Veridise. **Auditing Report**

Hardening Blockchain Security with Formal Methods

FOR

MINA

Mina Snap

Veridise Inc.
July 31, 2023

► **Prepared For:**

Mina Foundation
`https://minaprotocol.com/`

► **Prepared By:**

Jacob Van Geffen
Bryan Tan

► **Contact Us:** `contact@veridise.com`

► **Version History:**

Jul. 31, 2023	V1.01 - Fix typos
Jul. 07, 2023	V1

Contents

1 Executive Summary

From Jun. 20, 2023 to Jun. 22, 2023, Mina Foundation engaged Veridise to review the security of their Mina Snap. This project implements a wallet for the Mina protocol as a MetaMask snap, and it includes features such as managing private keys, managing known Mina networks, and submitting transactions. Interactions with a Mina network node take place via the node's GraphQL API.

The security review covered the TypeScript implementation of the snap component of the project. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on Git commit `240fbbe`. The auditing strategy involved extensive manual auditing by the Veridise engineers.

Code assessment. The Mina Snap developers provided the source code of the Mina Snap implementation for review. To facilitate the Veridise auditors' understanding of the code, the Mina Snap developers also demonstrated and provided an example frontend application (not in the scope of the audit) that can be used with the snap. The source code also contained some documentation in the form of READMEs and documentation comments on functions and storage variables.

The source code did not contain a test suite, although the Veridise auditors noted that several files in the source code also indicate that the developers use linting and static analysis tools such as ESLint.

Summary of issues detected. The audit uncovered 22 issues, 2 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, malicious or buggy frontends or snaps can extract the user's private keys (V-MISN-VUL-001) or delete them (V-MISN-VUL-002) without any explicit authorization from the user. The Veridise auditors also identified 10 low-severity issues, including a race condition in the account and network logic (V-MISN-VUL-011), overly broad snap permissions (V-MISN-VUL-005), and misleading confirmation prompts when submitting transactions (V-MISN-VUL-004). Lastly, the auditors identified 8 warnings as well as a few informational issues. The Mina Snap developers resolved all 22 issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Mina Snap. Mainly, the auditors felt uncomfortable that there is no automated testing, and they noted that some parts of the code could be tested independently of the snap. For example, some of the code interacting with the GraphQL endpoints do not depend on any Snap-specific functionality; issues such as V-MISN-VUL-012 could be caught with integration tests.

Most of the issues raised by the auditors relate to user confirmation and error handling, such as V-MISN-VUL-001 and V-MISN-VUL-004. To help identify similar issues in the future, we recommend that the Mina Snap undergo thorough QA testing and user acceptance testing before the project is released to the public.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Mina Snap	`240fbbe`	TypeScript	MetaMask Snaps

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jun. 20 - Jun. 22, 2023	Manual & Tools	2	6 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	2	2
Medium-Severity Issues	0	0
Low-Severity Issues	10	10
Warning-Severity Issues	8	8
Informational-Severity Issues	2	2
TOTAL	22	22

Table 2.4: Category Breakdown.

Name	Number
Authorization	5
Data Validation	5
Usability Issue	3
Optimization	3
Maintainability	2
Logic Error	1
Access Control	1
Information Leakage	1
Race Condition	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Mina Snap's smart contracts. In our audit, we sought to answer the following questions:

- Is it possible for the user's private keys to leak to an external source?
- Is the key management functionality implemented correctly?
- Are all actions correctly authorized by the user?
- Do authorization prompts contain enough relevant information for the user to make an informed decision?
- Does the business logic correctly handle errors?
- Is untrusted input correctly validated and/or sanitized?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved extensive code review.

Scope. The scope of this audit is limited to the `packages/snap` folder of the source code provided by the Mina Snap developers. Specifically, the audit only covers the implementation of the snap component of the Mina Snap. During the audit, the Veridise auditors referred to external packages and services used by the Mina Snap but assumed that they have been implemented correctly.

Methodology. To understand the intended behavior of the Mina Snap, the Veridise auditors first met with the Mina Snap developers, who provided a live demonstration of the Mina Snap. They then began a manual review of the code. During the audit, the Veridise auditors regularly met with the Mina Snap developers to ask questions about the code. The auditors also referred to documentation found on the Mina protocol website.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-MISN-VUL-001	mina_exportPrivateKey does not obtain user's au. . .	High	Fixed
V-MISN-VUL-002	mina_resetSnapConfig does not prompt for user c.	High	Fixed
V-MISN-VUL-003	mina_changeNetwork silently no-ops on invalid n.	Low	Fixed
V-MISN-VUL-004	The sendZkAppTx() prompt has misleading inforr	Low	Fixed
V-MISN-VUL-005	Overly broad permission on RPC endowment	Low	Fixed
V-MISN-VUL-006	mina_exportPrivateKey logs the private key to c. . .	Low	Fixed
V-MISN-VUL-007	mina_changeAccount does not confirm user intent	Low	Intended Behavior
V-MISN-VUL-008	sendPayment() and sendStakeDelegation() hard-co	Low	Fixed
V-MISN-VUL-009	sendStakeDelegation() dialog title is too vague	Low	Fixed
V-MISN-VUL-010	Unhandled notification failure can cause snap t. . .	Low	Fixed
V-MISN-VUL-011	Potential race condition when modifying snap co. .	Low	Fixed
V-MISN-VUL-012	Transaction submit failureReason not checked	Low	Fixed
V-MISN-VUL-013	getMinaClient() does not check for invalid netw. . .	Warning	Fixed
V-MISN-VUL-014	createAccount() is inefficiently implemented	Warning	Fixed
V-MISN-VUL-015	Some RPCs are never used by the front end	Warning	Fixed
V-MISN-VUL-016	Non-compliance with BIP44 account discovery alg	Warning	Acknowledged
V-MISN-VUL-017	getAccounts() performs O(N) HTTP queries	Warning	Acknowledged
V-MISN-VUL-018	No notification shown in submitZkAppTx()	Warning	Fixed
V-MISN-VUL-019	GraphQL HTTP response code not checked	Warning	Fixed
V-MISN-VUL-020	RPC method parameters are not validated	Warning	Acknowledged
V-MISN-VUL-021	Consider enabling GitHub security scanning	Info	Fixed
V-MISN-VUL-022	Multiple GraphQL queries can be batched	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-MISN-VUL-001: mina_exportPrivateKey does not obtain user's authorization

Severity	High	**Commit**	240fbbe
Type	Authorization	**Status**	Fixed
File(s)	`src/index.ts`		
Location(s)	onRPCRequest()		
Fixed At	6c59b7a		

The `mina_exportPrivateKey` method can be called to "export" the private key of the specified account to the caller. However, the method will unconditionally return the private key, even if the user may not want to export the private key to the caller.

```
case EMinaMethod.EXPORT_PRIVATE_KEY: {
  const { index, isImported } = request.params as { index?: number; isImported?:
    boolean };
  const { privateKey } = await getKeyPair(index, isImported);
  console.log('-privateKey:', privateKey);
  return { privateKey };
}
```

Snippet 4.1: Implementation of the `mina_exportPrivateKey` method in `onRPCRequest()`

Impact If the user has enabled the snap for the current website, then any JavaScript code that executes on that website will be able to obtain the user's private keys. This can be exploited by malicious actors that employ XSS and phishing attacks. Note that this issue is further exacerbated by V-MISN-VUL-005, where malicious snaps could also steal the private keys.

Recommendation The developers mentioned that they wanted to implement a password prompt, but they found that there is currently no easy way to do this within MetaMask. Despite this, we would still recommend that the snap requests the user's authorization before proceeding, so as to mitigate the damage done by XSS or phishing attacks. For example, the snap can show a confirmation dialog to the user that displays the risks of exporting the private key (or a link to such an explanation) and the origin of the RPC call. The user must click "Accept" before the export can proceed.

4.1.2 V-MISN-VUL-002: mina_resetSnapConfig does not prompt for user confirmation

Severity	High	**Commit**	240fbbe
Type	Authorization	**Status**	Fixed
File(s)	`src/index.ts,src/mina/configuration.ts`		
Location(s)	onRPCRequest(), resetSnapConfiguration()		
Fixed At	6c59b7a, 3f90632		

The `mina_resetSnapConfig` method can be called clear all data stored by the Mina snap. However, this is performed even without any user confirmation.

```
// Implementation of the mina_resetSnapConfig method in onRPCRequest()
case EMinaMethod.RESET_CONFIG: {
  return resetSnapConfiguration();
}

// The definition of resetSnapConfiguration()
export const resetSnapConfiguration = async (): Promise<NetworkConfig> => {
  await snap.request({
    method: ESnapMethod.SNAP_MANAGE_STATE,
    params: { operation: 'clear' },
  });
  const snapConfig = await getSnapConfiguration();
  const networkConfig = await getNetworkConfig(snapConfig);
  return networkConfig;
};
```

Snippet 4.2: The relevant locations in the code. None of the called functions will prompt for user confirmation.

Impact Without user confirmation, the user may be more prone to accidentally deleting all of their data, including their private keys. While automatically generated private keys can be recovered as they are deterministic, how to do so is not obvious to a non-technical user. Furthermore, there will be no way to recover imported private keys.

Additionally, a malicious snap or frontend can call `mina_resetSnapConfig` to make an attack more damaging. For example, an attacker can first exploit V-MISN-VUL-001, and then they can call `mina_resetSnapConfig` to make it more difficult for a victim to regain control over their accounts.

Recommendation The snap should prompt the user for confirmation before invoking `resetSnapConfiguration()`. The prompt should clearly indicate the risks of resetting the snap configuration and require the user to type in a message such as “I understand that resetting the snap may result in loss of access to accounts.” for confirmation.

4.1.3 V-MISN-VUL-003: mina_changeNetwork silently no-ops on invalid network

Severity	Low	**Commit**	240fbbe
Type	Logic Error	**Status**	Fixed
File(s)	`src/mina/configuration.ts`		
Location(s)	changeNetwork()		
Fixed At	d38b39d		

Given a `networkName`, the `changeNetwork()` function is used to update the currently selected network in the persistent state to the `networkName`. This is only performed if `networkName` is not the currently active network, and the network actually exists. The `changeNetwork()` function is invoked on calls to the `mina_changeNetwork` JSON-RPC endpoint. Neither of these validate that the given network actually exists, so if the caller provides an invalid network, then the method will return successfully without having performed any state changes.

```
export const changeNetwork = async (networkName: ENetworkName): Promise<NetworkConfig
    > => {
  let snapConfig = await getSnapConfiguration();
  if (networkName != snapConfig.currentNetwork && snapConfig.networks[networkName]) {
    snapConfig.currentNetwork = networkName;
    await updateSnapConfig(snapConfig);
  }
  const networkConfig = await getNetworkConfig(snapConfig);
  return networkConfig;
};
```

Snippet 4.3: Implementation of `changeNetwork()`

Impact If a dApp (or a user) provides an invalid network name to `mina_changeNetwork`, then the snap will not do anything. The user may mistakenly believe that the network has been changed, which can cause subsequent accidents or errors. For example, the user may accidentally send funds with an account on the wrong network.

Recommendation Throw an error if the network does not exist.

4.1.4 V-MISN-VUL-004: The sendZkAppTx() prompt has misleading information

Severity	Low	**Commit**	240fbbe
Type	Authorization	**Status**	Fixed
File(s)	`src/mina/index.ts`		
Location(s)	sendZkAppTx()		
Fixed At	6c59b7a		

The `mina_sendTransaction` endpoint is used to send a zkApp transaction using the currently selected account. This is implemented by the `sendZkAppTx()` function. In order for the transaction to be made, the user will first be prompted for confirmation. The prompt will be titled "Confirm transaction" and it will show both the user's account as well as the fee provided by the fee payer (in terms of MINA). This prompt has several problems:

1. The prompt does not show any details about the target of the transaction.
2. The network is not shown.
3. Similar to V-MISN-VUL-008, the fee is always denominated in "MINA"; however, the actual token is that of the current network.
4. The title does not indicate that the transaction is a zkApp transaction.

```
export const sendZkAppTx = async (args: ZkAppTxInput, networkConfig: NetworkConfig)
    => {
  const { publicKey, privateKey } = await getKeyPair();

  const confirmation = await popupDialog(
    ESnapDialogType.CONFIRMATION,
    'Confirm transaction',
    `Submit ZkApp transaction \nFrom: ${publicKey}\nFee: ${args.feePayer.fee} MINA`,
  );
  if (!confirmation) {
    await popupNotify('Transaction rejected');
    return null;
  }
```

Snippet 4.4: Lines in `sendZkAppTx()` that prompt the user for confirmation

Impact

1. Because no details are shown about the target of the transaction, the user cannot make an informed decision about whether they should accept the transaction. It may be easier for malicious actors to trick users into sending funds to the wrong addresses or executing harmful transactions. For example, a malicious actor might trick a user into calling a token contract's transfer function to drain the user's tokens, but the user has no way of knowing that from the confirmation dialog.
2. Since the network is not shown, it is more likely for a user to accidentally send (and approve) a transaction on the wrong network.
3. When the network does not have "MINA" as its currency, the fee will still be displayed in terms of "MINA". The user may be mislead into thinking that the actual cost will be in terms of MINA, but the fees will actually be paid in a different currency.

```
export async function popupDialog(
  type: ESnapDialogType,
  prompt: string,
  textAreaContent: string,
) {
  const response = await snap.request({
    method: ESnapMethod.SNAP_DIALOG,
    params: {
      type,
      content: panel([
        heading(prompt),
        text(textAreaContent),
      ])
    }
  });
  return response;
}
```

Snippet 4.5: Implementation of `popupDialog()`, which constructs a snap UI consisting of a heading and a paragraph.

Recommendation The developers should revise the prompt to contain additional, unambiguous information about the transaction.

1. We recommend that the developers display at **minimum**: the network, the target address of the transaction, any `memo`, and any information about the method being invoked (and potentially the arguments). This can be done under a separate heading (e.g., "Transaction Details") to avoid cluttering the UI. Security may also be improved by displaying the origin of the JSON-RPC request received by the snap.
2. Instead of hardcoding `MINA` in the string, the `MINA` should be replaced with a code snippet such as `${networkConfig.token.symbol}`.
3. The title of the prompt should be changed to "Confirm zkApp transaction" to clarify that the transaction involves a zkApp.

Developer Response The developers updated the dialog title and made the dialog show the network name and the zkApp transaction details.

4.1.5 V-MISN-VUL-005: Overly broad permission on RPC endowment

Severity	Low	**Commit**	240fbbe
Type	Access Control	**Status**	Fixed
File(s)	`snap.manifest.json`		
Location(s)	N/A		
Fixed At	6c59b7a		

The `snap.manifest.json` for this snap grants access to the `endowment:rpc` permission. Specifically, it grants both `dapps` and `snaps` permission. Based on a demonstration of a frontend application written by the developers, the `dapps` permission seems necessary. However, it is not clear why the `snaps` permission is necessary.

Impact The `snaps` permission allows other snaps to communicate with the Mina snap. Other snaps would be able to invoke methods such as `mina_exportPrivateKey` to extract information from the Mina snap.

Recommendation Remove the `snaps` permission if it is not necessary.

Developer Response The developers removed the `snaps` permission.

4.1.6 V-MISN-VUL-006: mina_exportPrivateKey logs the private key to console

Severity	Low	**Commit**	240fbbe
Type	Information Leakage	**Status**	Fixed
File(s)	`src/index.ts`		
Location(s)	onRPCRequest()		
Fixed At	6c59b7a		

The `mina_exportPrivateKey` method can be called to "export" the private key of the specified account to the caller. However, this will log the private key to the developer console.

```
case EMinaMethod.EXPORT_PRIVATE_KEY: {
  const { index, isImported } = request.params as { index?: number; isImported?:
    boolean };
  const { privateKey } = await getKeyPair(index, isImported);
  console.log('-privateKey:', privateKey);
  return { privateKey };
}
```

Snippet 4.6: Implementation of the `mina_exportPrivateKey` method in `onRPCRequest()`

Impact The private key will be unintentionally exposed to the developer console. This becomes an attack vector for a phishing attack: if a phisher successfully convinces a user to open the developer console and then export the private keys to a frontend that does not display the private keys, then the phisher will be able to steal the private keys.

Recommendation Remove the line that logs the private key to the console.

4.1.7 V-MISN-VUL-007: mina_changeAccount does not confirm user intentions

Severity	Low	**Commit**	240fbbe
Type	Authorization	**Status**	Intended Behavior
File(s)	`packages/snap/src/index.ts`		
Location(s)	onRPCRequest()		
Fixed At	N/A		

When changing accounts, the RPC `mina_changeAccount` only requires the index of the account and the `isImported` flag; no confirmation or password is required from the caller. As a result, any malicious actors that can make calls to `mina_changeAccount` can arbitrarily change accounts.

```
case EMinaMethod.CHANGE_ACCOUNT: {
  const { accountIndex, isImported } = request.params as { accountIndex: number;
    isImported?: boolean };
  const accountInfo = await changeAccount(accountIndex, isImported);
  console.log('-account changed to:', accountInfo);
  return accountInfo;
}
```

Snippet 4.7: Relevant lines in `onRPCRequest()`

Impact Without restrictions to changing accounts, malicious websites with access to the snap could change the active account without the user's consent. This could lead to unexpected behavior from the user's point of view, such as transactions going to/coming from the wrong account. For example, an attacker can silently change the account before issuing a transaction, taking advantage of V-MISN-VUL-004so that users will be tricked into transferring funds to the attacker.

Recommendation There are several ways to guard against malicious browsers/front-ends from arbitrarily changing the account:

1. Require a password dialogue to change accounts. Although we understand that MetaMask Flask currently does not expose this functionality, requiring a password from the user before switching accounts could help prevent malicious behavior from the frontend.
2. Include a confirmation pop-up for the user when changing accounts. While this would not be as secure as requiring the password, we believe it is a reasonable substitute while MetaMask does not enable password pop-ups.

Developer Response The developers noted that the current behavior is consistent with how MetaMask works, which doesn't prompt the user on account change. They chose to make `mina_changeAccount` work in this way to help improve the user experience.

4.1.8 V-MISN-VUL-008: sendPayment() and sendStakeDelegation() hard-codes "MINA" as the token

Severity	Low	**Commit**	240fbbe
Type	Authorization	**Status**	Fixed
File(s)	`packages/snap/src/mina/index.ts`		
Location(s)	sendPayment(), sendStakeDelegation()		
Fixed At	6c59b7a		

In `sendPayment()`, a call to `popupDialogue(...)` confirms that the transaction to be sent is correct according to the user. However, this dialogue hard-codes `MINA` as the token. In reality, the type of token is determined by `networkConfig.token` and not fixed to `MINA`. A similar issue occurs in `sendStakeDelegation()`.

```
export const sendPayment = async (args: TxInput, networkConfig: NetworkConfig) => {
  const { publicKey, privateKey } = await getKeyPair();

  const confirmation = await popupDialog(
    ESnapDialogType.CONFIRMATION,
    'Confirm transaction',
    ‘From: ${publicKey}\nTo: ${args.to}\nAmount: ${args.amount} MINA\nFee: ${args.fee
    } MINA‘,
  );
  ...
}
```

Snippet 4.8: Relevant lines in `sendPayment()`

Impact The user may unintentionally approve a payment transaction of the wrong amount, or reject a transaction that has the correct amount whenever the token being sent is not `MINA`.

Recommendation Replace the instances of `MINA` with `${networkConfig.token.symbol}` in the popup dialogues of `sendPayment()` and `sendStakeDelegation()`.

4.1.9 V-MISN-VUL-009: sendStakeDelegation() dialog title is too vague

Severity	Low	**Commit**	240fbbe
Type	Usability Issue	**Status**	Fixed
File(s)	`src/mina/index.ts`		
Location(s)	sendStakeDelegation()		
Fixed At	6c59b7a, 5b634c6		

When the `mina_sendStakeDelegation` JSON-RPC method is invoked, it will show a confirmation dialog to the user before submitting a stake delegation transaction to a Mina node. This dialog will be titled "Confirm transaction" and show some details about the transaction. The title is vague and does not clearly indicate that the transaction is a stake delegation transaction. Furthermore, if the user does not approve the transaction, a vague "Transaction rejected" message will be displayed in the MetaMask notifications.

```
export const sendStakeDelegation = async (args: StakeTxInput, networkConfig:
    NetworkConfig) => {
  const { publicKey, privateKey } = await getKeyPair();

  const confirmation = await popupDialog(
    ESnapDialogType.CONFIRMATION,
    'Confirm transaction',
    `Block producer address: ${args.to}\nFrom: ${publicKey}\nFee: ${args.fee} MINA`,
  );
  if (!confirmation) {
    await popupNotify('Transaction rejected');
    return null;
  }
```

Snippet 4.9: Relevant lines in `sendStakeDelegation()`

Impact Users may be confused by the confirmation prompt and/or notification and believe that the transaction may be of a different type, such as a payment or zkApp transaction. This confusion could potentially be exploited by attackers in social engineering attacks.

Recommendation Change the title of the dialog and the notification message to be more specific, such as to "Confirm stake delegation transaction" and "Stake delegation transaction rejected", respectively.

4.1.10 V-MISN-VUL-010: Unhandled notification failure can cause snap to be terminated

Severity	Low	**Commit**	240fbbe
Type	Data Validation	**Status**	Fixed
File(s)	`src/util/popup.util.ts`		
Location(s)	popupNotify()		
Fixed At	5b634c6		

The `popupNotify()` helper function is used to display a notification through the `snap_notify` JSON-RPC method. This can fail if MetaMask Flask's notification rate-limiting triggers, so that the promise completes with a rejection. As a result, an error may be thrown if this occurs; since there is no logic that checks for an error, the snap will be terminated unexpectedly.

```
export async function popupNotify(message: string) {
  await snap.request({
    method: ESnapMethod.SNAP_NOTIFY,
    params: { type: 'native', message },
  });
}
```

Snippet 4.10: Implementation of popupNotify()

Impact The `popupNotify()` method is called by `submitStakeDelegation()` and `submitPayment()`. If a user sends payments or stake delegation transactions rapidly, then multiple notifications will be requested. This will cause the rate limiter to activate and cause the `snap.request(...)` to return a rejected promise, eventually causing an error that terminates the snap.

Recommendation Handle potential errors from failed notifications by wrapping `await snap.request(...)` in a try-catch block (logging any errors), or using the `.catch` method of `Promise`, as in `(await snap.request(...)).catch(err => { ... })`.

4.1.11 V-MISN-VUL-011: Potential race condition when modifying snap configuration

Severity	Low	**Commit**	240fbbe
Type	Race Condition	**Status**	Fixed
File(s)	`src/mina/configuration.ts,src/mina/accounts.ts`		
Location(s)	See description		
Fixed At	None		

Several functions concerning the account and network logic may read or modify the persistent state in a way that is prone to race conditions, specifically, if there are concurrent JSON-RPC method calls being handled by the snap. There do not seem to be any safeguards or protections against race conditions in the code.

Example Race Condition Scenario As an example of a race condition, consider the following scenario:

1. User switches network on the frontend, which calls the JSON-RPC `mina_changeNetwork` method on the snap.
2. User quickly then triggers the snap's `mina_sendPayment` JSON-RPC method from the frontend.
3. The `mina_changeNetwork` call is received by the snap and proceeds to the point where the persistent state is retrieved in `changeNetwork()`.
4. The `mina_sendPayment` call is received by the snap and proceeds to the point where the keypair of the current account is fetched from the persistent state.
5. `changeNetwork()` updates the persistent state
6. The payment confirmation dialog pops up for the old network/account, but the new network/account is expected.

The user will expect the payment confirmation dialog to pop up for the new network, but it will show the account for the old network instead.

As another example, consider what happens if `mina_sendPayment` is instead replaced by `mina_editAccountName`: the account being edited will be on the old network, and the persistent state saved will use the state *before* the network was changed, so the final state stored will not have the network change.

Impact A race condition can be triggered if a user modifies an account or the network selection in multiple places simultaneously, such as in different tabs or if the actions are performed quickly enough. This can cause subtle bugs, as shown in the above example. While we have not confirmed whether it is possible in practice, the example race condition scenario above shows that it is possible in theory.

The functions that write to the persistent state include:

- `changeNetwork()`
- `changeAccount()`
- `createAccount()`
- `importAccount()`

- `editAccountName()`
- `resetSnapConfiguration()`

The functions that read from the persistent state include the above functions, as well as `onRPCRequest()` (and hence in all logic related to the JSON-RPC methods) and `generateKeyPair()`.

Any combination of concurrent JSON-RPC method calls involving at least one function writing to persistent state as well as any other function writing to or reading from persistent state is prone to a race condition.

Recommendation To reduce the possibility of a race condition, the developers could use a mutex, semaphore, or other synchronization primitive to ensure 1) each JSON-RPC method call has exclusive access to critical sections; and 2) the "happened-before" relationship is maintained when reading or modifying persistent state. The former is likelier to happen and has a larger impact, while the latter is not as likely and not as important to correctness.

We note that adding synchronization will likely increase the complexity of the codebase and introduce subtle bugs such as deadlocks. Ensuring criterion (1) would introduce moderate complexity, while ensuring criterion (2) would introduce significant complexity. The developers should carefully weigh the tradeoffs of any fix being made.

One specific mitigation would be to use a read-write lock to restrict concurrent access to the persistent state, especially for "read-modify-write" patterns which are meant to be atomic. This would allow concurrent readonly operations to occur simultaneously, while disallowing a read-modify-write and another persistent state operation from occurring simultaneously. While this will not ensure criterion (2), this can ensure criterion (1) holds.

Developer Response The developers changed the code to use a mutex so that only one JSON-RPC method call can be handled at a time. The auditors note that this can mitigate race conditions, but at the cost of potentially opening up denial-of-service attack vectors if some JSON-RPC methods take a long time to run (esp. those that send HTTP requests).

4.1.12 V-MISN-VUL-012: Transaction submit failureReason not checked

Severity	Low	**Commit**	240fbbe
Type	Data Validation	**Status**	Fixed
File(s)	`src/mina/transaction.ts`		
Location(s)	submitPayment(), submitStakeDelegation(), submitZkAppTx()		
Fixed At	5b634c6		

The methods `submitPayment()`, `submitStakeDelegation()`, and `submitZkAppTx()` each submit a Mina protocol transaction by sending a GraphQL mutation to a Mina node. In response, the Mina node will return an object containing the response to the transaction submission. This object will contain a `failureReason` list that is nonempty if an error has occurred. However, none of the above methods or its callers check whether `failureReason` is empty (i.e., check for errors).

Impact If there is a nonempty `failureReason`, then an error has occurred. The snap will still show notifications indicating that the transaction has been submitted, but in reality the submission has failed.

Recommendation The developers should confirm whether the error should be handled by the snap or by the caller of the JSON-RPC method. If the error is meant to be handled in the snap, the developers should add code to check that `failureReason` is empty or null (i.e., that there are no errors). Note that the `failureReason` schema for a zkApp transaction is slightly different from that of a payment or stake delegation transaction.

4.1.13 V-MISN-VUL-013: getMinaClient() does not check for invalid networks

Severity	Warning	**Commit**	240fbbe
Type	Data Validation	**Status**	Fixed
File(s)	`src/util/mina-client.util.ts,src/constants/config.constant.ts`		
Location(s)	getMinaClient()		
Fixed At	6c59b7a, 369fed9		

The `getMinaClient()` function is used to construct an instance of `Client` (from the `mina-signers` library) that will be used to sign transactions. The client takes a `network` argument that determines what type of Mina network that the actions will be for, either `mainnet` or `testnet`. This will be determined based on the "network name" in the configuration of the snap. Specifically, if the network name is `Mainnet`, then the network type will be set to `mainnet`; otherwise, the network type will be set to `testnet`.

```
export const getMinaClient = (networkConfig: NetworkConfig) => {
  if (networkConfig.name === ENetworkName.MAINNET) {
    return new Client({ network: 'mainnet' });
  }
  return new Client({ network: 'testnet' });
};
```

Snippet 4.11: Definition of `getMinaClient()`

Impact Currently, this has no impact as there are only three Mina networks mainnet, devnet, and berkeley, with the latter two being `testnet` type.

However, if more mainnet type networks are added to the configuration in the future, then the developers may forget to add a the case in `getMinaClient()`. Such additional mainnet type networks will be assumed to be `testnet` type, which could result in bugs.

Recommendation The developers should modify `getMinaClient()` so that it checks exactly for the cases specified in `config.constant.ts`. If an invalid network name is provided, the function should throw an error.

4.1.14 V-MISN-VUL-014: createAccount() is inefficiently implemented

Severity	Warning	**Commit**	240fbbe
Type	Optimization	**Status**	Fixed
File(s)	`src/mina/account.ts`		
Location(s)	createAccount()		
Fixed At	4bdaec0		

The JSON-RPC method `mina_createAccount`, which is implemented by the `createAccount()` function, allows the caller to create a new generated account. The `createAccount()` first function recursively searches for the smallest index in the `generatedAccounts` array that is corresponding to a public key that is not already in the list of generated or imported accounts. Then it updates the persistent state with the newly generated account.

The `createAccount()` function takes an optional `index` parameter that indicates the index that the search should begin at. Note that if `index` is **not** provided, the search will begin at the largest index of the generated accounts; however, if `index` is provided, the search will begin at `index`. Note that the recursive call to `createAccount()` explicitly provides the index. This means that if the original JSON-RPC call explicitly provides a small `index` such as 0, and there are already many existing accounts, then there will be $O(n)$ recursive calls to `createAccount()`, where n is the total number of accounts.

Impact If the frontend explicitly provides a small value for `index` and there are already a large number of accounts, then there could be a large number of recursive calls. The performance here may suffer as the snap will retrieve the persistent state and generate a key pair on each recursive call.

Recommendation

- The developers should change the case where index is explicitly provided so that existing accounts are skipped over. For example, one possible fix (untested by the auditors) is:

```
const currentMaxIndex = 0;
const { generatedAccounts } = networks[currentNetwork];
if (Object.keys(generatedAccounts).length) {
  currentMaxIndex = Math.max(...Object.keys(generatedAccounts).map((key) =>
    Number(key)));
}

if (index && index > currentMaxIndex) {
  newAccountIndex = index;
} else {
  newAccountIndex = currentMaxIndex + 1;
}
```

  This change forces input `index` values to be larger than any previous index. The developers could instead choose to ensure that the input `index` does not match any existing keys from `Object.keys(generatedAccounts)`, but this option would complicate the code for incrementing `newAccountIndex` in the case that a duplicate key pair is generated.

```
export const createAccount = async (name: string, index?: number): Promise<any> => {
    const snapConfig = await getSnapConfiguration();
  const { networks, currentNetwork } = snapConfig;
  let newAccountIndex;
  if (index) {
    newAccountIndex = index;
  } else {
    const { generatedAccounts } = networks[currentNetwork];
    if (Object.keys(generatedAccounts).length) {
      const currentMaxIndex = Math.max(...Object.keys(generatedAccounts).map((key) =>
     Number(key)));
      newAccountIndex = currentMaxIndex + 1;
    } else {
      newAccountIndex = 0;
    }
  }
  const { publicKey } = await generateKeyPair(networks[currentNetwork],
    newAccountIndex);
  const duplicateAddress = checkDuplicateAddress(networks[currentNetwork], publicKey)
    ;
  if (duplicateAddress) {
    return createAccount(name, newAccountIndex + 1);
  }
  snapConfig.networks[currentNetwork].currentAccIndex = newAccountIndex;
  snapConfig.networks[currentNetwork].selectedImportedAccount = null;
  snapConfig.networks[currentNetwork].generatedAccounts[newAccountIndex] = { name,
    address: publicKey };
  await updateSnapConfig(snapConfig);
  return { name, address: publicKey };
}
```

Snippet 4.12: Implementation of `createAccount()`

- Instead of recursively calling `createAccount()`, the recursion could be rewritten to use a loop instead. This will prevent `createAccount()` from repeatedly getting the snap configuration from the persistent state.

Developer Response The developers noted:

> Currently, there are no plans to use the `index` in the front-end, so we will remove the `index` parameter here to make the logic simpler.

The developers changed `createAccount()` so that it always calculates the `currentMaxIndex` and replaced the recursion with a loop.

4.1.15 V-MISN-VUL-015: Some RPCs are never used by the front end

Severity	Warning	**Commit**	240fbbe
Type	Maintainability	**Status**	Fixed
File(s)	`src/index.ts`		
Location(s)	onRPCRequest()		
Fixed At	47489b2		

Based on `packages/site`, it appears that the following JSON-RPC methods are not used by the current frontend:

- `hello`
- `mina_verifyMessage`
- `mina_getTxDetail`
- `mina_getTxHistory`
- `mina_resetSnapConfig`
- `mina_sendStakeDelegation`
- `mina_requestNetwork`

Impact These JSON-RPC methods give (possibly unintended) access to functionality like sending a stake delegation transaction, changing the name of a network, or reseting the configuration of the snap. This means that malicious front-ends using this snap could access this functionality when they should not.

Recommendation The actions here will depend on whether or not the developers intend for frontends to access the functionality provided by these JSON-RPC methods in the future.

- If the snap is only meant to work with the current frontend created by the developers, these functionalities should be disabled in the snap, as they will effectively be "dead code" until they are used. This will help reduce attack vectors that can be exploited by malicious websites.
- If this functionality is not ****meant to be used by any future frontends, then these JSON-RPC methods should be deleted from `onRpcRequest()`.

Developer Response The developers noted that they are using these JSON-RPC methods in another project. However, they will remove the `hello` method, as it is unused.

4.1.16 V-MISN-VUL-016: Non-compliance with BIP44 account discovery algorithm

Severity	Warning	**Commit**	240fbbe
Type	Usability Issue	**Status**	Acknowledged
File(s)	`src/mina/account.ts`		
Location(s)	importAccount()		
Fixed At	None		

The BIP44 standard notes that:

> Software should prevent a creation of an account if a previous account does not have a transaction history (meaning none of its addresses have been used before).

However, when accounts are imported into the snap in the `importAccount()` function, there are no such checks for the account transaction history.

Impact We note that this does not appear to have a direct security impact, although it can lead to confusion if users accidentally import the wrong account and the account shows that they have zero funds. In such a situation, users could become irrational and take risky actions that compromise their security. If there was a check for account transaction history, users would immediately realize any such mistakes or accidents.

Recommendation As adding such a check could add unnecessary complexity to the implementation of `importAccount()`, the developers should weigh the tradeoffs of having such a check and include it if they believe it can help reduce user confusion.

Developer Response The developers acknowledged the issue and mentioned that they do not have an immediate fix for this issue.

4.1.17 V-MISN-VUL-017: getAccounts() performs O(N) HTTP queries

Severity	Warning	**Commit**	240fbbe
Type	Optimization	**Status**	Acknowledged
File(s)	`src/mina/account.ts`		
Location(s)	getAccounts()		
Fixed At	None		

The `mina_getAccounts` JSON-RPC method fetches the (public) account information of every account stored in the wallet. This is implemented by the `getAccounts()` function, which will send GraphQL queries (via HTTP) to a Mina protocol node. However, when sending the queries, the `getAccounts()` method will make an individual HTTP request for each account, so that there are $O(n)$ HTTP requests, where n is the total number of accounts.

```
const allAccounts = [...generatedAccountsArr, ...importedAccountsArr];
await Promise.all(
  allAccounts.map((account) => {
    return getAccountInfo(account.address, networks[currentNetwork]).then((data) => {
      account.balance = data.account.balance;
    });
  }),
);
```

Snippet 4.13: Relevant code in `getAccounts()`

```
export async function getAccountInfo(publicKey: string, networkConfig: NetworkConfig)
     {
  const query = getAccountInfoQuery(networkConfig.name === ENetworkName.BERKELEY);
  const variables = { publicKey };

  const data = await gql(networkConfig.gqlUrl, query, variables);
```

Snippet 4.14: Relevant code in `getAccountInfo()`. The `gql()` function will make an HTTP request when invoked.

Impact If the wallet stores many accounts, the snap will make many HTTP requests in parallel. This may result in unnecessary server load on the Mina node and a waste of bandwidth.

Recommendation If possible, modify the GraphQL query to batch the account retrieval and use a single HTTP request to obtain the desired information.

Developer Response The developers acknowledged the issue but do not plan to resolve this in the short term.

4.1.18 V-MISN-VUL-018: No notification shown in submitZkAppTx()

Severity	Warning	**Commit**	240fbbe
Type	Usability Issue	**Status**	Fixed
File(s)	`src/mina/transaction.ts`		
Location(s)	sendZkAppTx()		
Fixed At	5b634c6		

In `sendPayment()` and `sendStakeDelegation()`, a notification is shown when the corresponding transaction is successfully submitted to a Mina node. However, in `submitZkAppTx()`, there is no similar logic to show a notification.

Impact Notifications will provide a transaction history for payment and stake delegation transactions. However, zkApp transactions will not be logged in this way as they have no notifications.

Recommendation The developers should consider whether omitting the notifications is intended and make the code consistent with their expectations.

Developer Response The developers stated that a notification is supposed to be shown.

4.1.19 V-MISN-VUL-019: GraphQL HTTP response code not checked

Severity	Warning	**Commit**	240fbbe
Type	Data Validation	**Status**	Fixed
File(s)	`src/graphql/index.ts`		
Location(s)	gql()		
Fixed At	f0b8b8d		

The `gql()` helper function sends a GraphQL query or mutation over HTTP. The function assumes that the response will be valid GraphQL response that can be deserialized as JSON. In particular, `gql()` does not check the status code of the response. However, if the HTTP response returns a non-200 status code, it is possible for the response to 1) be invalid JSON; or 2) only *appear* to be a valid GraphQL response but is invalid in practice.

```
export async function gql(url: string, query: string, variables = {}) {
  try {
    const response = await fetch(url, /* ... */);
    const { data, errors } = await response.json();
    if (errors) throw new Error(errors[0].message);
    return data;
  } catch (err) {
    console.error('packages/snap/src/graphql/index.ts:30', err.message);
    throw err;
  }
}
```

Snippet 4.15: Relevant lines in `gql()`

Impact The GraphQL Over HTTP specification states that:

> If the response uses a non-`200` status code and the media type of the response payload is `application/json` then the client MUST NOT rely on the body to be a well-formed *GraphQL response* since the source of the response may not be the server but instead some intermediary such as API gateways, proxies, firewalls, etc.

In such a scenario, the snap may assume that a transaction submission is successful when no actual action has been taken. This may confuse the user and cause them to perform potentially damaging actions.

Recommendation The developers should check the value of `response.status()` is equal to `200` and handle any errors appropriately.

4.1.20 V-MISN-VUL-020: RPC method parameters are not validated

Severity	Warning	**Commit**	240fbbe
Type	Data Validation	**Status**	Acknowledged
File(s)	`src/index.ts`		
Location(s)	onRPCRequest()		
Fixed At	None		

In `onRPCRequest()`, the case for each method assumes that the parameters provided by the JSON-RPC method caller have a specific type or structure. However, there is no validation that the parameters **actually** have such type or structure.

```
case EMinaMethod.NETWORK_CONFIG: {
  const { name, gqlUrl, gqlTxUrl, explorerUrl, token } = networkConfig;
  // ...

case EMinaMethod.SEND_PAYMENT: {
  const txInput = request.params as TxInput;
  // ...

// Definition of TxInput in src/interfaces.ts
export type TxInput = {
  to: string;
  amount: number;
  fee: number;
  memo?: string;
  nonce?: number;
};
```

Snippet 4.16: Example of type casting without validation in `onRPCRequest()`.

Impact If the JSON-RPC method caller omits required fields, then the values of the fields will be `undefined`, which can result in bugs. It is good practice to validate required parameters and their types, and doing so can help prevent bugs and potential security vulnerabilities in the future.

Recommendation The developers should insert additional parameter validation where helpful. For example, it would be good to validate that all `number` fields are indeed numbers. More complex parameters such as the parameters to `mina_sendStakeDelegation` should also be explicitly validated.

Developer Response The developers acknowledged the issue but do not plan to fix it in the short term.

4.1.21 V-MISN-VUL-021: Consider enabling GitHub security scanning

Severity	Info	**Commit**	240fbbe
Type	Maintainability	**Status**	Fixed
File(s)	N/A		
Location(s)	N/A		
Fixed At	N/A		

The developers mentioned that they would like to make sure that their dependencies are free of vulnerabilities. We note that the GitHub repository that contains the source code of the snap does not have GitHub code scanning nor Dependabot enabled. We recommend that the developers enable both to receive regular reports about vulnerabilities in the dependency tree.

4.1.22 V-MISN-VUL-022: Multiple GraphQL queries can be batched

Severity	Info	**Commit**	240fbbe
Type	Optimization	**Status**	Acknowledged
File(s)	`packages/snap/src/mina/transaction.ts`		
Location(s)	getTxHistory()		
Fixed At	None		

The function `getTxHistory()` makes four separate GraphQL queries, each in a different HTTP request, in order to fetch transaction history. Note that only two URLs will be accessed by the requests.

```
export async function getTxHistory(networkConfig: NetworkConfig, options:
    HistoryOptions, address: string) {
  let getPendingTxList = gql(networkConfig.gqlUrl, TxPendingQuery(), { address });
  let getTxList = gql(networkConfig.gqlTxUrl, getTxHistoryQuery(), { ...options,
    address });
  let getZkAppTxList: any = { zkapps: [] };
  let getZkAppPending: any = { pooledZkappCommands: [] };
  if (networkConfig.name === ENetworkName.BERKELEY) {
    getZkAppTxList = gql(networkConfig.gqlTxUrl, getZkAppTransactionListBody(), { ...
    options, address });
    getZkAppPending = gql(networkConfig.gqlUrl, getPendingZkAppTxBody(), { ...options
    , address });
  }
  const [{ pooledUserCommands }, { transactions }, { zkapps }, { pooledZkappCommands
    }] = await Promise.all([
    getPendingTxList,
    getTxList,
    getZkAppTxList,
    getZkAppPending,
  ]);

  ...
}
```

Snippet 4.17: Relevant lines in `getTxHistory()`. The `gql` method will send an HTTP request to a Mina node's GraphQL API endpoint.

Impact Issuing four HTTP requests instead of two will unnecessarily increase the server load of the Mina node and waste bandwidth.

Recommendation If possible, batch the requests to the same URL in the same HTTP request. This may require modifying the GraphQL queries.

Developer Response The developers acknowledged the issue but do not plan to resolve this in the short term.