Google | Calif

Mbed TLS 3.6.2

Security Audit Report

hacking-team@calif.io | July 3, 2025

Executive Summary

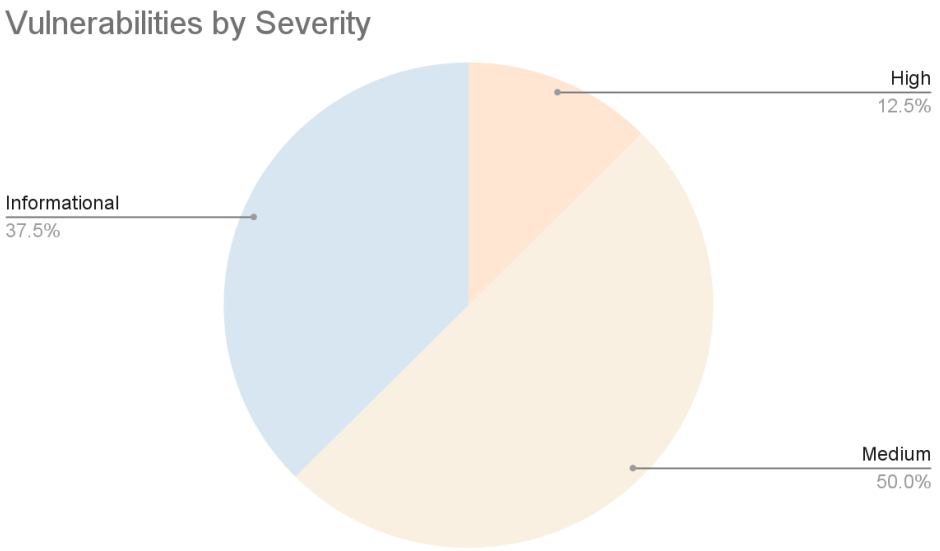
Synopsis

Mbed TLS is a lightweight, open-source cryptographic library for embedded systems, supporting TLS/SSL protocols, cryptographic algorithms, and compliance with standards like PKCS#1, PKCS#12, and X.509.

This report details a white-box security assessment of Mbed TLS version 3.6.2, commissioned by Google and performed by Calif. The audit, conducted from December 2024 to March 2025, focused on the library's implementation of X.509/ASN.1, RSA, ECDSA, and LMS functionalities through source code review, fuzzing, and testing.

The assessment identified eight vulnerabilities, including one of 'High' severity, four of 'Medium' severity, and three 'Informational' findings. These discoveries warrant proactive remediation to enhance the security posture of the library.

All identified vulnerabilities were disclosed to the Mbed TLS team in April 2025, and their respective severity levels were discussed and confirmed prior to this publication.



Key Findings

The most significant vulnerabilities discovered are:

- CVE-2025-47917: Misleading memory management in X.509 name parsing leading to arbitrary code execution

- CVE-2025-47917: Unchecked return value in LMS verification allows signature bypass
- CVE-2025-48965: Null pointer dereference in parsing X.509 distinguished names leading to DoS
- CVE-2025-49601: Out-of-bounds-read in LMS public key import leading to DoS or information disclosure
- CVE-2025-52497: Integer underflow in decoding PEM keys leading to DoS

Recommendations

To mitigate the identified vulnerabilities, all users of affected Mbed TLS versions should upgrade to version 3.6.4 or a subsequent release at the earliest opportunity.

Table Of Contents

Executive Summary 2
- Synopsis 2
- Key Findings 2
- Recommendations 3

Table Of Contents 4

Scope 5

Methodology 6

Findings 7
- 01. CVE-2025-47917: Misleading memory management in X.509 name parsing leading to arbitrary code execution 8
- 02. CVE-2025-49600: Unchecked return value in LMS verification allows signature bypass 10
- 03. CVE-2025-48965: Null pointer dereference in parsing X.509 distinguished names leading to DoS 13
- 04. CVE-2025-49601: Out-of-bounds read in LMS public key import leading to DoS or information disclosure 15
- 05. CVE-2025-52497: Integer underflow in decoding PEM keys leading to DoS 17
- 06. Accepting ECDSA signatures with leading zero bytes leading to signature malleability 19
- 07. Missing salt length validation in RSA-PSS signature verification 21
- 08. Narrowing type casting leading to LMS/LMSOTS public key malleability 24

Document Version History 27

Appendix 28
- A. Wycheproof test drivers 28
- B. Security considerations for LMS module 28

Scope

This report details a white-box security audit of the Mbed TLS cryptographic library, commissioned by Google. The objective was to identify security vulnerabilities in specific, high-risk components of the library through source code analysis and dynamic testing.

Audit Target

The assessment focused on a specific version of the Mbed TLS library, sourced directly from its official repository.

- **Library:** Mbed TLS
- **Version:** 3.6.2
- **Commit Hash:** `107ea89`

In-Scope Components

The following components and functionalities were the primary focus of this assessment:

- **X.509/ASN.1:** Certificate encoding and parsing logic.
- **RSA:** All digital signature operations.
- **ECDSA:** All operations.
- **LMS (Leighton-Micali Signatures):** All operations.

Out-of-Scope Components

To ensure a focused review, the following components were explicitly excluded from the scope of this audit:

- The TLS/DTLS protocol-level implementations.
- Random number generation modules (TRNG, CTR-DRBG).
- Build systems, documentation, and example programs.

Methodology

The security assessment was conducted as a white-box security audit, where our team had complete access to the Mbed TLS source code. The process was iterative and combined several analysis techniques to build a comprehensive understanding of the library's security posture.

Our methodology consists of the following steps:

- **Setup:** Compiled and built the latest Mbed TLS version.
- **Preliminary Research:** Studied historical vulnerabilities and known attack vectors.
- **Code Review:** Analyzed design documents and source code, with a focus on - implementation, cryptographic correctness, memory safety, and constant time constructions.
- **Fuzzing:** Performed fuzzing and Wycheproof testing on X.509 encoding/decoding, RSA, ECDSA and LMS. Relevant test drivers are detailed in Appendix A and can be found in this repository.

For each potential vulnerability identified, the team conducted a thorough validation process. This involved developing a proof-of-concept (PoC) to confirm the finding and accurately assess its impact, from denial of service to potential arbitrary code execution. All confirmed findings were then documented with a severity rating, a technical description, and actionable recommendations for remediation.

Following the validation phase, we reported all findings to the Mbed TLS team. We operated under a coordinated disclosure policy, providing detailed technical information for each vulnerability. This collaborative process included discussing the potential impact and agreeing on the final severity ratings presented in this report. We worked with the Mbed TLS team to ensure they had sufficient information to create and release patches.

Findings

ID	Finding	Severity
01	CVE-2025-47917: Misleading memory management in X.509 name parsing leading to arbitrary code execution	High
02	CVE-2025-49600: Unchecked return value in LMS verification allows signature bypass	Medium
03	CVE-2025-48965: Null pointer dereference in parsing X.509 distinguished names leading to DoS	Medium
04	CVE-2025-49601: Out-of-bounds read in LMS public key import leading to DoS or information disclosure	Medium
05	CVE-2025-52497: Integer underflow in decoding PEM keys leading to DoS	Medium
06	Accepting ECDSA signatures with leading zero bytes leading to signature malleability	Informational
07	Missing salt length validation in RSA-PSS signature verification	Informational
08	Narrowing type casting leading to LMS/LMSOTS public key malleability	Informational

01. CVE-2025-47917: Misleading memory management in X.509 name parsing leading to arbitrary code execution

Severity	High
CVSS 3.1	7.4 CVSS:3.1/AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H
Mbed TLS Advisory	https://mbed-tls.readthedocs.io/en/latest/security-advisories/mbedtls-security-advisory-2025-06-7/

Summary

A heap use-after-free vulnerability was discovered in the `MbedTLS_x509_string_to_names()` function due to misleading memory management. The function frees a memory structure that the calling application is likely to still hold a pointer to, based on the function's documentation. This can lead to a use-after-free or double-free condition, with a high risk of enabling arbitrary code execution.

Technical details

The function `MbedTLS_x509_string_to_names` converts Distinguished Name strings into a linked list of `MbedTLS_asn1_named_data`. Each invocation frees the objects in the linked list. When `MbedTLS_x509_string_to_names` is called multiple times, objects from previous calls are freed, leading to a heap use-after-free or double-free vulnerability.

This vulnerability exists in the `cert_write.c` example of Mbed TLS:

- The program uses `MbedTLS_x509_string_to_names` to parse Directory Names (DN) in the Subject Alternative Name (SAN). If the SAN contains multiple DNs, the function is called multiple times.
- Each subsequent call frees the objects created in previous calls.
- This causes a heap use-after-free issue when the program later calls `MbedTLS_x509write_crt_set_subject_alternative_name` to access the already-freed objects.

Impact

Depending on the application's code and the system's memory allocator implementation, this vulnerability can be exploited by an attacker who provides a specially crafted X.509 certificate. Successful exploitation could corrupt memory, leading to a denial of service (DoS) or, in the worst case, arbitrary code execution on the system processing the malicious certificate.

Steps to reproduce

1. Compile Mbed TLS with ASAN enabled.
2. Use the `cert_write` program located in the `programs/x509` directory, execute the following command:

```
None
./cert_write subject_key=<directory
mbedtls>/framework/data_files/rsa_pkcs1_1024_clear.pem
subject_name="CN=dw.yonan.net" issuer_crt=<directory
mbedtls>/framework/data_files/enco-ca-prstr.pem issuer_key=<directory
mbedtls>/framework/data_files/rsa_pkcs1_1024_clear.pem
not_before=20190210144406 not_after=20290210144406 md=SHA1 version=3
san="DN:CN=#0000;DN:CN=#0000"
```

3. Observe an ASAN log indicating a heap use after free vulnerability.

Recommendations

Ensure that when calling `MbedTLS_x509_string_to_names()` multiple times, the linked list parameter passed in each call is an independent variable and not shared across calls, to prevent the unintended freeing of these variables.

02. CVE-2025-49600: Unchecked return value in LMS verification allows signature bypass

Severity	Medium
CVSS 3.1	4.6 CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N
Mbed TLS Advisory	https://mbed-tls.readthedocs.io/en/latest/security-advisories/mbedtls-security-advisory-2025-06-3/

Summary

The Leighton-Micali Signature (LMS) system is a stateful hash-based signature scheme designed for post-quantum security, providing protection against attacks from future quantum computers. Mbed TLS's LMS implementation, `mbedtls_lms_verify`, fails to check the return value of the core function it calls, which can lead to incorrect signature verification when one of the function calls fails.

Technical details

The return values of calls to `create_merkle_leaf_value` and `create_merkle_internal_value` in `mbedtls_lms_verify` are not checked. These functions return an integer that indicates whether the call succeeded or not. A non-zero return value means the call failed and output might not have been written to the output buffer as expected. In this case, the output buffer (`Tc_candidate_root_node`) is not initialized, so the result of signature verification is unpredictable if one of the unchecked function calls fails.

```
C/C++
int mbedtls_lms_verify(const mbedtls_lms_public_t *ctx,
                       const unsigned char *msg, size_t msg_size,
                       const unsigned char *sig, size_t sig_size)
{
   unsigned int q_leaf_identifier;
   unsigned char Kc_candidate_ots_pub_key[MBEDTLS_LMOTS_N_HASH_LEN_MAX];
   unsigned char Tc_candidate_root_node[MBEDTLS_LMS_M_NODE_BYTES_MAX];

   ...

   create_merkle_leaf_value(
       &ctx->params,
       Kc_candidate_ots_pub_key,
       MERKLE_TREE_INTERNAL_NODE_AM(ctx->params.type) + q_leaf_identifier,
       Tc_candidate_root_node);

...
```

Within `create_merkle_leaf_value` and `create_merkle_internal_value`, there are three operations that can fail, i.e., `psa_hash_setup`, `psa_hash_update` and `psa_hash_finish`. The possible errors are noted in their declarations in psa/crypto.h.

Impact

This vulnerability could allow an attacker to bypass LMS signature verification.

Recommendations

Return values of all function calls should be checked, especially when they are part of a critical operation like signature verification. The compilation configuration of Mbed TLS does enable warnings for unused return values (`-Wall` is added). However, the functions `create_merkle_leaf_value` and `create_merkle_internal_value` are not declared with `__attribute__((warn_unused_result))` which is required for the compiler to detect when their return values are not used. This has been discussed in issue 3963 and the recommended solution is to annotate the critical functions with `MBEDTLS_CHECK_RETURN_CRITICAL` (see the example code snippet below).

```
C/C++
MBEDTLS_CHECK_RETURN_CRITICAL
static int create_merkle_leaf_value(const mbedtls_lms_parameters_t *params,
                                    unsigned char *pub_key,
                                    unsigned int r_node_idx,
                                    unsigned char *out)
```

Additionally, the status returned by `psa_hash_abort` is also ignored in `lms.c` and `lmots.c` . The CodeQL query below can be used to find function calls of which return values are unused.

```
None
/**
 * @name Unused Return Values
 * @description Finds function calls where the return value is not used
 * @kind problem
 * @problem.severity warning
 * @id cpp/unused-return-value
 */

 import cpp

 from FunctionCall call, Function f
 where
   // Get the function being called
   call.getTarget() = f and
```

```
    // Caller is not a test function
    not (call.getEnclosingFunction().getName().matches("%test%")) and
    // Check if function returns non-void value(s)
    not (f.getType() instanceof VoidType) and
    // Return value is unused
    call.getParent() instanceof ExprStmt
 select call,
    "Call to '" + f.getName() + "' at line " +
call.getLocation().getStartLine() + " in " + call.getFile().getBaseName() +
" is not used"
```

03. CVE-2025-48965: Null pointer dereference in parsing X.509 distinguished names leading to DoS

Severity	Medium
CVSS 3.1	5.3 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L
Mbed TLS Advisory	https://mbed-tls.readthedocs.io/en/latest/security-advisories/mbedtls-security-advisory-2025-06-6/

Summary

When converting X.505 distinguished names into a linked list of `MbedTLS_asn1_named_data` structures, Mbed TLS fails to validate the availability of the named value (`val`) field in the `MbedTLS_asn1_named_data` structure, resulting in a null pointer dereference.

Technical details

The function `MbedTLS_x509_string_to_names` converts X.505 distinguished names into a linked list of `MbedTLS_asn1_named_data` structures.

```
C/C++
typedef struct MbedTLS_asn1_named_data {
       MbedTLS_asn1_buf oid; /**< The object identifier. */
       MbedTLS_asn1_buf val; /**< The named value. */
       struct MbedTLS_asn1_named_data *next;
       unsigned char MbedTLS_PRIVATE(next_merged);
} MbedTLS_asn1_named_data;
```

The `MbedTLS_x509_string_to_names()` function parses the buffer to extract the object identifier and the name value. It then calls `MbedTLS_asn1_store_named_data` with the object identifier (including `oid` and `oid_len`) and the name value (including `val` and `val_len`).

```
C/C++
MbedTLS_asn1_named_data *MbedTLS_asn1_store_named_data(
        MbedTLS_asn1_named_data **head,
        const char *oid, size_t oid_len,
        const unsigned char *val,
        size_t val_len)
```

If `val_len` is zero, `MbedTLS_asn1_store_named_data` frees the `val` field but retains the `oid` entry in the list. Subsequent attempts to access the same `oid` encounter a null `val` pointer, leading to a crash.

Impact

The immediate impact of this vulnerability is a denial of service (DoS). Any remote attacker can crash a client or server that uses the vulnerable Mbed TLS library to parse X.509 certificates.

Steps to reproduce

1. Compile Mbed TLS with ASAN enabled.
2. Use the `cert_write` program located in the `programs/x509` directory, execute the following command:

```
None
./cert_write subject_key=<directory
mbedtls>/framework/data_files/rsa_pkcs1_1024_clear.pem
subject_name="CN=dw.yonan.net" issuer_crt=<directory
mbedtls>/framework/data_files/enco-ca-prstr.pem issuer_key=<directory
mbedtls>/framework/data_files/rsa_pkcs1_1024_clear.pem
not_before=20190210144406 not_after=20290210144406 md=SHA1 version=3
san="DN:R=\\\\000,R=#4500,R=3450"
```

3. Observe an ASAN log indicating an exception triggered by a segmentation fault (SEGV) on an unknown address: 0x000000000000.

Recommendations

Check that the `val` field in the `MbedTLS_asn1_named_data` struct is not null before performing any operations on it.

04. CVE-2025-49601: Out-of-bounds read in LMS public key import leading to DoS or information disclosure

Severity	Medium
CVSS 3.1	6.5 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L
Mbed TLS Advisory	https://mbed-tls.readthedocs.io/en/latest/security-advisories/mbedtls-security-advisory-2025-06-4/

Summary

The Leighton-Micali Signature (LMS) system is a stateful hash-based signature scheme designed for post-quantum security. When importing LMS public key, Mbed TLS reads 4 bytes from the input buffer before actually checking its length (`key_size`), which can lead to out-of-bounds read.

Technical details

The function `mbedtls_lms_import_public_key` takes in a buffer of bytes (`key`) and parses it into an object of type `mbedtls_lms_public_t`. The function does not check if the input buffer contains at least 4 bytes before reading a `uint32_t` from it, which might lead to out-of-bounds read if the input buffer is shorter than expected.

```
C/C++
int mbedtls_lms_import_public_key(mbedtls_lms_public_t *ctx,
                                  const unsigned char *key, size_t key_size)
{
   mbedtls_lms_algorithm_type_t type;
   mbedtls_lmots_algorithm_type_t otstype;

   type = (mbedtls_lms_algorithm_type_t) MBEDTLS_GET_UINT32_BE(key,
PUBLIC_KEY_TYPE_OFFSET);
...
```

Impact

This out-of-bounds read has two potential consequences:

- Denial of Service (DoS): The read from an unmapped memory region will likely cause a segmentation fault, crashing the application.
- Information Disclosure: If the adjacent memory is mapped and contains sensitive data from other parts of the application, that data could be read and potentially leaked.

An attacker could exploit this by providing a malformed public key to any system that parses them, leading to service disruption or a data leak.

Recommendations

The function `mbedtls_lms_import_public_key` should enforce that the input buffer has at least 4 bytes before reading an `uint32_t` from it. For generality, we should check if the input buffer is long enough to contain the encoded algorithm type based on its offset (`PUBLIC_KEY_TYPE_OFFSET`) and length (`MBEDTLS_LMS_TYPE_LEN`).

```
C/C++
int mbedtls_lms_import_public_key(mbedtls_lms_public_t *ctx,
                                  const unsigned char *key, size_t key_size)
{
       mbedtls_lms_algorithm_type_t type;
       mbedtls_lmots_algorithm_type_t otstype;

       if (key_size < PUBLIC_KEY_TYPE_OFFSET + MBEDTLS_LMS_TYPE_LEN) {
              return MBEDTLS_ERR_LMS_BAD_INPUT_DATA;
       }
       type = (mbedtls_lms_algorithm_type_t) MBEDTLS_GET_UINT32_BE(key,
PUBLIC_KEY_TYPE_OFFSET);
       ...
```

05. CVE-2025-52497: Integer underflow in decoding PEM keys leading to DoS

Severity	Medium
CVSS 3.1	6.5 CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:L
Mbed TLS Advisory	https://mbed-tls.readthedocs.io/en/latest/security-advisories/mbedtls-security-advisory-2025-06-2/

Summary

Inadequate buffer validation in `MbedTLS_pem_read_buffer` leads to an integer underflow, potentially causing a denial-of-service (DoS) or limited information disclosure.

Technical details

The function MbedTLS_pk_parse_key handles the parsing of private or public keys in PEM or DER format. When processing PEM files, it calls `MbedTLS_pem_read_buffer`, which parses a PEM string as follows:

1. Calls MbedTLS_base64_decode with dst set to NULL to calculate the buffer size needed for decoding.
2. Allocates memory for the buffer based on the calculated length.
3. Calls MbedTLS_base64_decode again, this time decoding the Base64 data into the allocated buffer.
4. Calls pem_check_pkcs_padding to verify PKCS padding and adjust the data length.

The issue arises when `MbedTLS_base64_decode` is called with a source buffer (`src`) size smaller than 4 bytes:

- When `src` is smaller than 4 bytes, the first call to `MbedTLS_base64_decode` calculates a length of 1.
- Memory is allocated accordingly, but during the second call, decoding is skipped because processing occurs in 4-byte blocks. This sets the output length (`olen`) to 0.
- Passing this zero-length value to `pem_check_pkcs_padding` causes an integer underflow when it subtracts 1, leading to invalid memory access or out-of-bounds errors.

Impact

This heap-based integer overflow is likely to corrupt heap metadata, which will cause the application to crash.

Steps to reproduce

1. Compile Mbed TLS with ASAN enabled.
2. Use the `pk_sign` program located in the `programs/pkey` directory, execute the following command:

```
None
./pk_sign private_key_poc.pem test
```

The `private_key_poc.pem` file contains the following content:

```
None
00000000: 2d2d 2d2d 2d42 4547 494e 2045 4320 5052  -----BEGIN EC PR
00000010: 4956 4154 4520 4b45 592d 2d2d 2d2d 0a50  IVATE KEY-----.P
00000020: 726f 632d 5479 7065 3a20 342c 454e 4352  roc-Type: 4,ENCR
00000030: 5950 5445 440a 4445 4b2d 496e 666f 3a20  YPTED.DEK-Info:
00000040: 4145 532d 3132 382d 4342 432c 3742 4133  AES-128-CBC,7BA3
00000050: 3844 4530 3046 3637 3835 3145 3432 3037  8DE00F67851E4207
00000060: 3231 3638 3039 4333 4242 3135 0a0a 3851  216809C3BB15..8Q
00000070: 2d2d 2d2d 2d45 4e44 2045 4320 5052 4956  -----END EC PRIV
00000080: 4154 4520 4b45 592d 2d2d 2d2d 0a         ATE KEY-----.
```

3. Observe an ASAN log indicating a heap buffer overflow vulnerability.

Recommendations

Validate the source buffer size before Base64 decoding and ensure the output length (`olen`) is non-zero before further operations.

06. Accepting ECDSA signatures with leading zero bytes leading to signature malleability

Severity	Informational
CVSS 3.1	N/A

Summary

When verifying ECDSA signatures, Mbed TLS accepts signatures with leading zero bytes, leading to signature malleability.

Technical details

A DER-compliant signature must be represented as:

```
SEQUENCE {
    INTEGER r,
    INTEGER s
}
```

Each INTEGER (i.e., `r` and `s`) must:

- Be a positive integer.
- Not have unnecessary leading zero bytes, except when a leading zero byte is required to avoid a negative value (MSB = 1).

When verifying an ECDSA signature, the function `MbedTLS_asn1_get_mpi` is used to parse the INTEGER values (`r` and `s`) from the DER-encoded signature.

The function does not check for unnecessary leading zero bytes in `r` or `s`, which violates DER encoding rules. This enables signature malleability, wherein an attacker can derive multiple valid signatures for the same message by appending or removing leading zero bytes in `r` or `s`.

Signature malleability undermines non-repudiation by allowing the creation of altered but valid signatures. In systems where signature uniqueness determines validity (e.g., preventing replay attacks), this vulnerability can have severe consequences.

Steps to reproduce

1. Compile the file `ecdsa-verify-der-test.c` provided in **Appendix A**.
2. Execute the compiled file using the test vector ecdsa_secp192k1_sha256_test.json.
3. Check the program output. You will notice that certain test cases fail, as shown below:

```
None
Running Test Group 1...
Test 1 passed valid.
...
...
Test 4 failed -0x0000.
Test 5 failed -0x0000.
Test 6 passed invalid.
...
```

4. Inspect the content of the file `ecdsa_secp192k1_sha256_test.json`. It reveals that the ECDSA implementation has accepted signatures containing leading byte 0.

```
None
...
{
    "tcId": 4,
    "comment": "long form encoding of length of sequence",
    "msg": "313233343030",
    "sig":
"308135021900d781b83e5846f00406b23fd03959a9a050ff008a07b0a81402186134acee459
6d1f2be2a2898d9e66fc8677227b149b0b3f7",
    "result": "invalid",
    "flags": [
        "BER"
    ]
},
{
    "tcId": 5,
    "comment": "length of sequence contains leading 0",
    "msg": "313233343030",
    "sig":
"30820035021900d781b83e5846f00406b23fd03959a9a050ff008a07b0a81402186134acee4
596d1f2be2a2898d9e66fc8677227b149b0b3f7",
    "result": "invalid",
    "flags": [
        "BER"
    ]
}
...
```

Recommendations

Add a check in `MbedTLS_asn1_get_mpi` to validate and reject unnecessary leading zero bytes in `r` and `s` to ensure compliance with the DER standard.

07. Missing salt length validation in RSA-PSS signature verification

Severity	Informational
CVSS 3.1	N/A

Summary

When verifying RSA-PSS signatures, Mbed TLS failed to properly check the size of salt length (`s_len`), leading to acceptance of invalid signatures.

Technical details

The `MbedTLS_rsa_rsassa_pss_verify_ext` function verifies RSA-PSS signatures, with the `expected_salt_len` parameter indicating the salt length. When `MbedTLS_RSA_SALT_LEN_ANY` is passed as `expected_salt_len`, the function skips validation for salt length, violating PKCS#1 v2.1 standards. This allows signatures with mismatched salt lengths to pass verification, reducing cryptographic integrity.

```
C/C++
int MbedTLS_rsa_rsassa_pss_verify_ext(MbedTLS_rsa_context *ctx,
                                      MbedTLS_md_type_t md_alg,
                                      unsigned int hashlen,
                                      const unsigned char *hash,
                                      MbedTLS_md_type_t mgf1_hash_id,
                                      int expected_salt_len,
                                      const unsigned char *sig)
```

Steps to reproduce

1. Compile the file `rsassa-pss-verify-test.c` provided in **Appendix A**.
2. Execute the compiled file using the test vector rsa_pss_2048_sha1_mgf1_20_test.json.
3. Check the program output. You will notice that certain test cases fail, as shown below:

```
None
Running Test Group 1...
...
Test case 44: PASS (invalid)
Test case 45: PASS (invalid)
Test case 46: FAIL (invalid)
```

```
Test case 47: FAIL (invalid)
...
```

4. Inspect the content of the file `rsa_pss_2048_sha1_mgf1_20_test.json`. It indicates that the RSA PSS implementation has accepted signatures with a modified `s_len`.

None

```
...
{
    "tcId": 46,
    "comment": "s_len changed to 0",
    "msg": "313233343030",
    "sig":
"0dd16c3ccc10b280bc36c0104e7c5fe47107c1ba511d197357aa7a537e90f079a00385744a8
5a070804e9134a75fa73bf1c053162ed2e622ef1d3a1b9f117c47a7b68f9e1000bf851570987
fbb9f8b5fd2bfc058f95f2bd12ca977e44f596df0a1c48de9d0c840732d94ac2f11156c9e739
de8df8931efae8aa42cd6254b3fbe1405313e8b19ca86045edf87631bd219f6923b8dfd783ac
9e7c913cf7348c7b5028b478898a366b893938a94d2fea92e78001ae2baaf5dc0c31e9b0d461
9e0fde45414b0c5863c8826406d87b48fbe0c52164d0a8d1fd00b883ddae8e1235c846d51e5c
b20d724576dfdfa01d15f47cbac56b17543fcdfe81dd70dca545ffdd0",
    "result": "invalid",
    "flags": [
        "WeakHash"
    ]
},
{
    "tcId": 47,
    "comment": "s_len changed to 32",
    "msg": "313233343030",
    "sig":
"18bd764174873263341771a783534921ccc5f3395ca96a3a57706bab1f78905c002f3cd6e17
91e238a8ba6b9fddd74d4e758527bc3ce76a2d9b37e130bccb8e235f8388e54152f447346a58
0f4808bcc17dfa51c69c2625efee575314b609b8e30f1caf822411ba1cecbb2c295c76620ea1
b64fadbd4a8b52ea398f60538f3a19fc9c7c5f7b7de802e16c290d635278590bc367b935eb72
09547aa1cb378e54e2e383d8a2c67a69e790fcab540a51cf756c86a5fd0f337b14246eda65e9
b8b85e6ebe62e89156a387e9d1b7206da72c0822d20a20637391956d473fea426505e6a54126
0b92cb4b66980592dcf92bfa71d264c575496dcc098bac82edd5c6dc1",
    "result": "invalid",
    "flags": [
        "WeakHash"
    ]
}
...
```

Recommendations

Enforce strict salt length validation in `MbedTLS_rsa_rsassa_pss_verify_ext` by ensuring it matches the hash length when `MbedTLS_RSA_SALT_LEN_ANY` is used, or avoid using `MbedTLS_RSA_SALT_LEN_ANY` in this context altogether.

08. Narrowing type casting leading to LMS/LMSOTS public key malleability

Severity	Informational
CVSS 3.1	N/A

Summary

Under certain compilation configuration, Mbed TLS accepts several encodings for the algorithm type of an LMS/LMOTS public key, which makes the public key malleable.

Technical details

The `mbedtls_lms_import_public_key` function reads an `uint32_t` from the input buffer and explicitly casts it to `mbedtls_lms_algorithm_type_t`, an enum. When enum packing is enabled for optimization (e.g., `-fshort-enums` flag is added) the size of an enum is enough to contain all of its possible values. In the case of `mbedtls_lms_algorithm_type_t`, only one byte is needed because the type has only one value that is 0x6. Therefore, when casting `uint32_t` to `mbedtls_lms_algorithm_type_t`, three MSBs might be truncated, leading to multiple encodings for one algorithm type of public key. Similarly, `mbedtls_lmots_import_public_key` also has the same issue with casting `uint32_t` to `mbedtls_lmots_algorithm_type_t`.

```
C/C++
int mbedtls_lms_import_public_key(mbedtls_lms_public_t *ctx,
                                  const unsigned char *key, size_t key_size)
{
   mbedtls_lms_algorithm_type_t type;
   mbedtls_lmots_algorithm_type_t otstype;

   type = (mbedtls_lms_algorithm_type_t) MBEDTLS_GET_UINT32_BE(key,
PUBLIC_KEY_TYPE_OFFSET);

   if (type != MBEDTLS_LMS_SHA256_M32_H10) {
       return MBEDTLS_ERR_LMS_BAD_INPUT_DATA;
   }
...
```

```
C/C++
typedef enum {
       MBEDTLS_LMS_SHA256_M32_H10 = 0x6,
```

```
} mbedtls_lms_algorithm_type_t;
```

Steps to reproduce

1. Compile the following program with the `-fshort-enums` flag added.

```C/C++
#include <stdio.h>
#include "mbedtls/lms.h"
#include <string.h>


static inline uint32_t mbedtls_get_unaligned_uint32(const void *p)
{
   uint32_t r;
   memcpy(&r, p, sizeof(r));
   return r;
}

int main() {
   printf("Size of mbedtls_lms_algorithm_type_t: %lu\n",
sizeof(mbedtls_lms_algorithm_type_t));

   char *bytes = "\x06\x00\x00\x00";
   mbedtls_lms_algorithm_type_t type = (mbedtls_lms_algorithm_type_t)
mbedtls_get_unaligned_uint32(bytes);
   printf("Comparison 1 returns %d\n", type == MBEDTLS_LMS_SHA256_M32_H10);

   bytes = "\x06\x05\x04\x03";
   type = (mbedtls_lms_algorithm_type_t)
mbedtls_get_unaligned_uint32(bytes);
   printf("Comparison 2 returns %d\n", type == MBEDTLS_LMS_SHA256_M32_H10);

   return 0;
}
```

2. Run the compiled file on a little-endian platform and you will get the following output.

```None
Size of mbedtls_lms_algorithm_type_t: 1
Comparison 1 returns 1
```

```
Comparison 2 returns 1
```

Recommendations

Instead of casting `uint32_t` to the enum representing algorithm type, we should cast the expected enum value to `uint32_t` and compare it with the `uint32_t` read from input (see the example code snippet below).

```
C/C++
if (MBEDTLS_GET_UINT32_BE(key, PUBLIC_KEY_TYPE_OFFSET) !=
(uint32_t)MBEDTLS_LMS_SHA256_M32_H10) {
       return MBEDTLS_ERR_LMS_BAD_INPUT_DATA;
}
```

Document Version History

Release date	Changes
Jan 13, 2025	Initial version
Mar 5, 2025	Added LMS audit results
July 3, 2025	Added CVEs and Mbed TLS official advisories

Appendix

A. Wycheproof test drivers

As part of this security audit, Calif developed Wycheproof test drivers for RSA and ECDSA, focusing on key functionalities such as decryption/encryption and signature verification. The test vectors used in these tests are sourced from Wycheproof Test Vectors.

Below are the specific tests and their corresponding functions used for RSA and ECDSA:

RSA Tests

- PKCS#1 v2.1 RSAES-PKCS1-V1_5-DECRYPT: Test the decryption functionality using the RSA PKCS#1 v1.5 standard.
- PKCS#1 v2.1 RSAES-OAEP-DECRYPT: Test the decryption functionality using the RSA Optimal Asymmetric Encryption Padding (OAEP) scheme.
- PKCS#1 v2.1 RSASSA-PKCS1-v1_5-VERIFY: Verify digital signatures using the RSA PKCS#1 v1.5 standard.
- PKCS#1 v2.1 RSASSA-PSS-VERIFY: Verify digital signatures using the RSA Probabilistic Signature Scheme (PSS).

ECDSA Tests

- Verify an ECDSA signature using the P1316 standard.
- Verify an ECDSA signature encoded in DER format.

B. Security considerations for LMS module

- An attacker can forge the signature for a different message given a valid signature for one message if the checksum `Cksm(Q)` is not added when calculating the digit array (see Step 5 of Algorithm 3 of RFC 8554 corresponding to `create_digit_array_with_checksum`). This issue has been discussed in section 9.3 of RFC 8554. In v3.6.2 of Mbed TLS, the checksum has been correctly implemented.

- If an LMOTS private key is reused, an attacker can bypass the security of the checksum mentioned above to forge a signature for a chosen message. This issue can be avoided by maintaining the integrity of the LMS private context (`mbedtls_lms_private_t`), especially its next usable key index (the `q_next_usable_key` field of `mbedtls_lms_private_t`).

- Persistence of LMS private contexts into nonvolatile memory should be handled by users of the Mbed TLS library as the library does not guarantee integrity of LMS private contexts when corruptions happen or when the keys are explicitly freed (e.g., by calling `mbedtls_lms_private_free`), which can lead to LMOTS private key reuse and signature forgery. This issue has been mentioned in

section 9.2 of RFC 8554. Due to this reason, some functions are marked as not intended for production use as of v3.6.2 - e.g., `mbedtls_lms_generate_private_key`, `mbedtls_lms_sign`, `mbedtls_lmots_verify`, `mbedtls_lmots_generate_private_key` (explicitly stated in the comment of these functions). According to this change log, LMS signature verification is production-ready, while generation is not.

- It is stated in the document of Mbed TLS that "Mbed TLS 2.x, and in Mbed TLS 3.x at the time of writing, the PSA API is not thread-safe". This also applies to the LMS module as it utilizes PSA API for many cryptographic operations. It is recommended that a single thread should only use or access one context (usually passed to a function through a parameter named `ctx`) at a same time, unless the documentation for the functions that access the shared context explicitly states the function is thread-safe, or you perform explicit locking yourself (perhaps in a wrapper function).