Transaction Verifier

Source Code Security Review

coinspect

flare

Transaction Verifier

Source Code Review

Version: v240516 Prepared for: Flare May 2024

Source Code Review

1. Executive Summary

In March 2024, Flare engaged Coinspect to conduct a security review of the **Transaction Verifier** implementation. The primary goal of this project was to evaluate the security of the script that users will execute before signing transactions. This script parses and interprets a transaction's payload into a human-readable format, enabling users to verify all the transaction parameters before signing. This process is crucial for preventing blind signing and unintended interactions with undesired contracts.

Solved	Caution Advised	Resolution Pending
High: 1	High: 0	High: 0
Medium: 0	Medium: 0	Medium: 0
Low: 0	Low: 0	Low: 0
No Risk: 2	No Risk: 0	No Risk: 0
Total: 3	Total: 0	Total: 0

Coinspect identified one high-risk issue showing how the verifier fails to flag contract calls with zero data, bypassing the contract's address detection.

2. Summary of Findings

2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
FTXV-001	Contract logic executed with zero data is always considered a native transfer	High
FTXV-002	Add warning or error string to notify users a checksum failure	None
FTXV-003	Alert users when calling already known malicious contracts	None

3. Scope

The scope was set to be the repositories:

- Tx Verifier as of commit `0e5983aaec2fdecc6497500b05ce7787d06d38cc`.
- Tx Verifier Lib as of commit `629a989fa7133f61190ab4279b67ba2f480d0329`.

4 Assessment

The transaction verifier library provides the logic and the infrastructure used by the transaction verifier. This library provides a command line interface enabling users to verify their transactions. Users can make the verification process before signing to prevent or mitigate the risks of the following adversarial scenarios when signing:

- Unauthorized tampering with transaction parameters or deceptive attempts related to the message to be signed.
- The absence of a clear and human-readable interface showing the transaction's content and parameters to sign, which would ultimately require a blind signature.
- Uncertainty about the nature and security of the contract being called.

To mitigate these risks, the transaction verifier deconstructs a transaction's payload into key parameters, aiding users in understanding exactly what they are signing. Its architecture supports transaction verification on **EVM**, **P**, and **C** subnet-like chains (e.g., AVAX).

4.1 Security assumptions

The system is a barrier meant to be used prior to a user's signing stage. This system does not take care of any signature made afterwards since it does not implement any wallet-like interface. Additionally, it is assumed that users provide the transaction data they intend to sign, which is not modified later before signing.

4.2 Decentralization

The verifier flags contract calls that are made to foreign contracts (not deployed by Flare). Because of this, the tool requires some sort of maintenance by the Flare Team as they need to provide an updated list of their contract addresses and interfaces.

4.3 Code Quality & Testing

The verifier library repository includes several test-case scenarios using transactions of different types, chains, and destinations, calling a vast array of contracts and functions. The test script was easy to run and modify, allowing Coinspect to validate several scenarios checking the verifier's behavior.

5. Detailed Findings

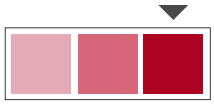
FTXV-001

Contract logic executed with zero data is always considered a native transfer

Status
Solved

Resolution
Fixed

Risk
High



Impact
High

Likelihood
High

Location

`src/evm/index.ts`

Description

The verifier assumes that calls with no data (`0x`) are always a native token transfer on EVMs. Because of this, any call made with zero data to a contract with a `fallback`/`recieve` function that executes further logic will bypass the contract's call detection. This misinterpretation could make unsuspecting users believe that

they are simply making a transfer but, instead, arbitrary logic could be executed from the `fallback`/`recieve` function:

```
function _getType(tx: Transaction): string {
  if (utils.isZeroHex(tx.data)) {
    return txtype.TRANSFER_C;
  } else {
    return txtype.CONTRACT_CALL_C;
  }
}
```

A practical example of a contract within the Flare Ecosystem that executes logic with zero data is `WNat`:

```
/**
    * A proxy for the deposit method.
    */
receive() external payable {
    deposit();
}
```

Then, the system will not try to retrieve any contract data since it has an early return, when the transaction is not a contract call:

```
  if (_getType(tx) !== txtype.CONTRACT_CALL_C) {
    return {};
  }
```

The impact of this issue is considered high, as unsuspecting users might be deceived into believing they are not executing any contract logic when, in fact, they are. Regarding likelihood, the issue occurs spontaneously without requiring any specific conditions other than calling a contract that implements a payable `fallback` or `receive` function with zero data.

Recommendation

Ensure that the call's destination does not have deployed code.

Status

Fixed in commit `10cac15b9816f31c3bf77d1177ab9e9b8bae4615`.

An enhanced mechanism for determining the transaction's type was added to the `_getType()` function. This function now checks whether the destination has deployed code and considers the chain where the transaction originated.

Proof of Concept

A test was conducted using a transaction designed to execute the `receive()` function of the `WNat` contract (located at `0xc67dce33d7a8efa5ffeb961899c73fe01bce9273` on Coston2).

The verifier incorrectly interpreted this as a native transfer rather than identifying it as a contract call, which would execute the `deposit()` function. This misclassification should lead to a failed transaction since it involves executing contract logic, not a native token transfer.

Output

```
Test 1/1: passed
 Input: 0x02f47235850c5115f1008517f551650082520894
 c67dce33d7a8efa5ffeb961899c73fe01bce9273880de0b6b3a764000080c0808080
 Transaction verification: {
  "network": "Flare Testnet Coston2",
  "type": "transferC",
  "description": "Funds transfer on C-chain",
  "recipients": [
    "0xc67dce33d7a8efa5ffeb961899c73fe01bce9273"
  ],
  "values": [
    "1000000000000000000"
  ],
  "fee": "2160900000000000",
  "warnings": [],
  "messageToSign":
"0xf694b4f08049d72fb1aea1e88cada1c40ea4ce99514a0ad9cd6d128ef351c619"
}
```

Transaction

```
  {
    "txHex": "0x02f47235850c5115f1008517f551650082520894

c67dce33d7a8efa5ffeb961899c73fe01bce9273880de0b6b3a764000080c0808080",
    "txVerification": {
      "network": "Flare Testnet Coston2",
      "type": "transferC",
      "description": "Funds transfer on C-chain",
      "recipients": ["0xc67dce33d7a8efa5ffeb961899c73fe01bce9273"],
```

```
      "values": ["100000000000000000"],
      "fee": "2160900000000000",
      "warnings": [],
      "messageToSign":
"0xf694b4f08049d72fb1aea1e88cada1c40ea4ce99514a0ad9cd6d128ef351c619"
    }
  }
```

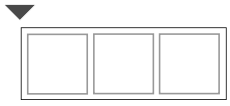
FTXV-002

Add warning or error string to notify users a checksum failure

Status
Solved

Resolution
Fixed

Risk
None



Impact
Recommendation

Likelihood
–

Location

`src/warning.ts`

Description

When verifying an address of a transaction (either recipient or a parameter), the system fails the validation if its checksum is invalid. However, there are no clear error messages or warnings alerting users upon this event.

Therefore, users have to compare the submitted values with the expected ones to identify what triggered the failure. This breaks the secure user experience, degrading the verification functionality.

Recommendation

Add a warning or error string to the validation failure explaining the failure reason.

Status

Fixed on commit `10cac15b9816f31c3bf77d1177ab9e9b8bae4615`.

Address checksum is now included into the system.

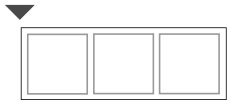
FTXV-003

Alert users when calling already known malicious contracts

Status
Solved

Resolution
Acknowledged

Risk
None



Impact
Recommendation

Likelihood
–

Location

`src/evm/contract/index.ts`

Description

Unsuspecting users may inadvertently interact with a known malicious contract, unknowingly putting their assets at risk.

The verifier retrieves contract data from the explorer when the contract's address is not listed in Flare's Registry. Although it flags that the contract is not part of the Flare Contracts, the process currently lacks a mechanism to alert users that they are engaging with a contract already identified as malicious.

```
export async function getContractData(
    network: number,
    address: string
): Promise<ContractData | null> {
    let data = _getDataFromRegistry(network, address)
```

```
    if (data == null) {
        data = await _getDataFromExplorer(network, address)
    }
    return _toContractData(network, data)
}

async function _getDataFromExplorer(
    network: number,
    address: string
): Promise<AbiContractData | null> {
    return explorer.getContract(network, address)
}
```

Recommendation

Include a mechanism to compare the contract's address with a block-list and raise a warning if users interact with those contracts.

Status

Acknowledged.

6. Disclaimer

The information presented in this document is provided “as is” and without warranty. Security Audits are a “point in time” analysis, and as such, it's possible that something in scope may have changed since the tasks reflected in this report were executed. This report shouldn't be considered a perfect representation of the risks threatening the analyzed systems and/or applications in scope.