ESP-IDF BluFi Reference Application Vulnerability Disclosure

Espressif Systems

Version 1.1 – March 6, 2025

Prepared By
James Chambers

1 Summary

The following vulnerabilities are in the ESP-IDF BluFi Reference Application (https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/blufi).

These vulnerabilities affect at least ESP-IDF versions 5.0.7, 5.1.5, 5.2.3, and 5.3.1, which are the latest release versions at time of submission.

Memory corruption vulnerabilities were verified using an ESP32-S3-DevKitC-1. Proof-of-concepts for triggering the buffer overflows were implemented with Python 3 code running on macOS.

BluFi application projects for each release version are included in the `blufi_apps.zip` attachment. Proof-of-concept exploit code is included in the `blufi_exploits.zip` attachment.

2 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

Title	Status	ID	Risk
Buffer Overflows in WiFi Credential Setting Commands	Reported	WXR	High
Buffer Overflow in Diffie-Hellman Key Negotiation Commands	Reported	V3L	High
Unauthenticated Key Exchange	Reported	YL7	High
Out-of-Bounds Reads in Diffie-Hellman Key Negotiation	Reported	YQG	Medium
Key Exchange Algorithm Doesn't Meet Best Practices	Reported	2BB	Medium
Lack of Cryptographic Authentication for Secure Messages	Reported	A2M	Low

3 Finding Details

High

Buffer Overflows in WiFi Credential Setting Commands

Overall Risk	High	**Finding ID**	NCC-BluFi-Ref-WXR
Impact	High	**Category**	Data Validation
Exploitability	High	**Status**	Reported

CVSS 8.7 (CVSS:4.0/AV:A/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)

Impact

An attacker within Bluetooth range can achieve arbitrary code execution on an ESP32 device running the BluFi reference code by exploiting the WiFi credential setting commands.

Description

Handlers for the WiFi credential setting commands `ESP_BLUFI_EVENT_RECV_STA_SSID`, `ESP_BLUFI_EVENT_RECV_STA_PASSWD`, `ESP_BLUFI_EVENT_RECV_SOFTAP_SSID`, and `ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD` unsafely copy the input credential buffer into a global variable buffer. While these commands use `strncpy` to copy the buffer, the supplied length limit matches the length of the input, rather than the length of the destination buffer.

For example, `ESP_BLUFI_EVENT_RECV_STA_SSID` copies the input SSID from `param->sta_ssid.ssid` into `sta_config.sta.ssid`, but provides the length limit `param->sta_ssid.ssid_len` instead of `sizeof(sta_config.sta.ssid)`:

```
case ESP_BLUFI_EVENT_RECV_STA_SSID:
      strncpy((char *)sta_config.sta.ssid, (char *)param->sta_ssid.ssid, param->sta_ssid.ssid
      ↳ _len);
      sta_config.sta.ssid[param->sta_ssid.ssid_len] = '\0';
```

Each of these unsafe `strncpy` operations allows an attacker to overflow the buffers in global memory. Which variables are adjacent to the vulnerable buffers is determined at compile time. An attacker could overwrite adjacent function pointers or data structures that can be manipulated to take control of the instruction pointer, and then achieve arbitrary code execution through techniques such as Return Oriented Programming.

We noted that in ESP-IDF 5.0 builds the WiFi config structures are located very close to the `btc_profile_cb_tab` callback table. This allows for a simple overflow to overwrite the BluFi callback function pointer entry, which allows the attacker to take control of the instruction pointer.

In ESP-IDF 5.3 builds there was more unrelated data located between the WiFi config structures and the `btc_profile_cb_tab` callback table, and overwriting random data structure pointers caused the device to crash. In this case we were able to use a different technique to take control of the instruction pointer.

In ESP-IDF 5.0 and 5.3 builds, the pointer to the BluFi security state structure, `blufi_sec`, was located just after the WiFi config structures. By setting up a fake `blufi_security` struct in the SoftAP config structure and then overwriting the `blufi_sec` pointer to redirect to this fake data structure, we were able to create a write-what-where primitive triggered via the `SEC_TYPE_DH_PARAM_DATA` security negotiation command. Using the write-what-where primitive we then overwrote a function pointer in `btc_profile_cb_tab` to take control of the instruction pointer, as before.

Recommendation

Handlers for `ESP_BLUFI_EVENT_RECV_STA_SSID`, `ESP_BLUFI_EVENT_RECV_STA_PASSWD`, `ESP_BLUFI_EVENT_RECV_SOFTAP_SSID`, and `ESP_BLUFI_EVENT_RECV_SOFTAP_PASSWD` should all be updated to fix the vulnerable `strncpy` calls.

The call to each `strncpy` should use the size of the destination buffer (e.g., `sizeof(sta_config.sta.ssid)`) as the length parameter, and add a null terminator at `buf_len - 1` when necessary (e.g., `sizeof(sta_config.sta.ssid) - 1`).

Location

`examples/bluetooth/blufi/main/blufi_example_main.c`

High

Buffer Overflow in Diffie-Hellman Key Negotiation Commands

Overall Risk	High	**Finding ID**	NCC-BluFi-Ref-V3L
Impact	High	**Category**	Data Validation
Exploitability	High	**Status**	Reported

CVSS 8.7 (CVSS:4.0/AV:A/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N)

Impact

An attacker within Bluetooth range can achieve arbitrary code execution on an ESP32 device running the BluFi reference code by exploiting the initial secure key exchange commands.

These commands are used to establish a secure channel at the beginning of the connection, so even if extra functionality were added to the BluFi application to require a user to authenticate before setting WiFi credentials, these commands would be vulnerable to unauthenticated attackers.

Description

At the beginning of the secure key exchange process, the BluFi reference code reads Diffie-Hellman parameters from the peer (a BluFi client application), including the generator g and modulus p used for calculating public keys. The parameters are read into the `blufi_sec->dhm` Diffie-Hellman context structure:

```
uint8_t *param = blufi_sec->dh_param;
memcpy(blufi_sec->dh_param, &data[1], blufi_sec->dh_param_len);
ret = mbedtls_dhm_read_params(&blufi_sec->dhm, &param, &param[blufi_sec->dh_param_len]);
```

The ESP32 then calculates a public key to match the length of the modulus `p` provided by the peer. Calling `mbedtls_dhm_get_len(&blufi_sec->dhm)` returns the size of the `P` multi-precision integer in the Diffie-Hellman parameters. (Older versions of the BluFi reference application, such as the one in ESP-IDF v4.4, do the equivalent by calling `mbedtls_mpi_size( &blufi_sec->dhm.P )`.)

```
const int dhm_len = mbedtls_dhm_get_len(&blufi_sec->dhm);
ret = mbedtls_dhm_make_public(&blufi_sec->dhm, dhm_len, blufi_sec->self_public_key, dhm_len,
↪ myrand, NULL);
```

However, the BluFi application hard-codes the size of its public key buffer to be 128 bytes (1024 bits), which may not match the size of the parameters received from the peer:

```
struct blufi_security {
#define DH_SELF_PUB_KEY_LEN     128
#define DH_SELF_PUB_KEY_BIT_LEN (DH_SELF_PUB_KEY_LEN * 8)
    uint8_t  self_public_key[DH_SELF_PUB_KEY_LEN];
#define SHARE_KEY_LEN           128
#define SHARE_KEY_BIT_LEN       (SHARE_KEY_LEN * 8)
    uint8_t  share_key[SHARE_KEY_LEN];
    size_t   share_len;
#define PSK_LEN                 16
    uint8_t  psk[PSK_LEN];
    uint8_t  *dh_param;
```

```
    int      dh_param_len;
    uint8_t  iv[16];
    mbedtls_dhm_context dhm;
    mbedtls_aes_context aes;
};
```

If the peer specifies a modulus larger than 1024 bits, the ESP32 will overflow its public key buffer when calculating a public key.

Since the attacker controls the Diffie-Hellman generator and modulus parameters, they can force the other party to calculate a specific public key no matter what random private key exponent they choose. This allows the attacker to send an arbitrary buffer to overflow the `self_public_key` buffer with.

The largest possible size an MbedTLS multi-precision integer can be is 1024 bytes, or 8192 bits (`MBEDTLS_MPI_MAX_SIZE`). Range checks and modular exponentiation functions used by `mbedtls_dhm_make_public` require that the modulus is an odd number, and that the generated public key is in the range $[2, p-2]$.

To calculate specific bytes into `self_public_key`, the attacker creates an arbitrary buffer and then converts it into an integer. The last few bits must be adjusted so that it is an odd number and has a remainder of 1 when divided by 3. This is used as the generator parameter g. The modulus is then $3g$. No matter what random positive non-zero exponent x the device chooses, the result of the public key calculation $g^x \mod 3g$ is g.

By overwriting the `dh_param` pointer and `dh_param_len` size value within the same `blufi_security` data structure, the attacker can then send another `SEC_TYPE_DH_PARAM_DATA` command to write to an arbitrary location in memory with the call to `memcpy(blufi_sec->dh_param, &data[1], blufi_sec->dh_param_len)`. This creates a write-what-where primitive that the attacker can use to take control of the device's instruction pointer.

Recommendation

The call to `mbedtls_dhm_make_public` should not use an output length `olen` larger than the size of the destination buffer. As the output length should also be at least equal to `ctx->len`, consider increasing the size of `self_public_key` to `MBEDTLS_MPI_MAX_SIZE`, or dynamically allocating a sufficiently sized buffer for the public key.

There are further cryptographic concerns for using finite field Diffie-Hellman with a key size limited to 1024 bits, as well as allowing the peer to specify the generator and modulus parameters instead of using safe pre-determined values. These concerns are described in finding "Key Exchange Algorithm Doesn't Meet Best Practices".

Location

`esp-idf/examples/bluetooth/blufi/main/blufi_security.c`

High

Unauthenticated Key Exchange

Overall Risk	High	**Finding ID**	NCC-BluFi-Ref-YL7
Impact	High	**Category**	Cryptography
Exploitability	Medium	**Status**	Reported

CVSS 7.4 (CVSS:4.0/AV:A/AC:H/AT:P/PR:N/UI:P/VC:H/VI:H/VA:N/SC:N/SI:N/SA:N)

Impact

An attacker who can perform a Man-in-the-Middle on the BluFi BLE connection would be able to decrypt and/or alter all messages sent across the connection, which could reveal secret information such as WiFi network credentials.

Description

The BluFi reference application secure key exchange process is unauthenticated, making it vulnerable to an active Man-in-the-Middle attack on the connection between a client and ESP32 device.

Without authenticated pairing, the BluFi client has no way to check if it has connected to the real target device. If they connect to an attacker's BLE device instead, the attacker can establish a separate connection with the real target device and pass messages across the link, with the ability to decrypt and tamper with anything sent between the client and ESP32 device.

Recommendation

The lack of authentication in the key exchange process is difficult to solve without some kind of public key infrastructure or ability to display a code on the ESP32-based device (similar to Bluetooth passkey or numeric comparison). These solutions would be specific to each application built upon the BluFi reference app.

The lack of authenticated pairing should at least be called out with a warning for developers using the BluFi reference application.

Location

`esp-idf/examples/bluetooth/blufi/main/blufi_security.c`

Medium

Out-of-Bounds Reads in Diffie-Hellman Key Negotiation

Overall Risk	Medium	**Finding ID**	NCC-BluFi-Ref-YQG
Impact	Low	**Category**	Data Exposure
Exploitability	High	**Status**	Reported

CVSS 5.3 (CVSS:4.0/AV:A/AC:L/AT:N/PR:N/UI:N/VC:L/VI:N/VA:N/SC:N/SI:N/SA:N)

Impact

An attacker within Bluetooth range can cause the ESP32 device to read out-of-bounds data into Diffie-Hellman parameter setup structures, leaking very limited information about the content of the out-of-bounds data.

Description

The `blufi_dh_negotiate_data_handler` for handling DH key negotiation messages fails to check the length of the received data buffer before reading from it.

For example, `blufi_dh_negotiate_data_handler` initially reads a message type byte without checking that `len` is greater than zero:

```
uint8_t type = data[0];
```

For `SEC_TYPE_DH_PARAM_LEN` messages, the handler accesses offsets 1 and 2 into `data` without checking its length is at least 3:

```
case SEC_TYPE_DH_PARAM_LEN:
    blufi_sec->dh_param_len = ((data[1]<<8)|data[2]);
```

The most significant OOB read is in the handler for `SEC_TYPE_DH_PARAM_DATA`, where it can read up to 65k bytes of out-of-bounds data based on the attacker-controlled `dh_param_len` value. The size of the `data` buffer isn't checked against `blufi_sec->dh_param_len` before doing a `memcpy` operation:

```
memcpy(blufi_sec->dh_param, &data[1], blufi_sec->dh_param_len);
```

An attacker could trigger this by sending an `SEC_TYPE_DH_PARAM_LEN` message that specifies a length of 0xFFFF, and then send a `SEC_TYPE_DH_PARAM_DATA` message only containing one byte (the `type` value).

Recommendation

Use the `len` argument to validate the size of the `data` buffer before reading from it. Ensure that the size and offset of the read remains within the bounds of the buffer.

Location

`esp-idf/examples/bluetooth/blufi/main/blufi_security.c`

Medium

Key Exchange Algorithm Doesn't Meet Best Practices

Overall Risk	Medium	**Finding ID**	NCC-BluFi-Ref-2BB
Impact	High	**Category**	Cryptography
Exploitability	Low	**Status**	Reported

CVSS 5.9 (CVSS:4.0/AV:A/AC:H/AT:N/PR:N/UI:P/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N)

Impact

The secure key negotiation scheme used in the BluFi reference application doesn't follow modern best practices for algorithm or parameter choice.

A compromised secure key exchange process could allow an attacker to eavesdrop on messages between a client and ESP32 device to leak sensitive data such as WiFi network credentials.

Description

There are several cryptographic concerns with the way secure key exchange is performed in the BluFi reference application:

- Using finite field Diffie-Hellman with a key size limited to 1024 bits
- Allowing the peer to specify the generator and modulus parameters instead of using safe pre-determined values
- Using finite field Diffie-Hellman (DH) instead of Elliptic-curve Diffie–Hellman (ECDH)

Limiting the Diffie-Hellman key to only 1024 bits does not meet modern best practices for cryptography. Attacks against pre-computed 1024 bit Diffie-Hellman parameters have been considered feasible for nearly a decade. While the ESP32 reference application allows arbitrary parameters to be supplied by the peer, the reference client applications for Android and iOS do use pre-computed parameters:

- https://github.com/EspressifApp/EspBlufiForAndroid/blob/1d707064e0930e3d4a3dc303e9c81971e1304dc0/lib-blufi/src/main/java/blufi/espressif/BlufiClientImpl.java#L55
- https://github.com/EspressifApp/EspBlufiForiOS/blob/87456f811bdc25e8e0aa2e307eee164a304f10e3/BlufiLibrary/Security/BlufiSecurity.m#L33

Using Elliptic-Curve Diffie-Hellman key exchange instead of finite field Diffie-Hellman would avoid this weakness. See https://weakdh.org/.

Diffie-Hellman parameter validation in the mbedTLS library is not rigorous enough to prevent the client from choosing insecure or intentionally malicious parameters; it primarily validates public keys generated with generator and modulus parameters that are assumed to be safe.

Recommendation

Consider using ECDH instead of DH.

Safe base parameters should be chosen ahead of time and hard-coded into the BluFi ESP32 and client applications, so that only the public keys need to be exchanged between the ESP32 and client applications.

For example, for finite field Diffie-Hellman, generate a safe 2048-bit (or larger) group ahead of time. For Elliptic-Curve Diffie-Hellman, use Curve25519.

Location

`esp-idf/examples/bluetooth/blufi/main/blufi_security.c`

Low

Lack of Cryptographic Authentication for Secure Messages

Overall Risk	Low	**Finding ID**	NCC-BluFi-Ref-A2M
Impact	Low	**Category**	Cryptography
Exploitability	Medium	**Status**	Reported

CVSS 2.1 (CVSS:4.0/AV:A/AC:L/AT:P/PR:N/UI:P/VC:N/VI:L/VA:N/SC:N/SI:N/SA:N)

Impact

Encrypted messages sent over a BluFi BLE connection do not use cryptographic authentication to ensure that the content of the messages has not been tampered with.

Description

The reference BluFi application uses AES in CFB mode for encryption. This cipher mode is partially based on XOR operations, making it “malleable” - flipping a bit value in the ciphertext directly results in a corresponding flip in the corresponding plaintext.

Currently, lack of authentication on the secure key exchange process is a more compelling MitM attack as it would give the attacker the ability to fully decrypt and encrypt messages using a known secret key.

Recommendation

Use a standard authenticated encryption mode such as AES-GCM. Ensure that the cipher parameters are chosen securely. For example, for AES-GCM make sure to generate a securely random nonce value for the initialization vector (IV) of each encrypted stream.

Location

`esp-idf/examples/bluetooth/blufi/main/blufi_security.c`

4 Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Rating	Description
Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

Rating	Description
High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

Rating	Description
High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Rating	Description
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Category Name	Description
Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.

5 Contact Info

The team from NCC Group has the following primary member:

- James Chambers – Researcher
  james.chambers@nccgroup.com