Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Safeheron WASM MPC & Snap 09.2023

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi, A. Ahelleya

Index

Introduction
Scope
Identified Vulnerabilities
SFH-01-001 WP2: Key generation overwrites previous MPC account (Medium) FIXED
SFH-01-002 WP2: Request signatures require no user-modal interaction (Info)
SFH-01-003 WP2: Private key logged in the console (Medium) FIXED
SFH-01-004 WP2: Lack of payload validation in Snaps (Low) FIXED
SFH-01-005 WP2: Backup flow exports private keys as plaintext (Medium) FIXED
SFH-01-006 WP2: Suboptimal state follows request flow (Low) FIXED
SFH-01-007 WP2: Use of deprecated MetaMask permissions in Snaps (Low) FIXED
SFH-01-008 WP2: localhost string included on the allow-list (Low) FIXED
Conclusions

Introduction

"More Than Just a Wallet, It's the Next-Generation Digital Asset Custody - Compared to the previous generation of custody products, Safeheron offers enhanced digital asset security without losing private key control. Customers retain 100% control over their private keys and asset ownership, enjoying institutional-grade security and efficient management."

From https://www.safeheron.com/en-US/mpc-wallet/

This report describes the results of a penetration test and source code audit targeting the code repositories and implementations of two Safeheron components, namely the Safeheron WASM MPC and the Safeheron Multi-MPC Snap.

The work was requested by Safeheron Pte. Ltd. in August 2023 and carried out by Cure53 in September 2023. The project began in CW37 and the testing was finalized in CW39.

To comment on the resources allocated to this project, registered as *SFH-01,* a total of eighteen days were invested to reach the expected coverage. Moreover, a team consisting of three senior testers was assigned to the examination's preparation, execution and finalization.

The work was split into two separate work packages (WPs):

- **WP1**: Black-box penetration testing & code auditing of Safeheron WASM MPC code and implementation
- **WP2**: White-box penetration testing & code auditing of Safeheron Multi-MPC Snap code and implementation

Cure53 was provided with sources, a list of key items in focus, as well as all further means of access required to complete the tests. Originally, the methodology chosen for both WPs was white-box. However, due to some complications during the setup and preparation phases, it was mutually agreed to change the methodology of WP1 into black-box.

All preparations were done in September 2023, namely CW36, so that the Cure53 testing team could have a smooth start. Communications during the test were done using a dedicated shared Slack channel, which was set up to connect the Safeheron and Cure53 teams. Involved personnel from both parties could join discussions on Slack.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

While the exchanges on Slack were helpful, it was notable that several questions had to be asked. This was due to the prepared source material, which led to several issues, mainly in the frames of WP1. Ultimately, this was the reason for the methodology change in WP1, as well as a stronger emphasis on WP2 targets. No further noteworthy roadblocks were encountered during the test.

Cure53 gave frequent status updates about the test and the related findings. Live-reporting was offered by Cure53 and was executed via the aforementioned Slack channel.

Very good coverage was reached over the WP1-WP2 scope items. All eight findings were classified to be security vulnerabilities, meaning that no general weaknesses could be observed. The overall number of findings can be seen as acceptable and can be interpreted as a positive indicator for the security of the Safeheron MPC's MetaMask Snap. This is further supported by the fact that no vulnerabilities of *High* or *Critical* severity were identified during this audit.

Yet, it should once again be mentioned that the WP1 targets, spanning the Safeheron WASM MPC code and implementation, were not fully testable. This was caused by the unanticipated restrictions within the provided sources.

The report will now shed more light on the scope and test setup as well as the available material for testing. After that, the report will list all vulnerabilities in chronological order of discovery. Each finding will be accompanied with a technical description, a PoC where possible as well as mitigation or fix advice.

The report will then close with a conclusion in which Cure53 will elaborate on the impressions and recommendations formulated on the basis of this test. Moreover, the testing team will issue a more general verdict about the perceived security posture of the scope that comprises the Safeheron WASM MPC, as well as the Safeheron Multi-MPC Snap.

Scope

- **Penetration testing & code auditing of Safeheron WASM MPC & Multi-MPC Snap code and implementations**
  - **WP1**: Black-box penetration testing & code auditing of Safeheron WASM MPC code and implementation
    - **Sources:**
      - https://github.com/Safeheron/mpc-wasm-sdk
    - **Audited Commit:**
      - *27ec4b6ee5ebb56401852df00417d9186aa9cae3*
    - *The resources given for WP1 do not allow for the fulfillment of the described scope. Therefore, the methodology was changed from white-box into black-box.*
  - **WP2**: White-box penetration testing & code auditing of the Safeheron Multi-MPC Snap code and implementation
    - **Sources:**
      - https://github.com/Safeheron/multi-mpc-snap-monorepo
    - **Audited Commit:**
      - *80c59cc48c0e7c003e80c5d30a6895aa6d5d2eba*
    - **Commit for Fix Verification**
      - *d37b291c1874f98f919142a60fee51ab29b07ce3*
  - **Key focus areas:**
    - Architectural design and infrastructure
    - Data storage and encryption mechanisms
    - Authentication and authorization protocols
    - Network security measures
    - Incident response and recovery procedures
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *SFH-01-001*) to facilitate any future follow-up correspondence.

SFH-01-001 WP2: Key generation overwrites previous MPC account *(Medium)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

Safeheron MPC's state management layer for the MetaMask Snap only stores one wallet/account at a time. If a key-shard already exists in the storage but a keygen flow is re-initiated at the Snap level, the user would not be warned that their existing key-shard will be overwritten. Consequently, the newly generated key will overwrite the state, resulting in key loss.

Affected file:
packages/snap/src/mpc-flow/KeyGenFlow.ts

In order to generate a key-shard, the Safeheron Snap calls methods in *KeyGenFlow.ts*. First, the shard is initiated using *keyGenApproval*:

```
async keyGenApproval(
walletName: string,
party: Party
): Promise<SnapRpcResponse<string>> {
    await requestConfirm(
    panel([
       heading('Confirm to create an MPC wallet?'),
       text(`Wallet Name: ${walletName}`),
    ])
    )

    this.keyGen = this.mpcInstance.KeyGen.getCoSigner()

    this.walletName = walletName
    this.sessionId = uuidV4()
    this.keyGen!.setLocalParty(party.party_id, party.index)

    return succeed(this.sessionId)
}
```

Then, the *createWalletSuccess* method stores the key-shard in the Snap's state manager, which synchronizes with MetaMask's encrypted storage[1]:

```
async createWalletSuccess(
    sessionId: string
  ): Promise<SnapRpcResponse<AccountItem>> {
    this.verifySession(sessionId)

    // TODO validate pubkey
    const backuped = false
    const address = ethers.utils.computeAddress(`0x${this.pubKey}`)

    const snapAccount: SnapAccount = {
      id: uuidV4(),
      name: this.walletName!,
      address,
      options: {},
      supportedMethods: SUPPORTED_METHODS,
      type: 'eip155:eoa',
      backuped: false,
      pubkey: this.pubKey!,
      signKey: this.signKey,
    }

    await this.stateManager.saveOrUpdateAccount(snapAccount)

    return succeed({
      walletName: this.walletName!,
      address,
      backuped,
    })
  }
```

It is recommended that users receive warnings whenever the local state is about to be overwritten. Moreover, they should be informed about the consequences of key-regeneration actions. Alternatively, the key generation flow may be disabled if an account already exists locally, up until the key-shard is removed from the local state via a user-initiated account reset flow or similar.

[1] https://docs.metamask.io/snaps/reference/rpc-api/#snap_managestate

SFH-01-002 WP2: Request signatures require no user-modal interaction *(Info)*

Safeheron's MPC MetaMask Snap allows for requests to be signed without user-interaction. This means that the Safeheron dApp can sign requests on behalf of the key-shard stored in the Snap without active user presence.

Affected files:
packages/snap/src/mpc-flow/SignerFlow.ts

Safeheron MPC's MetaMask Snap initiates and approves request signatures through the *signApproval* method in *SignerFlow.ts*:

```
async signApproval(
      method: KeyringAccount['supportedMethods'][number],
      params: Record<string, any>,
      requestId?: string
  ): Promise<SnapRpcResponse<string>> {
      const wallet = this.getWalletWithError()

      if (requestId) {
      const requestIdIsValid = this.stateManager.isValidRequest(requestId)
      if (!requestIdIsValid) {
      throw new Error('Invalid request id: ' + requestId)
      }
      this.metamaskRequestId = requestId
      }

      this.sessionId = uuidV4()
      this.requestOrigin = Boolean(requestId) ? 'metamask' : 'website'
      this.signMethod = method
      this.signParams = params
      if (isTransaction(this.signMethod)) {
      this.normalizedTx = normalizeTx(params as TransactionObject)
      }
      this.signKey = wallet.signKey

      this.signer = this.mpcInstance.Signer.getCoSigner()

      return succeed(this.sessionId)
  }
```

The above method is called without any request for user approval. As a result, the Safeheron dApp will be able to sign requests on behalf of the key-shard that is stored in the Snap without their permission. Additionally, any client-side vulnerabilities in the dApp will be able to force signing requests (including transactions) on the user's behalf, doing so without their permission.

It is recommended to seek active user approval before signature requests can be generated. Generally speaking, confirming user presence before any authenticating cryptographic operation occurs is an industry standard. This can be seen, for example, in Yubikeys, where users are required to touch the Yubikey to allow it to authenticate, despite it being plugged into the computer and, in theory, programmable to dispense authenticating functions at any time.

SFH-01-003 WP2: Private key logged in the console *(Medium)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

Safeheron MPC's MetaMask Snap logs the user's private signing key directly in the web renderer's console. Given the extremely high sensitivity of the key material, this should be stopped.

Affected file:
packages/snap/src/StateManager.ts

In *StateManager.ts*'s *loadState* method, the user's private signing key is logged into the console:

```
console.debug('State Manager init: ', state.account, state.requests)
```

The property *state.account* is of type *SnapAccount*. This means it stores private signing keys:

```
export type SnapAccount = KeyringAccount & {
    signKey: string
    backuped: boolean
    pubkey: string
```

It is recommended to case logging of long-term private keys in consoles. Ideally, they should sit either in the program's namespacing or memory, such that they can never be read or accessed directly anyway. In that sense, they should exclusively be usable indirectly via wrapper functions that never expose or bring the key itself to the surface. Using the Snap’s persistent storage (see SFH-01-005) should be considered in this context.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

SFH-01-004 WP2: Lack of payload validation in Snaps *(Low)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

The Safeheron MPC MetaMask Snap uses the *superstruct npm* library[2] to validate the request body at runtime. However, it was observed that it failed to validate the type of request payloads, as well as their lengths at runtime, in certain areas.

Affected files:

- *packages/snap/src/rpc/internalMPCHander.ts*
- *packages/snap/src/mpc-flow/RecoveryFlow.ts*
- *packages/snap/src/mpc-flow/KeyGenFlow.ts*

In *internalMPCHandler.ts*, the *mpc_signApproval* method's request parameters have three properties with no input validation of any kind, namely *method*, *params* and *requestId*. To be more specific:

- *method* is asserted to be of type *"personal_sign"* | *"eth_sign"* | *"eth_signTransaction"* | *"eth_sendTransaction"* | *"eth_signTypedData"* | *"eth_signTypedData_v1"* | *"eth_signTypedData_v3"* | *"eth_signTypedData_v4"* without being validated at runtime.
- *params* is type-asserted to be of type *Record <string, unknown>*. This value is not type-validated at runtime; furthermore, at compile/transpile time, it is asserted to have properties that it may not have. Specifically, in *signApproval*, *params* is asserted to be of *TransactionObject* type.
- In the recovery flow's *recoverPrepare* method, and in the key generation flow's *keyGenApproval* method, the lengths of *walletName* are not validated.

It is recommended to validate *method* at runtime using *superstruct*. This could otherwise have harmful implications when chained with a client-side vulnerability. Moreover, it should be acknowledged that the signing flow does not require user approval. Thus, an attacker would be able to call arbitrary methods on the user's behalf without their knowledge. For *params*, set a narrower *TransactionObject* instead of *Record <string, unknown>* and also validate it at runtime using *superstruct*.

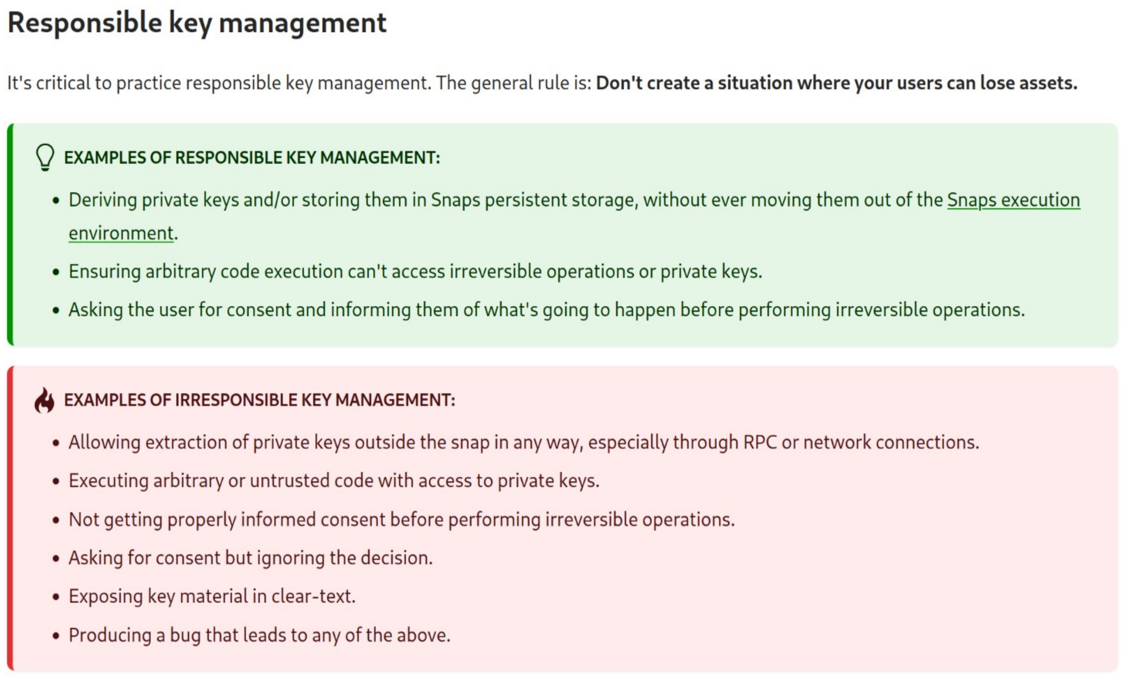
[2] https://www.npmjs.com/package/superstruct

SFH-01-005 WP2: Backup flow exports private keys as plaintext *(Medium)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

The wallet management methods *checkMnemonic* and *recoverApproval* return key-shard mnemonics in plaintext. This is in violation of MetaMask's own recommendations for key backup management, as shown in the figure below.

Affected files:
packages/snap/src/mpc-flow/walletManage.ts



Responsible key management

It's critical to practice responsible key management. The general rule is: **Don't create a situation where your users can lose assets.**

EXAMPLES OF RESPONSIBLE KEY MANAGEMENT:

- Deriving private keys and/or storing them in Snaps persistent storage, without ever moving them out of the Snaps execution environment.
- Ensuring arbitrary code execution can't access irreversible operations or private keys.
- Asking the user for consent and informing them of what's going to happen before performing irreversible operations.

EXAMPLES OF IRRESPONSIBLE KEY MANAGEMENT:

- Allowing extraction of private keys outside the snap in any way, especially through RPC or network connections.
- Executing arbitrary or untrusted code with access to private keys.
- Not getting properly informed consent before performing irreversible operations.
- Asking for consent but ignoring the decision.
- Exposing key material in clear-text.
- Producing a bug that leads to any of the above.

Fig.: MetaMask's recommendations for responsible key management[3]

It is recommended to follow MetaMask's recommendations in the available documentation. Recovery keys could be stored in Snaps' persistent storage, or otherwise exported in an encrypted form with user-passwords.

[3] https://docs.metamask.io/snaps/how-to/manage-keys/#responsible-key-management

SFH-01-006 WP2: Suboptimal state follows request flow *(Low)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

In both *signer* and *recovery* flows, private values such as *signKey* are retrieved from *StateManager* and stored in their respective *object* property in an unencrypted form. Furthermore, both flows fail to clear the *object* properties properly (including the *signKey* property) after their flow/session is finished.

Affected file:
packages/snap/src/StateManager.ts

While this state may be overwritten in any subsequent sessions, during the period between these two sessions, properties such as *signKey* stay in the *object* unencrypted.

It is recommended to clear the entire state including all *object* properties rather than just *signKey*. This should take place as soon as the round is finished, also in the context of inclusion for the *runRound* method in *signer* flow and the *refreshSuccess* method in the recovery flow.

SFH-01-007 WP2: Use of deprecated MetaMask permissions in Snaps *(Low)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

MetaMask Snaps include the deprecated *endowment:long-running* from stable releases, which is currently available only in MetaMask Flask[4]. This permission[5] means that a Snap can run indefinitely, essentially bypassing the lifecycle requirements[6] set for Snaps.

Affected file:
packages/snap/snap.manifest.json

While no exploits were found for this permission during this audit, this could still have potential security consequences if not removed.

[4] https://github.com/MetaMask/snaps/issues/945
[5] https://docs.metamask.io/snaps/reference/permissions/\#endowmentlong-running
[6] https://docs.metamask.io/snaps/concepts/lifecycle/

In addition, the Safeheron Snap uses *mpc_snapKeepAlive* method which keeps the Snap alive, without needing *endowment:long-running*. Therefore, it is recommended to remove the *endowment:long-running* endowment from the Snap config entirely.

SFH-01-008 WP2: *localhost* string included on the allow-list *(Low)*

***Note**: This issue was fixed by the development team during the test. The fix was reviewed and verified successfully by Cure53. The problem no longer exists.*

The Safeheron Snap's allow-list includes *localhost*, which may offer an option for a malicious local application to conduct rogue signatures. This would hinge on chaining this vulnerability with SFH-01-002. Note that the Safeheron Snap uses the allow-list to keep requests to the Snap limited to the featured domains.

Affected file:
packages/snap/src/rpc/permissions.ts

Affected code:
```
const local_websites = [
    'http://localhost:8080',
    'http://127.0.0.1:8080',
    'https://test-mpcsnap.safeheron.com',
    'https://mpcsnap.safeheron.com',
]
```

No other domains apart from those on the list are allowed to send requests to the Snap. However, a malicious app hosted on *localhost* by a user can perform actions with the same permissions as the dApp.

Combined with the existing vulnerability where signing flow does not require user permission (see SFH-01-002), a malicious app someone deploys on their *localhost* would be capable of signing requests on the user's behalf and without their permission.

It is recommended to remove *localhost* from the allow-list. Simply deleting the *localhost* entry from the allow-list is advised, with the caution for identifying both entries of this kind.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

Cure53 can conclude that the Safeheron MPC's MetaMask Snap already boasts a good security posture. However, it is still strongly recommended to swiftly resolve all of the spotted problems, as they clearly indicate that further refinement of the security implementation is needed.

To reiterate, the work completed in the frames of *SFH-01* was split into two WPs, both leveraging penetration testing and code auditing. WP1 focused on the Safeheron WASM MPC code and implementation, whereas WP2 targeted the Safeheron Multi-MPC Snap code and implementation. Three members of Cure53 completed the project over the period of eighteen days in September 2023.

Even though both WPs were originally expected to be completed with white-box methods, some problems and complications arose during the setup and preparations for WP1. Ultimately, black-box methods were deployed in WP1.

During the assessment, Cure53 identified several areas where security practices could be enhanced. While the technical details are given in individual tickets, the overarching observation is that some security measures are in place, but gaps are also present.

In total, eight vulnerabilities were identified. Importantly, their severity scores were *Low* and *Medium,* with no grave risks pinpointed. The first finding concerns overwriting of previous MPC account data (SFH-01-001), which is driven by the current state management layer's failure to warn users about overwriting the existing key-shard data. The problem could lead to potential key loss.

The second finding demonstrates that no user-interaction is required for request signatures in certain scenarios (SFH-01-002). Specifically, the Snap allows such requests and that can lead to unauthorized signatures. The next two findings respectively cover logging of private keys directly in the web renderer's console (SFH-01-003) and the fact that the Snap fails to validate the type and lengths of request payloads in certain areas (SFH-01-004).

Cure53 also found that the backup flow exposes key-shard mnemonics in plaintext (SFH-01-005), which is against MetaMask's own recommendations. The Snap also does not clear *object* properties, including sensitive ones, after the flow/session concludes, as discussed in SFH-01-006.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Final two concerns pertain to the usage of deprecated MetaMask Snap permissions (SFH-01-007) and a peculiar inclusion of *localhost* on the allow-list (SFH-01-008), which needs to be removed to prevent exploitation by malicious local applications.

Recommendations have been provided for each vulnerability as a guideline on how to enhance the security posture of the Snap. It is imperative for Safeheron to address these vulnerabilities promptly, as this will foster trustworthiness and safe operations of the platform for the user-base.

During the audit, the testing team gained an overall impression that Safeheron has made efforts in securing their platform. However, there is still room for improvement. The vulnerabilities identified stem from a combination of oversight in code implementation, lack of user-interaction in critical processes, and non-adherence to industry best practices. Some of these vulnerabilities, if exploited, could compromise user-data and trust. Notably the following aspects are deemed especially improvement-worthy:

- User-interaction: Critical processes, such as key generation and request signatures, lack sufficient user-interaction. This could lead to unintended actions without users' knowledge.
- Logging practices: Logging of sensitive information, especially private keys, is a significant concern. This is a basic security principle that was overlooked by Safeheron.
- Validation mechanisms: Payload validation is inconsistent, which could be exploited by malicious actors to perform unauthorized actions.
- Deprecated practices: The use of deprecated permissions and inclusion of *localhost* on the allow-list indicate a pronounced need for regular updates and adherence to constantly evolving best practices.

While these are the primary concerns, there were other minor issues that, when combined, could pose a risk. More rigorous implementation and regular reviews are needed to capitalize on the existing security-foundation of the Safeheron complex.

To conclude, the Safeheron MPC's MetaMask Snap has potential, but the security implementation calls for refinement. Cure53 strongly recommends collaborative and comprehensive approaches, possibly involving further training and scheduling additional security assessments to bolster adherence to best practices and solidity of the deployed protective measures.

Cure53 would like to thank Qian Yisijie and Yan Jie from the Safeheron Pte. Ltd. team for their excellent project coordination, support and assistance, both before and during this assignment.