestr, void **lsmrule);
int (*audit_rule_known)(struct audit_krule *krule);
int (*audit_rule_match)(u32 secid, u32 field, u32 op, void *lsmrule, struct audit_context *actx);

int (*binder_set_context_mgr)(struct task_struct *mgr);
int (*binder_transaction)(struct task_struct *from, struct task_struct *to);
int (*binder_transfer_binder)(struct task_struct *from,struct task_struct *to);
int (*binder_transfer_file)(struct task_struct *from,struct task_struct *to,struct file *file);

int (*bpf)(int cmd, union bpf_attr *attr, unsigned int size);
int (*bpf_map_alloc_security)(struct bpf_map *map);
void (*bpf_map_free_security)(struct bpf_map *map);
int (*bpf_map)(struct bpf_map *map, fmode_t fmode);
int (*bpf_prog_alloc_security)(struct bpf_prog_aux *aux);
void (*bpf_prog_free_security)(struct bpf_prog_aux *aux);
int (*bpf_prog)(struct bpf_prog *prog);

int (*bprm_check_security)(struct linux_binprm *bprm);
void (*bprm_committed_creds)(struct linux_binprm *bprm);
void (*bprm_committing_creds)(struct linux_binprm *bprm);
int (*bprm_set_creds)(struct linux_binprm *bprm);

int (*capable)(const struct cred *cred, struct user_namespace *ns, int cap, unsigned int opts);
int (*capget)(struct task_struct *target, kernel_cap_t *effective, kernel_cap_t *inheritable, kernel_cap_t *permitted);
int (*capset)(struct cred *new, const struct cred *old, const kernel_cap_t *effective, const kernel_cap_t *inheritable, const kernel_cap_t *permitted);
int (*cred_alloc_blank)(struct cred *cred, gfp_t gfp);

void (*cred_free)(struct cred *cred);
void (*cred_getsecid)(const struct cred *c, u32 *secid);
int (*cred_prepare)(struct cred *new, const struct cred *old, gfp_t gfp);
void (*cred_transfer)(struct cred *new, const struct cred *old);

int (*dentry_create_files_as)(struct dentry *dentry, int mode, struct qstr *name, const struct cred *old, struct cred *new);
int (*dentry_init_security)(struct dentry *dentry, int mode, const struct qstr *name, void **ctx, u32 *ctxlen);
void (*d_instantiate)(struct dentry *dentry, struct inode *inode);

int (*file_alloc_security)(struct file *file);
int (*file_fcntl)(struct file *file, unsigned int cmd, unsigned long arg);
```

```
void (*file_free_security)(struct file *file);
int (*file_ioctl)(struct file *file, unsigned int cmd, unsigned long arg);
int (*file_lock)(struct file *file, unsigned int cmd);
int (*file_mprotect)(struct vm_area_struct *vma, unsigned long reqprot,
unsigned long prot);
int (*file_open)(struct file *file);
int (*file_permission)(struct file *file, int mask);
int (*file_receive)(struct file *file);
int (*file_send_sigiotask)(struct task_struct *tsk, struct fown_struct *fown, int
sig);
void (*file_set_fowner)(struct file *file);

int (*getprocattr)(struct task_struct *p, char *name, char **value);

int (*ib_alloc_security)(void **sec);
int (*ib_endport_manage_subnet)(void *sec, const char *dev_name, u8
port_num);
void (*ib_free_security)(void *sec);
int (*ib_pkey_access)(void *sec, u64 subnet_prefix, u16 pkey);

void (*inet_conn_established)(struct sock *sk, struct sk_buff *skb);
int (*inet_conn_request)(struct sock *sk, struct sk_buff *skb, struct
request_sock *req);
void (*inet_csk_clone)(struct sock *newsk, const struct request_sock *req);

int (*inode_alloc_security)(struct inode *inode);
int (*inode_copy_up)(struct dentry *src, struct cred **new);
int (*inode_copy_up_xattr)(const char *name);
int (*inode_create)(struct inode *dir, struct dentry *dentry, umode_t mode);
int (*inode_follow_link)(struct dentry *dentry, struct inode *inode, bool rcu);
void (*inode_free_security)(struct inode *inode);
int (*inode_getattr)(const struct path *path);
int (*inode_getsecctx)(struct inode *inode, void **ctx, u32 *ctxlen);
void (*inode_getsecid)(struct inode *inode, u32 *secid);
int (*inode_getsecurity)(struct inode *inode, const char *name, void **buffer,
bool alloc);
int (*inode_getxattr)(struct dentry *dentry, const char *name);
int (*inode_init_security)(struct inode *inode, struct inode *dir, const struct qstr
*qstr, const char **name, void **value, size_t *len);
void (*inode_invalidate_secctx)(struct inode *inode);
int (*inode_killpriv)(struct dentry *dentry);
int (*inode_link)(struct dentry *old_dentry, struct inode *dir, struct dentry
*new_dentry);
int (*inode_listsecurity)(struct inode *inode, char *buffer, size_t buffer_size);
int (*inode_listxattr)(struct dentry *dentry);
int (*inode_mkdir)(struct inode *dir, struct dentry *dentry, umode_t mode);
int (*inode_mknod)(struct inode *dir, struct dentry *dentry, umode_t mode,
dev_t dev);
int (*inode_need_killpriv)(struct dentry *dentry);
int (*inode_notifysecctx)(struct inode *inode, void *ctx, u32 ctxlen);
int (*inode_permission)(struct inode *inode, int mask);
void (*inode_post_setxattr)(struct dentry *dentry, const char *name, const void
*value, size_t size, int flags);
int (*inode_readlink)(struct dentry *dentry);
int (*inode_removexattr)(struct dentry *dentry, const char *name);
int (*inode_rename)(struct inode *old_dir, struct dentry *old_dentry, struct
inode *new_dir, struct dentry *new_dentry);
int (*inode_rmdir)(struct inode *dir, struct dentry *dentry);
int (*inode_setattr)(struct dentry *dentry, struct iattr *attr);
int (*inode_setsecctx)(struct dentry *dentry, void *ctx, u32 ctxlen);
int (*inode_setsecurity)(struct inode *inode, const char *name, const void
*value, size_t size, int flags);
int (*inode_setxattr)(struct dentry *dentry, const char *name, const void *value,
size_t size, int flags);
int (*inode_symlink)(struct inode *dir, struct dentry *dentry, const char
*old_name);
int (*inode_unlink)(struct inode *dir, struct dentry *dentry);
void (*ipc_getsecid)(struct kern_ipc_perm *ipcp, u32 *secid);
int (*ipc_permission)(struct kern_ipc_perm *ipcp, short flag);

int (*ismaclabel)(const char *name);

int (*kernel_act_as)(struct cred *new, u32 secid);
int (*kernel_create_files_as)(struct cred *new, struct inode *inode);
int (*kernel_load_data)(enum kernel_load_data_id id);
int (*kernel_module_request)(char *kmod_name);
int (*kernel_post_read_file)(struct file *file, char *buf, loff_t size, enum
kernel_read_file_id id);
int (*kernel_read_file)(struct file *file, enum kernel_read_file_id id);

int (*key_alloc)(struct key *key, const struct cred *cred, unsigned long flags);
void (*key_free)(struct key *key);
int (*key_getsecurity)(struct key *key, char **_buffer);
int (*key_permission)(key_ref_t key_ref, const struct cred *cred, unsigned
perm);

int (*locked_down)(enum lockdown_reason what);
int (*mmap_addr)(unsigned long addr);
int (*mmap_file)(struct file *file, unsigned long reqprot, unsigned long prot,
unsigned long flags);

int (*msg_msg_alloc_security)(struct msg_msg *msg);
void (*msg_msg_free_security)(struct msg_msg *msg);
int (*msg_queue_alloc_security)(struct kern_ipc_perm *msq);
int (*msg_queue_associate)(struct kern_ipc_perm *msq, int msqflg);
void (*msg_queue_free_security)(struct kern_ipc_perm *msq);
int (*msg_queue_msgctl)(struct kern_ipc_perm *msq, int cmd);
int (*msg_queue_msgrcv)(struct kern_ipc_perm *msq, struct msg_msg *msg,
struct task_struct *target, long type, int mode);
int (*msg_queue_msgsnd)(struct kern_ipc_perm *msq, struct msg_msg *msg,
int msqflg);

int (*netlink_send)(struct sock *sk, struct sk_buff *skb);

int (*path_chmod)(const struct path *path, umode_t mode);
int (*path_chown)(const struct path *path, kuid_t uid, kgid_t gid);
int (*path_chroot)(const struct path *path);
int (*path_link)(struct dentry *old_dentry, const struct path *new_dir, struct
dentry *new_dentry);
int (*path_mkdir)(const struct path *dir, struct dentry *dentry, umode_t mode);
int (*path_mknod)(const struct path *dir, struct dentry *dentry, umode_t mode,
unsigned int dev);
int (*path_rename)(const struct path *old_dir, struct dentry *old_dentry, const
struct path *new_dir, struct dentry *new_dentry);
int (*path_rmdir)(const struct path *dir, struct dentry *dentry);
int (*path_symlink)(const struct path *dir, struct dentry *dentry, const char
*old_name);
int (*path_truncate)(const struct path *path);
int (*path_unlink)(const struct path *dir, struct dentry *dentry);

int (*ptrace_access_check)(struct task_struct *child, unsigned int mode);
int (*ptrace_traceme)(struct task_struct *parent);

int (*quotactl)(int cmds, int type, int id, struct super_block *sb);
int (*quota_on)(struct dentry *dentry);

void (*release_secctx)(char *secdata, u32 seclen);
void (*req_classify_flow)(const struct request_sock *req, struct flowi *fl);
int (*sb_alloc_security)(struct super_block *sb);
int (*sb_clone_mnt_opts)(const struct super_block *oldsb, struct super_block
*newsb, unsigned long kern_flags, unsigned long *set_kern_flags);
```

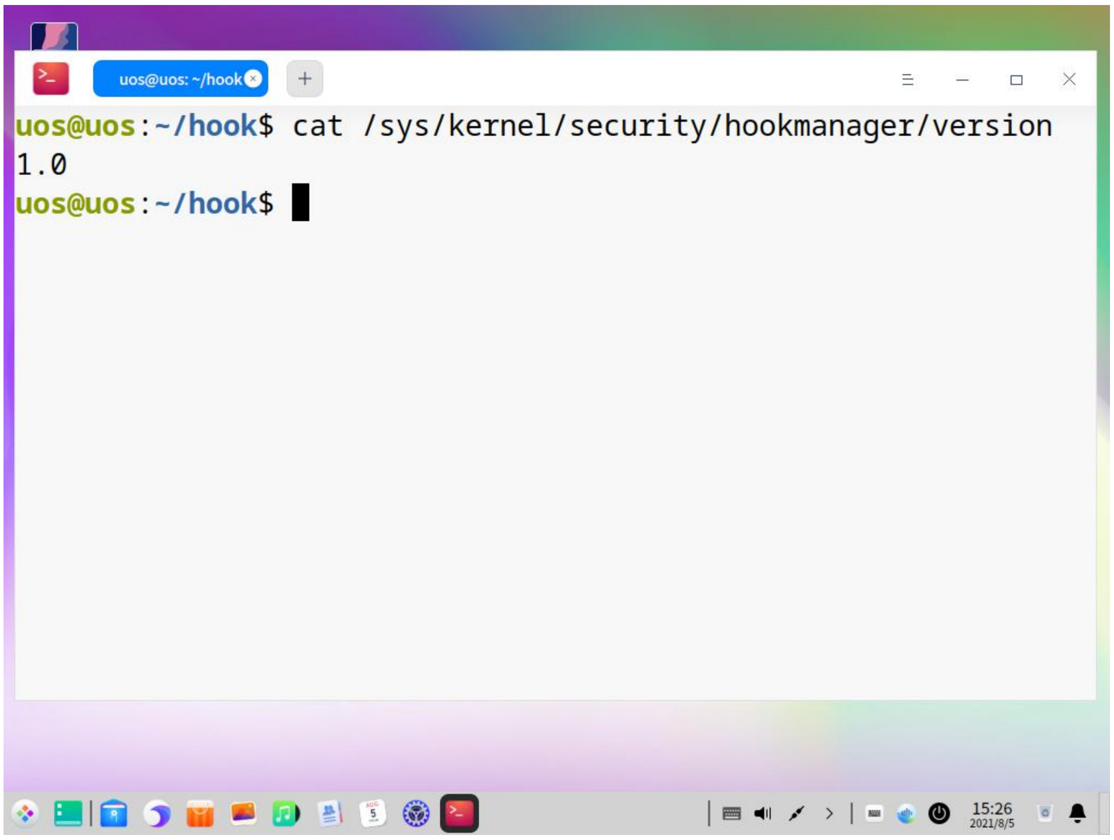
```
int (*sb_copy_data)(char *orig, char *copy);
void (*sb_free_security)(struct super_block *sb);
int (*sb_kern_mount)(struct super_block *sb, int flags, void *data);
int (*sb_mount)(const char *dev_name, const struct path *path, const char
*type, unsigned long flags, void *data);
int (*sb_parse_opts_str)(char *options, struct security_mnt_opts *opts);
int (*sb_pivotroot)(const struct path *old_path, const struct path *new_path);
int (*sb_remount)(struct super_block *sb, void *data);
int (*sb_show_options)(struct seq_file *m, struct super_block *sb);
int (*sb_statfs)(struct dentry *dentry);
int (*sb_umount)(struct vfsmount *mnt, int flags);

int (*sctp_assoc_request)(struct sctp_endpoint *ep, struct sk_buff *skb);
int (*sctp_bind_connect)(struct sock *sk, int optname, struct sockaddr *address,
int addrlen);
void (*sctp_sk_clone)(struct sctp_endpoint *ep, struct sock *sk, struct sock
*newsk);

int (*secctx_to_secid)(const char *secdata, u32 seclen, u32 *secid);
int (*secid_to_secctx)(u32 secid, char **secdata, u32 *seclen);

void (*secmark_refcount_dec)(void);
void (*secmark_refcount_inc)(void);

int (*secmark_relabel_packet)(u32 secid);
int (*sem_alloc_security)(struct kern_ipc_perm *sma);
int (*sem_associate)(struct kern_ipc_perm *sma, int semflg);
void (*sem_free_security)(struct kern_ipc_perm *sma);
int (*sem_semctl)(struct kern_ipc_perm *sma, int cmd);
int (*sem_semop)(struct kern_ipc_perm *sma, struct sembuf *sops, unsigned
nsops, int alter);

int (*setprocattr)(const char *name, void *value, size_t size);

int (*settime)(const struct timespec64 *ts, const struct timezone *tz);

int (*shm_alloc_security)(struct kern_ipc_perm *shp);
int (*shm_associate)(struct kern_ipc_perm *shp, int shmflg);
void (*shm_free_security)(struct kern_ipc_perm *shp);
int (*shm_shmat)(struct kern_ipc_perm *shp, char __user *shmaddr, int
shmflg);
int (*shm_shmctl)(struct kern_ipc_perm *shp, int cmd);
int (*sk_alloc_security)(struct sock *sk, int family, gfp_t priority);
void (*sk_clone_security)(const struct sock *sk, struct sock *newsk);
void (*sk_free_security)(struct sock *sk);
void (*sk_getsecid)(struct sock *sk, u32 *secid);

int (*socket_accept)(struct socket *sock, struct socket *newsock);
int (*socket_bind)(struct socket *sock, struct sockaddr *address, int addrlen);
int (*socket_connect)(struct socket *sock, struct sockaddr *address, int
addrlen);
int (*socket_create)(int family, int type, int protocol, int kern);
int (*socket_getpeername)(struct socket *sock);
int (*socket_getpeersec_dgram)(struct socket *sock, struct sk_buff *skb, u32
*secid);
int (*socket_getpeersec_stream)(struct socket *sock, char __user *optval, int
__user *optlen, unsigned len);
int (*socket_getsockname)(struct socket *sock);
int (*socket_getsockopt)(struct socket *sock, int level, int optname);
int (*socket_listen)(struct socket *sock, int backlog);
int (*socket_post_create)(struct socket *sock, int family, int type, int protocol,
int kern);
int (*socket_recvmsg)(struct socket *sock, struct msghdr *msg, int size, int
flags);
int (*socket_sendmsg)(struct socket *sock, struct msghdr *msg, int size);
int (*socket_setsockopt)(struct socket *sock, int level, int optname);
int (*socket_shutdown)(struct socket *sock, int how);
int (*socket_socketpair)(struct socket *socka, struct socket *sockb);
int (*socket_sock_rcv_skb)(struct sock *sk, struct sk_buff *skb);
void (*sock_graft)(struct sock *sk, struct socket *parent);

int (*syslog)(int type);

int (*task_alloc)(struct task_struct *task, unsigned long clone_flags);
int (*task_fix_setuid)(struct cred *new, const struct cred *old, int flags);
void (*task_free)(struct task_struct *task);
int (*task_getioprio)(struct task_struct *p);
int (*task_getpgid)(struct task_struct *p);
int (*task_getscheduler)(struct task_struct *p);
void (*task_getsecid)(struct task_struct *p, u32 *secid);
int (*task_getsid)(struct task_struct *p);
int (*task_kill)(struct task_struct *p, struct siginfo *info, int sig, const struct cred
*cred);
int (*task_movememory)(struct task_struct *p);
int (*task_prctl)(int option, unsigned long arg2, unsigned long arg3, unsigned
long arg4, unsigned long arg5);
```

```
int (*task_prlimit)(const struct cred *cred, const struct cred *tcred, unsigned int flags);
int (*task_setioprio)(struct task_struct *p, int ioprio);
int (*task_setnice)(struct task_struct *p, int nice);
int (*task_setpgid)(struct task_struct *p, pid_t pgid);
int (*task_setrlimit)(struct task_struct *p, unsigned int resource, struct rlimit *new_rlim);
int (*task_setscheduler)(struct task_struct *p);
void (*task_to_inode)(struct task_struct *p, struct inode *inode);

int (*tun_dev_alloc_security)(void **security);
int (*tun_dev_attach_queue)(void *security);
int (*tun_dev_attach)(struct sock *sk, void *security);
int (*tun_dev_create)(void);
void (*tun_dev_free_security)(void *security);
int (*tun_dev_open)(void *security);

int (*unix_may_send)(struct socket *sock, struct socket *other);
int (*unix_stream_connect)(struct sock *sock, struct sock *other, struct sock *newsk);

void (*uos_file_close)(struct file *file);

int (*vm_enough_memory)(struct mm_struct *mm, long pages);
int (*xfrm_decode_session)(struct sk_buff *skb, u32 *secid, int ckall);
int (*xfrm_policy_alloc_security)(struct xfrm_sec_ctx **ctxp, struct xfrm_user_sec_ctx *sec_ctx, gfp_t gfp);
int (*xfrm_policy_clone_security)(struct xfrm_sec_ctx *old_ctx, struct xfrm_sec_ctx **new_ctx);
int (*xfrm_policy_delete_security)(struct xfrm_sec_ctx *ctx);
void (*xfrm_policy_free_security)(struct xfrm_sec_ctx *ctx);
int (*xfrm_policy_lookup)(struct xfrm_sec_ctx *ctx, u32 fl_secid, u8 dir);
int (*xfrm_state_alloc_acquire)(struct xfrm_state *x, struct xfrm_sec_ctx *polsec, u32 secid);
int (*xfrm_state_alloc)(struct xfrm_state *x, struct xfrm_user_sec_ctx *sec_ctx);
int (*xfrm_state_delete_security)(struct xfrm_state *x);
void (*xfrm_state_free_security)(struct xfrm_state *x);
int (*xfrm_state_pol_flow_match)(struct xfrm_state *x, struct xfrm_policy *xp, const struct flowi *fl);
```

使用说明

下面以示例介绍 hookmanager 的使用流程。

1、确认 hookmanager 开启,如下图所示



```
uos@uos:~/hook$ cat /sys/kernel/security/hookmanager/version
1.0
uos@uos:~/hook$ 
```

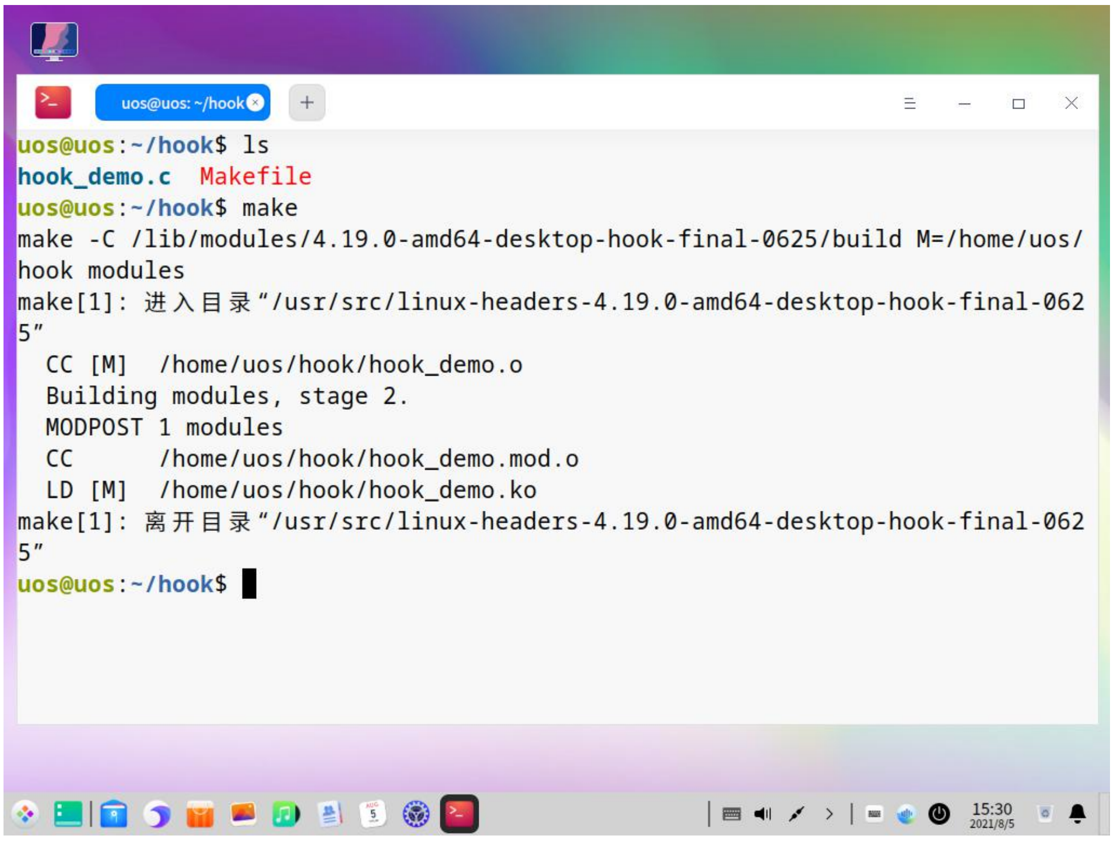
2、编写 hook_demo.c 测试模块，编译成 ko 文件并完成注册

hook_demo.c	

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/lsm_uos_hook_manager.h>
#include <linux/proc_fs.h>
#include <linux/fs_struct.h>
#include <linux/version.h>

#define MODULE_NAME "hook_demo"

typedef struct td_hook_entry {
    int hook_id;
    struct uos_hook_cb_entry cb;
}td_hook_entry_t;

int td_inode_alloc_security(struct inode * inode)
{
    printk("current: %s %s: test\n", current->comm, __FUNCTION__);
    return 0;
}

static td_hook_entry_t entries[] = {
    {
        .hook_id = UOS_INODE_ALLOC_SECURITY,
        .cb =
            {
                .owner = MODULE_NAME,
                .cb_addr = (unsigned long)td_inode_alloc_security,//hook

                .ret_type = UOS_HOOK_RET_TY_INT,//返回值类型


                .arg_len = 1,//参数个数

            },
    },
    {
        .hook_id = UOS_HOOK_NONE,
    },
};

int td_uos_manager_register_hook(void)
{
    int i = 0, j = 0;
    int error = 0;

    for (; entries[i].hook_id != UOS_HOOK_NONE; i++) {
        error = uos_hook_register(entries[i].hook_id, &entries[i].cb);
        if (error) {
            printk("Failed to register hook %d, hook_id = %d\n", i,
entries[i].hook_id);
            break;
        }
    }

    if (entries[i].hook_id == UOS_HOOK_NONE)
        return 0;
    for (; j < i; j++) {
        error = uos_hook_cancel(entries[j].hook_id, entries[j].cb.owner);
        if (error)
            printk("Failed to cancel hook %d\n", j);
    }

    return -1;
}

void td_uos_manager_cancel_hook(void)
{
    int i = 0;
    int error = 0;

    for (; entries[i].hook_id != UOS_HOOK_NONE; i++) {
        error = uos_hook_cancel(entries[i].hook_id, entries[i].cb.owner);
        if (error)
            printk("Failed to cancel hook %d, hook_id = %d\n", i,
entries[i].hook_id);
        }
}

int __init init_module(void)
{
    int error = 0;

#ifdef CONFIG_SECURITY_PATH
    printk("CONFIG_SECURITY_PATH is defined\n");
#endif


    pr_info("start to init uos hook demo\n");
    error = td_uos_manager_register_hook();
    if (error) {
        pr_info("failed to registe hook\n");
        return -1;
    }

        pr_info("finish to init uos hook demo\n");
    return 0;
}

void __exit cleanup_module(void)
{
	td_uos_manager_cancel_hook();
	pr_info("exit the uos hook demo\n");
}

MODULE_LICENSE("GPL");
MODULE_AUTHOR("jouyouyun");
MODULE_DESCRIPTION("The demo for uos lsm hook manager");
```

Makefile	

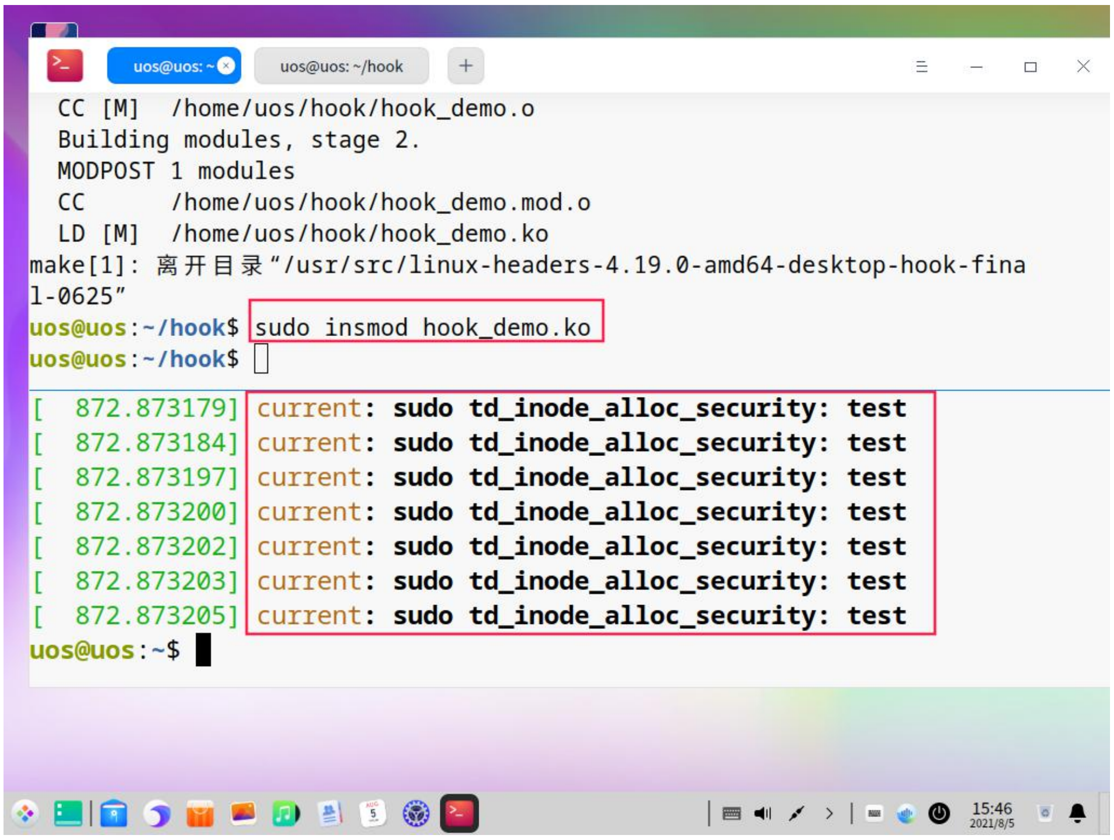
```
ifneq ($(KERNELRELEASE),)
obj-m:=hook_demo.o
else

KDIR := /lib/modules/$(shell uname -r)/build
PWD:=$(shell pwd)

all:
	make -C $(KDIR) M=$(PWD) modules
clean:
	rm -f *.ko *.o *.symvers *.cmd *.cmd.o
endif
```



```
uos@uos:~/hook$ ls
hook_demo.c  Makefile
uos@uos:~/hook$ make
make -C /lib/modules/4.19.0-amd64-desktop-hook-final-0625/build M=/home/uos/
hook modules
make[1]: 进入目录“/usr/src/linux-headers-4.19.0-amd64-desktop-hook-final-062
5”
  CC [M]  /home/uos/hook/hook_demo.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/uos/hook/hook_demo.mod.o
  LD [M]  /home/uos/hook/hook_demo.ko
make[1]: 离开目录“/usr/src/linux-headers-4.19.0-amd64-desktop-hook-final-062
5”
uos@uos:~/hook$ 
```

3、验证

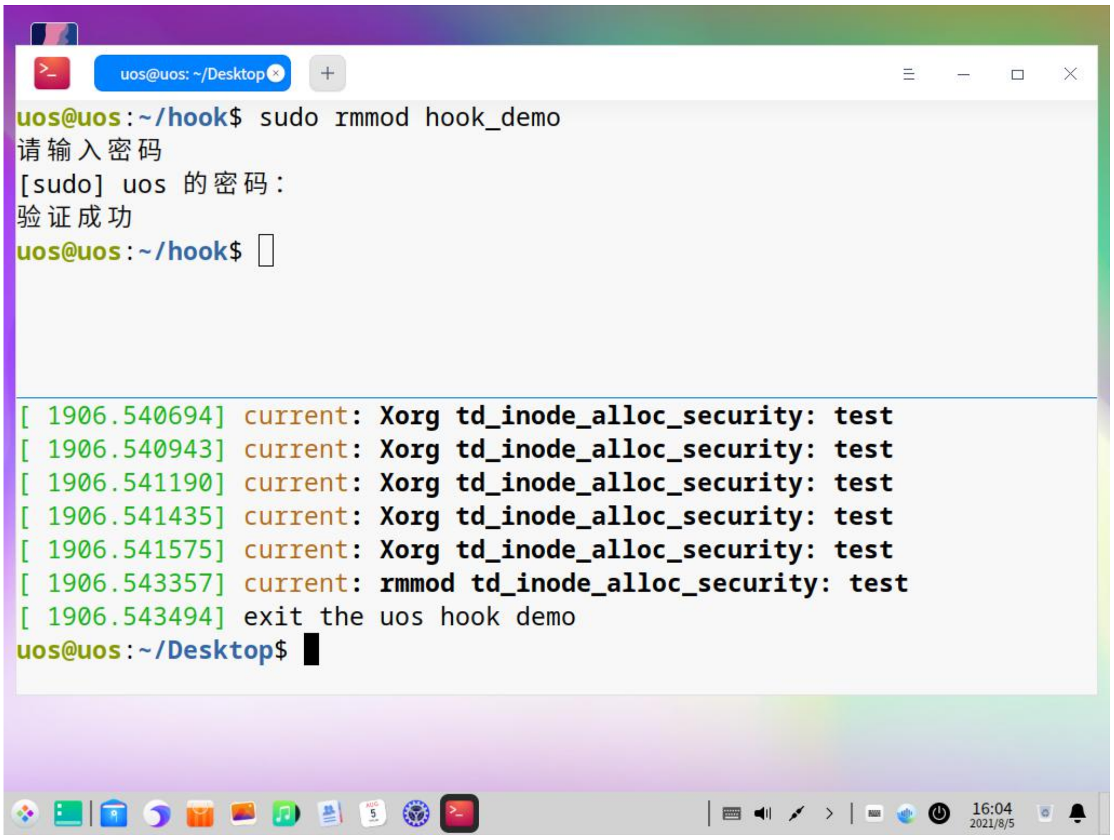
如果 sudo insmod hook_demo.ko 执行成功，说明我们成功注册 hook，在满足某些场景下该 hook 会通过 LSM 机制被自动调用，示例中可通过 sudo dmesg 观察到如下图所示的输出，说明我们通过 hook_demo.ko 注册的 td_inode_alloc_security 函数已被触发。



```
  CC [M]  /home/uos/hook/hook_demo.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/uos/hook/hook_demo.mod.o
  LD [M]  /home/uos/hook/hook_demo.ko
make[1]: 离开目录“/usr/src/linux-headers-4.19.0-amd64-desktop-hook-fina
l-0625”
uos@uos:~/hook$ sudo insmod hook_demo.ko
uos@uos:~/hook$ 
[  872.873179] current: sudo td_inode_alloc_security: test
[  872.873184] current: sudo td_inode_alloc_security: test
[  872.873197] current: sudo td_inode_alloc_security: test
[  872.873200] current: sudo td_inode_alloc_security: test
[  872.873202] current: sudo td_inode_alloc_security: test
[  872.873203] current: sudo td_inode_alloc_security: test
[  872.873205] current: sudo td_inode_alloc_security: test
uos@uos:~$ 
```

4、注销 hook

通过 sudo rmmod hook_demo 命令移除 hook_demo.ko，本例中该模块被移除时会调用 hookmanager 提供的注销接口 uos_hook_cancel 完成 hook 的注销，如下图所示：



```
uos@uos:~/hook$ sudo rmmod hook_demo
请输入密码
[sudo] uos 的密码:
验证成功
uos@uos:~/hook$ 
```

```
[ 1906.540694] current: Xorg td_inode_alloc_security: test
[ 1906.540943] current: Xorg td_inode_alloc_security: test
[ 1906.541190] current: Xorg td_inode_alloc_security: test
[ 1906.541435] current: Xorg td_inode_alloc_security: test
[ 1906.541575] current: Xorg td_inode_alloc_security: test
[ 1906.543357] current: rmmod td_inode_alloc_security: test
[ 1906.543494] exit the uos hook demo
uos@uos:~/Desktop$ 
```