Hardware Wallet Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.12.28, the SlowMist security team received the OneKey team's security audit application for OneKey Classic 1s, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "black box lead, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for hardware wallet includes two steps:

- The codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The hardware wallets are manually analyzed to look for any potential issues.

The following is a list of security audit items considered during an audit:

- Firmware Security
  - Firmware storage security audit
  - Firmware upgrade security audit
  - Firmware integrity security audit
  - Firmware decompilation security audit
  - Firmware configuration security audit
- Storage Security
  - Data storage security audit
- Exception Handling Security
  - Error handling security audit
  - Exception logs security audit
- Permission Security
  - App permissions detection
- Communication Security
  - Communication encryption security audit
- Device Interface Security
  - Interface security audit
- Business Security
  - Business logic security audit

- Authentication Security
  - Device-Based authentication security audit
- Transfer Security
  - Signature security audit
  - Deposit/Transfer security audit
  - Transaction broadcast security audit
- Secret key Security
  - Secret key generation security audit
  - Secret key storage security audit
  - Secret key usage security audit
  - Secret key backup security audit
  - Secret key destruction security audit
  - Insecure entropy source audit
  - Cryptography security audit
- Components Security
  - Third-party components security audit
- User Interaction Security
  - WYSIWYS
  - Password complexity requirements

3 Project Overview

3.1 Project Introduction

The main electronic components of OneKey Classic 1s are GD32F470VKH6 (MCU), THD89 (SE), and nRF52832 (Bluetooth). The task is to conduct a security audit of the firmware of OneKey Classic 1s.

Audit Version

https://github.com/OneKeyHQ/firmware-classic1s/tree/slowmist_audit commit: 17da65e3c234eddf4a78997c5898e8b03b0c2dd6

https://github.com/OneKeyHQ/THD89/tree/slowmist_audit

commit: 66e7fee2d00ea22616a9a488266c472abeafc6d3

Fixed Version

https://github.com/OneKeyHQ/firmware-classic1s

commit: 46b864f8d22713a4b27bf7d04dadc2d0430110c2

https://github.com/OneKeyHQ/THD89

commit: 26abec129fa20e8ecdf5b3f0d754eb63ff0f7ca0

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Allow downgrade of firmware	Firmware upgrade security audit	Suggestion	Acknowledged
N2	Firmware upgrade without identity verification	Firmware upgrade security audit	Suggestion	Acknowledged
N3	Missing security self-check functionality configuration	Firmware configuration security audit	Suggestion	Acknowledged
N4	SPC downgrade risk	Firmware configuration security audit	Low	Acknowledged
N5	The AES IV is transmitted without encryption	Communication encryption security audit	Medium	Fixed
N6	Sync session sniffing	Communication encryption security audit	Suggestion	Fixed
N7	Bluetooth and MCU communication is not authenticated and encrypted	Communication encryption security audit	Low	Acknowledged

NO	Title	Category	Level	Status
N8	The communication between the app and hardware wallet is not encrypted	Communication encryption security audit	Low	Acknowledged
N9	Exposed SWD interface	Device Interface Security	Medium	Fixed
N10	Password complexity issue	Device-Based authentication security audit	Suggestion	Acknowledged
N11	Optimization of verifyMnemonic	Secret key generation security audit	Suggestion	Acknowledged
N12	cached_keys pre-set issue	Secret key destruction security audit	Suggestion	Acknowledged
N13	Missing check for curve order	Cryptography security audit	Suggestion	Acknowledged
N14	Confirmation key and page up key sharing issue	User interaction security	Suggestion	Acknowledged
N15	Signature request smuggling	User interaction security	Low	Acknowledged

3.3 Vulnerability Summary

[N1] [Suggestion] Allow downgrade of firmware

Category: Firmware upgrade security audit

Content

Firmware upgrades allow users to downgrade to older firmware versions. If an older firmware version contains security vulnerabilities, an attacker could exploit these vulnerabilities by updating the device to the older version.

Solution

It is recommended to add a restriction to the firmware upgrade process that prevents users from downgrading to a firmware version that is older than the current version. This will help to prevent attackers from using older or vulnerable firmware to attack hardware wallets.

Status

Acknowledged

[N2] [Suggestion] Firmware upgrade without identity verification

Category: Firmware upgrade security audit

Content

Firmware upgrades do not require identity verification. Users can upgrade the firmware by entering bootloader mode. Password verification is not required. In the absence of identity verification, the device will not be reset after the upgrade.

Solution

It is recommended to reset the wallet after a firmware upgrade without password verification.

Status

Acknowledged

[N3] [Suggestion] Missing security self-check functionality configuration

Category: Firmware configuration security audit

Content

The lack of a security self-check function is a security risk. It is important to provide this type of function so that users can verify the security of their devices and firmware themselves. This helps to ensure that the device has not been maliciously modified and that the firmware is from an official source.

Solution

It is recommended to add a feature that allows users to self-check the security of their devices and firmware.

Status

Acknowledged

[N4] [Low] SPC downgrade risk

Category: Firmware configuration security audit

Content

The MCU used is GD32F470VKH6. GD32F470VKH6 provides SPC to protect the flash memory. The following is the description of SPC in the GD32F4xx_User_Manual_EN_Rev2 document.

No protection: when setting SPC byte to 0xAA, no protection performed. The main flash and option byte block are accessible by all operations.

Protection level low: when setting SPC byte to any value except 0xAA or 0xCC, protection level low performed. The main flash can only be accessed by user code. In debug mode, boot from SRAM or boot from boot loader mode, all operations to main flash is forbidden. If a read operation is executed to main flash in debug mode, boot from SRAM or boot from boot loader mode, a bus error will be generated. If a program/erase operation is executed to main flash in debug mode, boot from SRAM or boot from boot loader mode, the WPERR bit in FMC_STAT register will be set. At protection level low, option byte block is accessible by all operations. If program back to no protection level by setting SPC byte to 0xAA, a mass erase for main flash will be performed.

Protection level high: when setting SPC byte to 0xCC, protection level high performed. When this level is programmed, debug mode, boot from SRAM or boot from boot loader mode are disabled. The main flash block is accessible by all operations from user code. The SPC byte cannot be reprogrammed. So, if protection level high is programmed, it cannot move back to protection level low or no protection level.

Please refer to:

https://www.gigadevice.com.cn/Public/Uploads/uploadfile/files/20230228/GD32F4xx_User_Manual_EN_Rev2.8.pdf

The woot17-paper-obermaier, 3.2 Security Downgrade paper mentions an attack called RDP Level Downgrade that can downgrade RDP Level2 to RDP Level1.

Please refer to:

https://www.usenix.org/system/files/conference/woot17/woot17-paper-obermaier.pdf

One of the key points is that the RDP value is defined by the microcontroller configuration bits known as Option Bytes. Any value except 0xAA and 0xCC Option Byte values represent RDP Level1. Therefore, a glitch attack can be used to downgrade RDP Level2 to RDP Level1.

We found that GD32F470VKH6 has the same design flaw, which could allow downgrade from Protection level high to Protection level low.

Due to the use of THD89 Secure Element in onekey classic1s, the mnemonic or seed is only stored in the MCU when generating or creating a wallet. The mnemonic is calculated in THD89 Secure Element when signing. Therefore, we have not found a way to obtain the mnemonic at this time to exploit the SPC downgrade risk.

Solution

It is recommended to replace other MCU chips that are not affected by SPC downgrade risk.

Status

Acknowledged

[N5] [Medium] The AES IV is transmitted without encryption

Category: Communication encryption security audit

Content

In the communication process, the initialization vector (IV) is generated randomly. The AES key is negotiated and synchronized through the se_sync_session_key function. During synchronization, the default key is used for AES-ECB encryption transmission. The transmitted content is a new random number. The random number obtained by se_get_rand is not encrypted during transmission. Then, the random numbers from SE and MCU are merged and encrypted by the default AES key. Then, the r1 and r2 values are XORed and merged with the default AES key. Finally, the hash of the merged data is used as the SE session key. The encrypted data is then transmitted.

The IV is generated using the se_get_rand function to obtain a random. The transmission between se_get_rand and SE is not encrypted.

Code location: firmware-classic1s/legacy/firmware/se_chip.c

```
secbool se_transmit_mac(uint8_t ins, uint8_t p1, uint8_t p2, uint8_t *data,
                        uint16_t data_len, uint8_t *recv, uint16_t *recv_len) {
  uint8_t mac[4], iv_random[16];
  uint16_t pad_len;
  APDU_CLA = 0x84;
  APDU_INS = ins;
  APDU_P1 = p1;
  APDU_P2 = p2;
  APDU_P3 = 0x00;

  if (!se_get_rand(iv_random, 16)) {
    return secfalse;
  }
  ...
```

Code location: firmware-classic1s/legacy/firmware/se_chip.c

```
secbool se_sync_session_key(void) {
  uint8_t r1[16], r2[16], r3[32];
  uint8_t default_key[16] = {0xff};

  memset(default_key, 0xff, 16);
  uint8_t data_buf[64], hash_buf[32];
  uint8_t sync_cmd[5 + 48] = {0x00, 0xfa, 0x00, 0x00, 0x30};
  uint16_t recv_len = 0xff;
  aes_encrypt_ctx en_ctxe;
  aes_decrypt_ctx de_ctxe;
  memzero(data_buf, sizeof(data_buf));

  flash_otp_read(FLASH_OTP_BLOCK_THD89_SESSION_KEY, 0, default_key,
                 sizeof(default_key));

  // get random from se
  se_get_rand(r1, 16);
  // get random itself
  random_buffer(r2, 16);
  // organization data1
  memcpy(r3, r1, sizeof(r1));
  memcpy(r3 + sizeof(r1), r2, sizeof(r2));
  aes_init();
  aes_encrypt_key128(default_key, &en_ctxe);
  aes_ecb_encrypt(r3, data_buf, sizeof(r1) + sizeof(r2), &en_ctxe);

  // cal tmp sessionkey with x hash256
  memzero(r3, sizeof(r3));
  xor_cal(r1, r2, sizeof(r1), r3);
  memcpy(r3 + 16, default_key, 16);
  sha256_Raw(r3, 32, hash_buf);
  // use session key organization data2
  memcpy(se_session_key, hash_buf, 16);
  aes_encrypt_key128(se_session_key, &en_ctxe);
  aes_ecb_encrypt(r1, data_buf + 32, sizeof(r1), &en_ctxe);
  // send data1 + data2 to se and recv returned result
  memcpy(sync_cmd + 5, data_buf, 48);
  if (!thd89_transmit(sync_cmd, sizeof(sync_cmd), data_buf, &recv_len)) {
    memset(se_session_key, 0x00, SESSION_KEYLEN);
    return secfalse;
  }

  // handle the returned data
  aes_decrypt_key128(se_session_key, &de_ctxe);
  aes_ecb_decrypt(data_buf, r3, recv_len, &de_ctxe);
  if (memcmp(r2, r3, sizeof(r2)) != 0) {
    memset(se_session_key, 0x00, SESSION_KEYLEN);
    return secfalse;
  }

  se_session_init = true;

  return sectrue;
}
```

Solution

It is recommended that the IV be transmitted using encryption.

Status

Fixed

[N6] [Suggestion] Sync session sniffing

Category: Communication encryption security audit

Content

The default session_key is transmitted in cleartext. Additionally, the default session_key is stored in the OTP.

Therefore, it is considered insecure before se_sync_session_key is used to generate a new key.

After the device is reset, the default session_key in the OTP will be reused. In extreme cases, if the OTP is obtained, it will affect se_sync_session_key, resulting in the leakage of communication encryption keys.

Code location: firmware-classic1s/legacy/firmware/se_chip.c

```
secbool se_set_session_key(const uint8_t *session_key) {
  uint8_t cmd[32] = {0x00, 0xF6, 0x00, 0x02, 0x10};
  uint16_t resp_len = 0;
  memcpy(cmd + 5, session_key, SESSION_KEYLEN);
  return thd89_transmit(cmd, 21, NULL, &resp_len);
}
```

Solution

It is recommended to authenticate the communicating devices when generating a new session_key when creating a wallet or importing a wallet. This ensures that the components are authenticated, and it can be achieved using asymmetric encryption.

Status

Fixed

[N7] [Low] Bluetooth and MCU communication is not authenticated and encrypted

Category: Communication encryption security audit

Content

The communication between Bluetooth and MCU is not encrypted, which could allow an attacker to insert a malicious component to perform a man-in-the-middle attack. However, the Bluetooth firmware is not within the scope of this audit.

Solution

It is recommended to authenticate Bluetooth components, and to encrypt the communication between Bluetooth and MCU.

Status

Acknowledged

[N8] [Low] The communication between the app and hardware wallet is not encrypted

Category: Communication encryption security audit

Content

When a desktop app communicates with a hardware wallet, it is not authenticated and the communication is not encrypted. This means that the user cannot distinguish which app is sending a signing request to the hardware wallet. Therefore, a malicious app can send a signing request to the hardware wallet when the user is using it, or perform a man-in-the-middle attack to trick the user into signing a malicious request, which could then be used to steal the user's cryptocurrency assets.

Solution

It is recommended that apps should be authenticated when interacting with hardware wallets. Unauthenticated apps should not be allowed to communicate with hardware wallets. Additionally, apps and hardware wallets should use encrypted communication to avoid man-in-the-middle attacks.

Status

Acknowledged

[N9] [Medium] Exposed SWD interface

Category: Device Interface Security

Content

The Bluetooth uses SWD for firmware update, and the SWD interface is exposed. However, the Bluetooth hardware is not authenticated or verified to ensure that it is genuine, and the firmware is not checked to ensure that it is released by the official. Therefore, there is a risk of the Bluetooth hardware being replaced or the firmware being modified. This could occur in a supply chain attack scenario, where an attacker could replace the Bluetooth firmware or hardware to replace the signature data, and then steal cryptocurrency.

Code location: firmware-classic1s/legacy/bootloader/usb.c

```
#if BLE_SWD_UPDATE
      update_status = bUBLE_UpdateBleFirmware(
          fw_len, FLASH_BLE_SE_ADDR_START + FLASH_FWHEADER_LEN, ERASE_ALL);

#else
```

Solution

It is recommended to prohibit the use of the SWD interface in production environments. Additionally, the firmware of the Bluetooth module should be verified to ensure that it is released by the official. The Bluetooth hardware (nRF52832) should be authenticated to ensure that it has not been replaced.

Status

Fixed

[N10] [Suggestion] Password complexity issue

Category: Device-Based authentication security audit

Content

The password supports 4-9 digit numeric passwords, which have a single character type. The password complexity can be improved by increasing the character type, such as requiring that the password contain letters and numbers, and the password length should be at least 8 digits.

Solution

It is recommended to increase the character type of passwords to include letters, and to require passwords to be at least 8 characters long.

Status

Acknowledged

[N11] [Suggestion] Optimization of verifyMnemonic

Category: Secret key generation security audit

Content

In the wallet, the seed random comes from the secure element (SE). However, the process of generating the seed and mnemonic phrase is done in the microcontroller unit (MCU). After creation, the mnemonic phrase is encrypted and transmitted to the SE, where it is encrypted and stored. The mnemonic phrase is verified in the SE. If the mnemonic phrase is generated in the MCU, it can also be verified in the MCU, which can reduce the number of times the mnemonic phrase needs to be transmitted.

Code location: THD89/src/mid/config.c

```
bool config_verifyMnemonic(const char *mnemonic) {
  if (!storage_is_initialized()) {
    return false;
  }
  char dev_mnemonic[MAX_MNEMONIC_LEN + 1] = {0};
  if (!storage_get_mnemonics(dev_mnemonic, false)) {
    return false;
  }
  if (strlen(dev_mnemonic) != strlen(mnemonic)) {
    return false;
  }
  if (memcmp(dev_mnemonic, mnemonic, strlen(dev_mnemonic)) == 0) {
    return true;
  }
  return false;
}
```

Solution

It is recommended that the verifyMnemonic process be implemented in the MCU after creating a wallet, which can reduce the number of times the mnemonic phrase needs to be transmitted.

Status

Acknowledged

[N12] [Suggestion] cached_keys pre-set issue

Category: Secret key destruction security audit

Content

logic_storage_init clears the page data, init_wiped_storage regenerates the encryption and decryption element cached_keys, and sets the pin to empty. As an encryption and decryption element, cached_keys should be regenerated when creating or importing a wallet, rather than being pre-generated after reset. After reset, the original cached_keys can be cleared.

The cached_keys pre-set issue causes the cached_keys to be set before the user receives the wallet device, rather than being randomly generated by the user.

Code location: THD89/src/mid/storage.c

```
void storage_wipe(void) {
  memzero(cached_keys, sizeof(cached_keys));
  logic_storage_init(true);
  init_wiped_storage();
  device_wiped = true;
}
```

Code location: THD89/src/mid/logic_storage.c

```
void logic_storage_init(bool wipe) {
  ...
      phy_page_erase(logic_repeat_info[i].phy_index[unvalid_index]);
      p_logic_page = (phy_page_t *)phy_index_to_address(
          logic_repeat_info[i].phy_index[valid_index]);
      logic_page_map[p_logic_page->head.logic_index] =
          logic_repeat_info[i].phy_index[valid_index];
  ...
}
```

Code location: THD89/src/mid/storage.c

```
static void init_wiped_storage(void) {
  random_gen(cached_keys, sizeof(cached_keys));
  set_pin(PIN_EMPTY, PIN_EMPTY_LEN, NULL);
  unlocked = true;
}
```

Solution

It is recommended to regenerate cached_keys only when the user imports or creates a wallet.

Status

Acknowledged

[N13] [Suggestion] Missing check for curve order

Category: Cryptography security audit

Content

The init_rfc6979 function lacks a check for the order of the elliptic curve. The order is typically considered secure if it is greater than or equal to 256 bits.

Code location: THD89/src/crypto/rfc6979.c

```
void init_rfc6979(const uint8_t *priv_key, const uint8_t *hash,
                  const ECC_Domain_t *curve, rfc6979_state *state) {
  if (curve) {
    uint8_t hash_reduced[32] = {0};
    bignum_sub_cond((uint8_t *)hash, curve->pbnN->p, hash_reduced);
    hmac_drbg_init(state, priv_key, 32, hash_reduced, 32);
    memzero(hash_reduced, sizeof(hash_reduced));
  } else {
    hmac_drbg_init(state, priv_key, 32, hash, 32);
  }
}
```

Solution

It is recommended to add a check for the order in the init_rfc6979 function to ensure that the order is greater than or equal to 256 bits.

Status

Acknowledged

[N14] [Suggestion] Confirmation key and page up key sharing issue

Category: User interaction security

Content

When browsing transaction details, the confirmation key and the page up key are shared. The classic1s has four buttons, and the page up key should be in the middle two. Now that the confirmation key and the page up key are shared, the confirmation key is used for both transaction confirmation and scroll up, it is often accidentally pressed to confirm the transaction when using the confirmation key to scroll up, thinking that there are still transaction details.

Solution

It is recommended to separate the confirmation key's function. The confirmation key should only be used for transaction confirmation, and the middle two buttons should be used for browsing transaction details.

Status

Acknowledged

[N15] [Low] Signature request smuggling

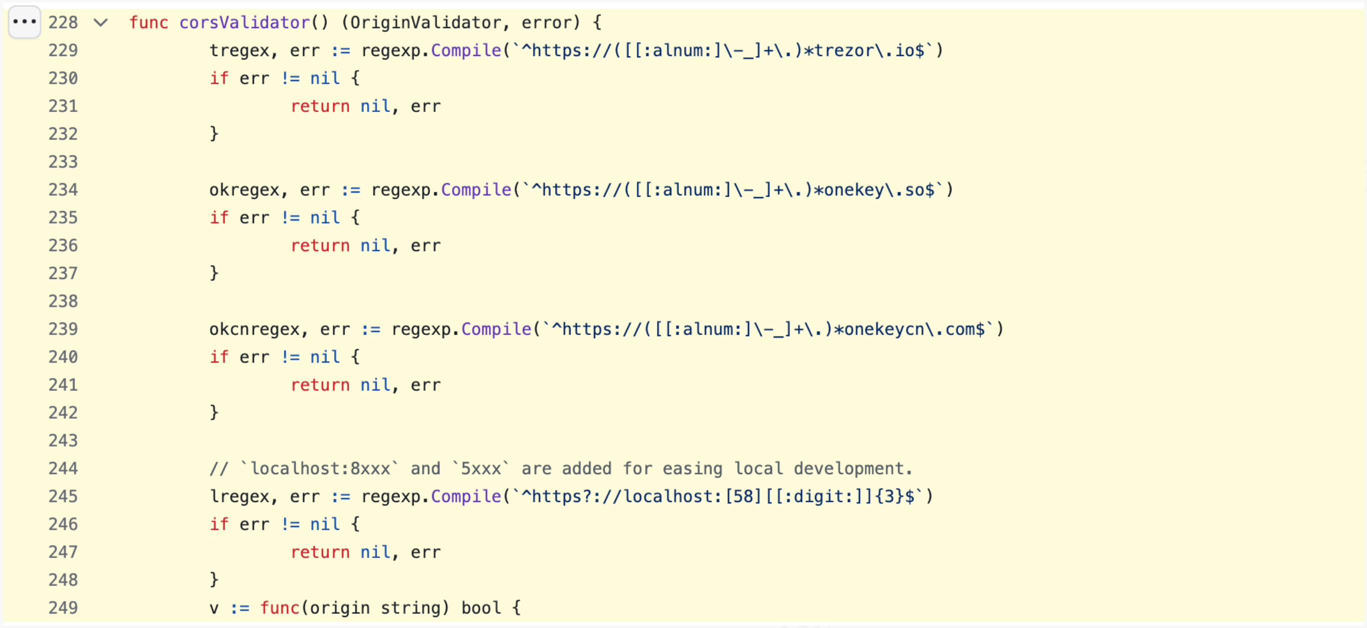
Category: User interaction security

Content

The signature of a trusted dapp request can be replaced by the signature of a malicious dapp request. In specific, the hardware wallet displays the signature content of the trusted dapp, then the malicious dapp will close the trusted request after signing and reopen the new signature content. In this scenario, if the user does not observe that the request has been replaced, they may think that the signature data is from a trusted dapp. In the case of consecutive requests, the signature request from the trusted dapp may not even appear.

OneKey Bridge is used to forward requests from jssdk.onekey.so. OneKey Bridge uses CORS to restrict communication to the following domains on http://localhost:21320/. However, jssdk.onekey.so allows any dapp to access and use it. jssdk.onekey.so does not check or pass the origin of dapp messages. Therefore, the hardware wallet cannot display the source of the signature request. The hardware wallet also allows unauthorized dapps to communicate with it, which leads to this issue.

Code location: https://github.com/OneKeyHQ/onekey-bridge/blob/onekey/server/api/api.go#L228



```go
func corsValidator() (OriginValidator, error) {
	tregex, err := regexp.Compile(`^https://([[:alnum:]\-_]+\.)*trezor\.io$`)
	if err != nil {
		return nil, err
	}

	okregex, err := regexp.Compile(`^https://([[:alnum:]\-_]+\.)*onekey\.so$`)
	if err != nil {
		return nil, err
	}

	okcnregex, err := regexp.Compile(`^https://([[:alnum:]\-_]+\.)*onekeycn\.com$`)
	if err != nil {
		return nil, err
	}

	// `localhost:8xxx` and `5xxx` are added for easing local development.
	lregex, err := regexp.Compile(`^https?://localhost:[58][[:digit:]]{3}$`)
	if err != nil {
		return nil, err
	}
	v := func(origin string) bool {
```

Solution

It is recommended to get the origin of the post message in jssdk.onekey.so and then pass it to the bridge. The bridge can pass it to the hardware wallet for authentication. Only dapps that have been authenticated can communicate with the hardware wallet.

Status

Acknowledged

4 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002401150001	SlowMist Security Team	2023.12.28 - 2024.01.15	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risk, 4 low risk vulnerabilities and 9 suggestions.

5 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.