Defense By Thesis

Security Audit Report

Coffer Network

Coffer App API

Initial Report // December 16, 2024
Final Report // January 16, 2025

Team Members

Jehad Baeth // Senior Security Auditor
Mukesh Jaiswal // Senior Security Auditor

Table of Contents

About Thesis Defense

Defense is the security auditing arm of Thesis, Inc., the venture studio behind tBTC, Fold, Mezo, Acre, Taho, Etcher, and Embody. At Defense, we fight for the integrity and empowerment of the individual by strengthening the security of emerging technologies to promote a decentralized future and user freedom. Defense is the leading Bitcoin applied cryptography and security auditing firm. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos SDK, NEAR and more. We offer our services within a variety of technologies including smart contracts, bridges, cryptography, node implementations, wallets and browser extensions, and dApps.

Defense will employ the Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

Section 1.0
Scope

Technical Scope

- **Repository:** https://github.com/coffer-network/coffer-app-api/tree/defense-audit
- **Audit Commit:** 1cedf8efbd9142906beda229cd43e4d3caa14be2
- **Verficiation Commit:** f07a67e82323df286e25fba1de543dda220ee788
- **Files in Scope:**
- pkg/bitcoin/
    - address.go
    - constants.go
    - fee.go
    - network.go
    - psbt.go
    - rpcclient.go
    - runes.go
    - utils.go
- pkg/wallet/
    - ledger/ledger.go
    - wallet.go

Section 2.0
Executive Summary

Schedule

This security audit was conducted from December 9, 2024 to December 16, 2024 by 2 security auditors for a total of 2 person-weeks.

Overview

Coffer Network is a decentralized platform that provides a programmable Smart Account infrastructure for Bitcoin that is intended to manage and stack Bitcoin efficiently. During this review, our team audited the implementation of the Coffer App APIs, implemented in Go.

Threat Model

We determined relevant areas of concern and attack vectors to guide our auditing investigation. We investigated several vectors of attack that a malicious actor could attempt including:

- Creating a valid script with invalid input, eg. A PSBT with a Timelock
- Man-in-the-Middle(MITM) attack
- Exploitation of insufficiently secure cryptography

We looked for implementation errors that could lead to security vulnerabilities. We checked the correctness of the calculation of transaction fees. We also checked that transaction data in general, and the inputs and outputs of PSBTs specifically, are sufficiently validated. We used automated analysis tools including `Gosec`, `staticheck`, `golangci-lint`.

Secure Implementation

In our audit we identified several areas of improvement. We found instances of incorrect implementation. We also found that the implementation lacks appropriate error handling.

We identified a pattern of insufficient validation in the handling of PSBTs. We also found an instance of implemented functionality could be replaced by a battle tested library implementation, improving the code.

There is code that is unused, and unreachable, in addition to untested.

Use of Dependencies

We did not identify any issues in the use of dependencies.

Tests

There are some tests implemented, however, given the limited scope of the audit, we were not able to run or evaluate these tests.

Project Documentation

The implementation does not contain any code comments, and there was no project documentation available for this audit. We recommend implementing comprehensive code comments that describe the intended behavior of each function. We also recommend creating comprehensive project documentation that users, developers, and auditors can use as a resource to learn about the application efficiently.

Section 3.0
Key Findings Table

Issues	Severity	Status
ISSUE #1 FakeP2wshWitness Value Added by Default to the Witness Array	Medium	Not Fixed
ISSUE #2 Insufficient Validation for Public Keys	Low	Fixed
ISSUE #3 The `getTxVSize` and `EstimatedGasFee` Functions Return an Incorrect Value of `vbytes`	Low	Fixed
ISSUE #4 Incorrect Use of Equality in For Loop in `GetP2wshVSize` and `GetP2trVSize` Functions	Low	Fixed
ISSUE #5 Use of SHA1 for Unique Identifier Generation	Low	Not Fixed
ISSUE #6 Insufficient Signature Validation in Bitcoin PSBT Processing	Low	Not Fixed
ISSUE #7 Insufficient PSBT Transaction Input and Output Validation	Low	Not Fixed
ISSUE #8 Lack of Size Validation for Hex-Encoded Taproot Script Components	Low	Not Fixed
ISSUE #9 Lack of Consistency Check Between Signed and Unsigned PSBTs	Low	Not Fixed
ISSUE #10 Incorrect Loop Condition Leading to Inflated Gas Fee Estimation	Low	Fixed
ISSUE #11 No Check for `pubkeyhash` Length in `PayToPubKeyHashScript` and `PayToWitnessPubKeyHashScript` Functions	None	Fixed
ISSUE #12 Silent Failure in HD Path Parsing Function	None	Not Fixed

Severity definitions can be found in Appendix A

Section 4.0
Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

FakeP2wshWitness Value Added by Default to the Witness Array

Medium Not Fixed

Location

/pkg/bitcoin/psbt.go#L343

Description

When calculating the gas fee, the witness value is updated by `FakeP2wshWitness` by default based on the required signatures. However, it does not consider the specific type of script used for the witness. The size of the witness can vary depending on the associated script.

For example, in the case of a P2TR (Pay-to-Taproot) script path spend, the witness size depends on several factors:

- The size of the leaf script
- The number of script inputs
- The depth of the leaf script in the script tree

For P2WPKH (Pay-to-Witness-PubKey-Hash), the witness size depends on the signature and the public key. As a result, the witness length may exceed the expected total size of witness, causing the estimated virtual size to be greater than the actual size.

Impact

The overestimated transaction size would result in increased transaction fees.

Recommendation

We recommend updating witness values based on the specific script they are associated with.

Verification Status

The Coffer team stated that they use 'FakeP2wshWitness' because multi-sig address generated from different types of addresses (p2phk, p2sh-p2wpkh, p2wpkh, p2tr) will result in a different length of witness for each type. As such the longest witness is selected to occupy the position.

ISSUE#2

Insufficient Validation for Public Keys

Low Fixed

Location

/pkg/bitcoin/address.go#L85

Description

The `checkPubKeys` function validates whether the number of public keys and the number of signers are within the permissible limits. However, the function lacks a check to verify whether the public keys are in compressed or uncompressed format.

In P2TR (Pay-to-Taproot) function implementation, all the public keys are mapped into an `xonly` format, which ensures that all the keys are in compressed format. However, the implementation of the P2WSH function is missing this check.

Impact

Accoding to BIP143, only compressed public keys are accepted in P2WPKH (Pay-to-Witness-PubKey-Hash) and P2WSH (Pay-to-Witness-Script-Hash). Each public key passed to a sigop inside version 0 witness program must be a compressed key: the first byte MUST be either 0x02 or 0x03, and the size MUST be 33 bytes. Transactions that break this rule will not be relayed or mined by default.

Using any other format such as uncompressed public key may lead to irrevocable fund loss.

Recommendation

We recommend that a check be added which only allows compressed keys.

ISSUE#3

The `getTxVSize` and `EstimatedGasFee` Functions Return an Incorrect Value of `vbytes`

Low Fixed

Location

pkg/bitcoin/fee.go#L106

pkg/bitcoin/psbt.go#L381-L383

Description

Virtual bytes are used to compare the fee rates between transactions. The virtual size of a transaction is determined by dividing its weight by 4.

In this function, the virtual size is calculated as a precise fraction, but the `math.Ceil` function operation is applied to round it up, yielding a rounded integer value.

Impact

The virtual size is a fractional value when computed accurately, but the `getTxVSize` function returns a rounded integer value. This discrepancy leads to a difference between the original value and the returned one, causing the user to pay an incorrect fee.

Recommendation

To obtain the correct value and prevent precision loss, the following methods can be used:

Multiply the input by 10^N (where N represents the desired number of decimal places), round the result, and then divide it by 10^N.

Use external libraries such as shopspring/decimal for more precise calculations.

ISSUE#4

Incorrect Use of Equality in For Loop in `GetP2wshVSize` and `GetP2trVSize` Functions

Low Fixed

Location

pkg/bitcoin/fee.go#L118

/pkg/bitcoin/fee.go#L134

Description

The functions `GetP2wshVSize` and `GetP2trVSize` update the witnesses based on the number of `mRequired`. However, during the iteration over the `mRequired` value, it uses an incorrect equality check, which results in the addition of extra value to the witnesses.

```
for i := 0; i <= mRequired; i++
   { witnesses = append(witnesses, FakeP2TRWitness) }
```

Impact

The inclusion of extra values in the witnesses increases the total witness length, which will be different from the original witness length and will increase the transaction size and cost.

Recommendation

We recommend updating the equality condition.

```
for i := 0; i < mRequired; i++
   { witnesses = append(witnesses, FakeP2TRWitness) }
```

ISSUE#5

Use of SHA1 for Unique Identifier Generation

Low Not Fixed

Location

/pkg/bitcoin/address.go#L255-L257

Description

The Coffer App API codebase utilizes the SHA1 hashing algorithm to generate a unique identifier from a list of addresses and an address type. The function `GetHashFromAddresses` concatenates the addresses

and address type, then applies SHA1 to produce a hash. This hash is subsequently used in the `MultisigWalletByUnique` function to query the existence of a wallet in the database.

While there is currently no known security impact associated with this implementation, the use of SHA1 is generally discouraged due to its susceptibility to collision attacks. Although the specific context of this function is outside the audit scope, it is important to consider the potential risks associated with using a deprecated hashing algorithm in future developments or changes.

Impact

While no immediate security issues have been identified, relying on a hashing algorithm that is known to be weak could lead to vulnerabilities down the line, especially if the application evolves to handle more sensitive data or if it is integrated with other systems.

Recommendation

Update the `GetHashFromAddresses` function to use SHA256 or SHA3 instead of SHA1. If there is a need to maintain domain separation, consider using domain separation tags instead of deprecated hash functions.

ISSUE#6

Insufficient Signature Validation in Bitcoin PSBT Processing

Low Not Fixed

Location

/pkg/wallet/ledger/ledger.go#L64

/pkg/bitcoin/psbt.go#L386

Description

In the current Bitcoin transaction processing implementation, the signature validation for PSBT lacks comprehensive verification against the secp256k1 elliptic curve cryptography standard. The implementation potentially allows processing of PSBT inputs without rigorous cryptographic signature validation, which could lead to the acceptance of malformed or unauthorized transaction signatures. Specifically, the verification process does not comprehensively ensure the signature is generated using the secp256k1 curve parameters or check the signature's integrity and authenticity, before propagating the transaction to Bitcoin.

Impact

Inadequate signature validation could compromise the integrity and non-repudiation principles of Bitcoin transactions, such a transaction would be rejected wasting gas.

Recommendation

We recommend implementing a robust signature validation mechanism using the `btcutil` and `btcd` libraries. We also recommend leveraging Schnorr signature verification

Verification Status

The Coffer team stated that this check is verified by the Bitcoin node.

ISSUE#7

Insufficient PSBT Transaction Input and Output Validation

Low Not Fixed

Location

/pkg/bitcoin/psbt.go#L386

Description

The current implementation of the PSBT validation process lacks comprehensive checks that are essential for preventing potential financial exploits. Specifically, the system fails to:

- Comprehensively verify that total input amounts are greater than or equal to total output amounts
- Prevent the creation of dust outputs that could potentially be used to manipulate transaction economics

Impact

The identified missing validations could potentially lead to:

- Financial loss through crafted transactions that bypass input-output amount verification
- Creation of dust outputs that could be used in potential denial-of-service or economic manipulation attacks
- Increased attack surface for malicious actors attempting to exploit transaction validation weaknesses

Recommendation

We recommend implementing a robust PSBT validation mechanism while utilizing battle-tested libraries such as btcutil and btcd in order to achieve the following:

- Implement strict input-output amount verification
- Validate input and output count ranges
- Add dust output protection using standardized thresholds
- Implement comprehensive error handling and logging

Verification Status

The Coffer team stated that this check is verified by the Bitcoin node.

ISSUE#8

Lack of Size Validation for Hex-Encoded Taproot Script Components

Low Not Fixed

Location

/pkg/bitcoin/psbt.go#L221-L249

Description

In the `decodeP2trScript` method of the PSBT builder, the implementation does not enforce size constraints on key components of the Taproot script, which could potentially lead to processing

malformed or oversized input data. The vulnerable code segment decodes hex-encoded strings for the following components without performing proper size validation:

- `witnessScript` : No validation against the Bitcoin network's maximum witness script size of 4,000 byte
- `internalKey` : No enforcement of public key size limits. Should be constrained to 33 bytes for compressed keys and 65 bytes for uncompressed keys.
- `tapHash` : No validation of the hash size, which should be consistently 32 bytes (256 bits)
- `controlBlockBytes` : Potential for processing control blocks that may exceed practical network limits

Impact

While the risk is not high, it represents a weakness that could be exploited by malicious actors to disrupt transaction processing through memory exhaustion through large input processing.

Recommendation

We recommend that the `decodeP2trScript` function be modified to include validation that adheres to the bitcoin limits.

ISSUE#9

Lack of Consistency Check Between Signed and Unsigned PSBTs

Low Not Fixed

Location

/pkg/bitcoin/psbt.go#L386

Description

The existing implementation fails to perform thorough consistency checks between unsigned and signed PSBTs. In other words, there is insufficient validation of transaction Input counts and details, and Output count and details during the PSBT signing process.

Without robust consistency verification, there is a risk of undetected modifications to transaction inputs and/or potential manipulation of transaction outputs.

Impact

Unauthorized modification of transaction details that could lead to unintended behavior.

Recommendation

We recommend verifying the integrity of transactions by checking the consistency between a signed transaction and its corresponding raw transaction. Verify that the input counts match, with an "Input count mismatch" error returned if they do not. Next, compare each input using a helper function, with an error provided for any discrepancies.

The output counts should also be checked for consistency, returning an “Output count mismatch” if they differ. Finally, each output’s value and script should be compared, with an error returned for any mismatches.

```
function verifyTransactionConsistency(signedTransaction, rawTransaction):
    // Step 1: Check Input Count
    if inputCount(signedTransaction) != inputCount(rawTransaction):
        return "Input count mismatch"

    // Step 2: Compare Inputs
    for i from 0 to inputCount(signedTransaction) - 1:
        if not compareInputs(signedTransaction.inputs[i], rawTransaction.inputs[i]):
            return "Input mismatch at index " + i

    // Step 3: Check Output Count
    if outputCount(signedTransaction) != outputCount(rawTransaction):
        return "Output count mismatch"

    // Step 4: Compare Outputs
    for i from 0 to outputCount(signedTransaction) - 1:
        if signedTransaction.outputs[i].value != rawTransaction.outputs[i].value or
           signedTransaction.outputs[i].script != rawTransaction.outputs[i].script:
            return "Output mismatch at index " + i

    // Return Success
    return "Transaction consistency verified"
```

Verification Status

The Coffer team stated that this check is verified by the Bitcoin node.

ISSUE#10

Incorrect Loop Condition Leading to Inflated Gas Fee Estimation

Low Fixed

Location

/pkg/bitcoin/psbt.go#L342

Description

The code contains a loop that is responsible for generating witnesses based on the `requiredSignature` parameter. The current implementation uses the following loop condition:

```
for i := 0; i <= requiredSignature; i++
```

This condition results in the creation of `requiredSignature` + 1 witnesses, which is likely more than intended. The extra iteration can lead to inflated gas fee estimations when the function is executed on the blockchain, as the gas fees are calculated based on the number of operations performed.

Impact

The gas fees estimate provided is inflated.

Recommendation

We recommend that the referenced loop condition be modified to ensure that the correct number of witnesses is generated.

ISSUE#11

No Check for `pubkeyhash` Length in `PayToPubKeyHashScript` and `PayToWitnessPubKeyHashScript` Functions

None Fixed

Location

/pkg/bitcoin/utils.go#L11

/pkg/bitcoin/utils.go#L30

Description

The functions `PayToPubKeyHashScript` and `PayToWitnessPubKeyHashScript` build scripts by adding the appropriate opcodes and public key hash value. However, they do not enforce a check on the length of the public key hash. If the public key hash length is not 20 bytes, the script validation will fail when executed.

For example, in the case of P2PKH, the original public key is duplicated using `OP_DUP` and then hashed with `OP_HASH160`. The resulting hashed value is compared with the public key hash in the `ScriptPubKey` and validated using `OP_EQUALVERIFY`. If the public key hash length is incorrect, the comparison will fail, causing the validation to fail since the hashed value will not match.

Impact

If the script validation fails, the transaction will be considered invalid and rejected by the Bitcoin network.

Recommendation

We recommend enforcing the check for `pubkeyhash` length of 20 bytes.

ISSUE#12

Silent Failure in HD Path Parsing Function

None Not Fixed

Location

/pkg/bitcoin/psbt.go#L283-L308

Description

The ParseHDPath function overlooks some potential edge cases that may lead to a silent failuer. For example the malformed HDPAth `m/44'/0'/0'/0/0` will not return an error as it should. Furthermore, the current implemention does not trim whitespaces, which may also lead to a silent failure.

Impact

The inflated gas fees may deter users from executing transactions.

Recommendation

We recommend implementing input validation to verify that the HD path starts with ‘m’, has the correct structure, and contains valid index values. Additionally, the function should be modified to return explicit

error messages for invalid paths instead of failing silently.

We also recommended trimming the leading and trailing whitespace from the input path to prevent silent failures. Lastly, we recommend looking into the parsing logic from the `go-ethereum` library, which could provide improved reliability.

Section 5.0
Appendix A

Severity Rating Definitions

At Defense by Thesis, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Severity Rating Definitions

At Thesis Defense, we utilize the Immunefi Vulnerability Severity Classification System - v2.3.

Severity	Definition

Critical	• Execute arbitrary system commands
	• Retrieve sensitive data/files from a running server, such as:
	  ◦ /etc/shadow
	  ◦ database passwords
	  ◦ blockchain keys (this does not include non-sensitive environment variables, open source code, or usernames)
	• Taking down the application/website
	• Taking down the NFT URI
	• Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as:
	  • Changing registration information
	  • Commenting
	  • Voting
	  • Making trades
	  • Withdrawals, etc.
	• Changing the NFT metadata
	• Subdomain takeover with already-connected wallet interaction
	• Direct theft of user funds Malicious interactions with an already-connected wallet, such as:
	  • Modifying transaction arguments or parameters
	  • Substituting contract addresses
	  • Submitting malicious transactions
	• Direct theft of user NFTs
	• Injection of malicious HTML or XSS through NFT metadata

High	• Injecting/modifying the static content on the target application without JavaScript (persistent), such as:
	  ◦ HTML injection without JavaScript
	  ◦ Replacing existing text with arbitrary text
	  ◦ Arbitrary file uploads, etc.
	• Changing sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as:
	  ◦ Email or password of the victim, etc.
	  ◦ Improperly disclosing confidential user information, such as:
	  ◦ ▪ Email address
	  ▪ Phone number
	  ▪ Physical address, etc.

	▪ PSubdomain takeover without already-connected wallet interaction
Medium	• Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: ◦ Changing the first/last name of user ◦ Enabling/disabling notifications ◦ Injecting/modifying the static content on the target application without JavaScript (reflected), such as: ◦ ▪ Reflected HTML injection ▪ Loading external site data ▪ Redirecting users to malicious websites (open redirect) >
Low	• Changing details of other users (including modifying browser local storage) without already-connected wallet interaction and with significant user interaction, such as: • ◦ Iframing leading to modifying the backend/browser state (must demonstrate impact with PoC) ◦ Taking over broken or expired outgoing links, such as: ◦ ▪ Social media handles, etc ▪ Temporarily disabling user to access target site, such as: ▪ ▪ Locking up the victim from login ▪ Cookie bombing, etc.
None	• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.

Section 6.0
Appendix B

Thesis Defense Disclaimer

Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.