Asigna Audit

December 2023

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the multisig creation scripts for the Asigna project.

During this audit we found 1 critical issue, 1 high issue, 1 medium issue and one minor issue. All issues were resolved or acknowledged.

Scope

The audited files were provided by the client. No github repository or commit hashes were provided.

The scope for this audit includes and is limited to the following files:

- `genAddress.ts`: Multisig creation for the Stacks blockchain
- `generateMultisigAddress.ts`: Multisig creation for the Bitcoin blockchain

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Missing Input Validation On Multisig Threshold	Critical	Resolved
HI-01	Missing Input Validation On Multisig Public Keys	High	Resolved
ME-01	Dangerous Multisig Schemes Allowed	Medium	Acknowledged
MI-01	Unnecessary Revealing of Public Keys	Minor	Acknowledged

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds

of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Missing Input Validation On Multisig Threshold

Location:
- `genAddress.ts:19`
- `generateMultisigAddress.ts:8`

Classification:
- CWE-20: Improper Input Validation[1]

The functions `generateMultisigAddress()` in both scripts use the threshold argument to set the signature threshold of the multisig. A reasonable value for this parameter may be:

```
1<threshold<publicKeys.length
```

In this way, several attacks are avoided, like a threshold over the amount of public keys, or a threshold of zero. But this parameter is never validated (`genAddress.ts`) or validated incorrectly (`generateMultisigAddress.ts`). This might allow the generation of insecure multisig addresses.

Recommendation

Validate the `threshold` parameter in both functions.

[1]https://cwe.mitre.org/data/definitions/20.html

Status

Resolved. Added additional checks on code.

High Severity Issues

HI-01 Missing Input Validation On Multisig Public Keys

Location:

- `genAddress.ts:19`
- `generateMultisigAddress.ts:8`

Classification:

- CWE-20: Improper Input Validation[2]

The functions `generateMultisigAddress()` in both scripts receive a list of public keys for the creation of the multisig address. The functions must validate that there is at least one public key in the array, but both fail to do any range checking on this parameter.

Note: `stacks.js` library provides some checks (validates that `publicKeys.length>0`) but it's not recommended to rely on lower-level functions for input validation.

Recommendation

Validate that `publicKeys.length>1` in both functions.

Status

Resolved. Added additional checks on code.

Medium Severity Issues

ME-01 Dangerous Multisig Schemes Allowed

Location:

- `genAddress.ts:19`
- `generateMultisigAddress.ts:8`

Classification:

- CWE-284:Improper Access Control[3]

In this scheme of k-of-n multisig, the following schemes are allowed:

[2] https://cwe.mitre.org/data/definitions/20.html
[3] https://cwe.mitre.org/data/definitions/284.html

1-of-n: Only one signature is needed: This is dangerous because it centralizes control to any multisig participant.

n-of-n: All signatures are needed: This is dangerous because the loss of a single key invalidates the multisig.

1-of-1: The multisig is a regular signature.

Recommendation

Unless there are valid use-cases for those special cases, validate the threshold and public key values so the dangerous cases are avoided.

Status

Acknowledged. The application must allow this kind of multisig by design. Nevertheless, a warning was added about all particular cases.

Minor Severity Issues

MI-01 Unnecessary Revealing of Public Keys

Location:

- `generateMultisigAddress.ts:14`

Classification:

- CWE-200:Exposure of Sensitive Information to an Unauthorized Actor[4]

The `generateMultisigAddress()` function creates the Multisig verification tapscript based on the signature threshold and public keys. While the variable `leafScriptAsm` is a correct k-of-n `CHECKSIGADD` taproot script, it will unnecessarily reveal the unused public keys of the multisig when the tapscript is spent[5].

Recommendation

Consider replacing it with several k-of-k `CHECKSIGADD` scripts if k and n are small values. This will not reveal any public key that does not participate in the transaction and additionally it is a cheaper transaction.

Status

Acknowledged. Implementation of k-of-k would result in UX problems that would negate the benefits.

[4]https://cwe.mitre.org/data/definitions/200.html
[5] https://github.com/bitcoinops/taproot-workshop/blob/master/2.3-tapscript.ipynb

Changelog

- 2023-12-22 – Initial report.
- 2023-12-29 – Fixes checked