Cyberscope

Audit Report

DEFIWAY

October 2024

Files: bridge.tact, outreq.tact, asm.fc, asm.tact, sign.tact, jetton.tact

Audited by © cyberscope

Table of Contents

Overview

The bridge smart contracts of DEFIWAY on the TON network, have undergone a comprehensive audit to address security vulnerabilities, ensure business logic integrity, and optimise performance, providing users with a secure and efficient experience.

The DEFIWAY bridge consists of two smart contracts: bridge.tact and outreq.tact. The outreq.tact contract is responsible for receiving external requests to withdraw amounts from the bridge. These requests have been validated by a group of external signers. The bridge.tact contract then receives these requests and verifies the validity of the signatures against known public keys. The request for withdrawals are then processed. Below, the main functionalities of both contracts are outlined:

Contract: `outreq.tact`

Actions:

Receives an incoming `Send` message and forwards an outgoing `OutInternal` message to the bridge address. Sets:

1. `OutInternal.sender = context().sender`
2. `OutInternal.out = Send.out`
3. `OutInternal.sign = Send.sign`

Incoming messages:

```
message Send {
 out: Out;
 sign: Sign;
}

struct Out {
 queryId: Int as uint64;
 vault: Address;
 recipient: Address;
 amount: Int as coins;
 gas: Int as coins;
}

struct Sign {
 deadline: Int;
 signatures: map<Int as uint8, Cell>;
}
```

Outgoing messages

```
message OutInternal {
 out: Out;
 sign: Sign;
 sender: Address;
}
```

Contract: `bridge.tact`

Actions

Receives `Pay` messages and withholds `Pay.amount`, any excess is returned to the sender.

Receives `OutInternal` messages from the `outreq` contract.

1. Verifies that the address of the sender is the same as that of the `outreq` contract derived for the provided `OutInternal.out.queryId`.
2. Verifies `OutInternal.out.Sender` is an approved `sender`.
3. Verifies that the `OutInternal.sign` includes valid signatures from all approved signers.
4. If `OutInternal.out.vault` is not the zero address, the contract sends `OutInternal.out.gas` to the `vault` along with the remaining balance of the incoming message.
5. If `OutInternal.out.vault` is the zero address, the contract sends to `OutInternal.out.recipient`, tokens equal to `OutInternal.out.amount`. Then sends the remaining balance of the incoming message to `OutInternal.out.sender`. This is the source of the `Send` message received from the `outreq` contract.

Receives `SetSenders` messages.

1. Verifies the list of new senders is signed by all current signers.
2. Sets the new senders as the approved senders recognised by the contract.

Receives `SetSigners` messages.

1. Verifies the list of new signers is signed by all current signers.
2. Sets the new signers as the approved signers recognised by the contract.

Incoming messages

```
message OutInternal {...}
message TokenNotification {...}
message Pay {
 uuid: Int as uint128;
 amount: Int as coins;
}
message SetSenders {
 new_senders: map<Address, Bool>;
 sign: Sign;
}
message SetSigners {
 new_signers: map<Int as uint8, Int as uint256>;
 sign: Sign;}
```

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Testing Deploy	https://testnet.tonscan.org/address/EQDvYkdd87iK2fgCj-IZQ2XkNUn2HjxjIgcm2fK657RU57t8

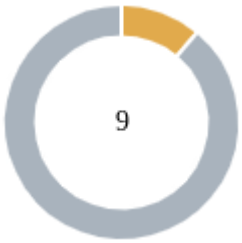
Audit Updates

Initial Audit	08 Oct 2024

Source Files

Filename	SHA256
asm.fc	7f4c6f21c06fc0afa3f4dbbdb691904d01cc65a62fa99169b69fb3d3318ad441
asm.tact	9060c836a2c6427149c085d34e3d0ce277e0a1e4db7abd64fa9dc2e9baf85f2e
bridge.tact	bab40d2052dd250e7eaa1d7e11fa464fe5cb58a1d4badce5805b83d928fdfd56
jetton.tact	c3a71180482965626347ea9e690555cbd8080b04774ffcd0f222b54d6bef08c7
outreq.tact	ffb8f9d341ed5dfbfe4b499bd375f3bf692a4e010bdbc3fa2beb484c31054d5f
sign.tact	673fd8dc61fa289e718acf42545fab625163f35b67d105d47fba567991970d9e

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	8	0	0	2

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	ISF	Inconsistent Storage Fees	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	EFD	Existing Function Declaration	Unresolved
Minor / Informative	MVN	Misleading Variables Naming	Unresolved
Minor / Informative	NBM	Non Bounceable Messages	Unresolved
Minor / Informative	PISF	Potentially Inadequate Storage Fees	Unresolved
Minor / Informative	RAR	Replay-Induced Access Restoration	Unresolved
Minor / Informative	UAW	Unverified Address Workchain	Unresolved
Minor / Informative	UGC	Underestimated Gas Consumption	Unresolved

ISF - Inconsistent Storage Fees

Criticality	Medium
Location	bridge.tact#L48
Status	Unresolved

Description

On TON, fees are paid for both the execution of a smart contract and the storage used. Storage fees are collected from a smart contract's balance during the storage phase of any transaction to cover payments for the account state, including the smart contract's code and data, up to the present time. The storage fee depends on the contract size, specifically the number of cells and the total number of bits within those cells.

It is advised that all fees be deducted from the value of the incoming message. While the contract checks that the provided message value can cover the assumed fees, it fails to reserve the necessary fees from the message's value. Specifically, in this implementation, the contract pays for the storage fees from its own balance during the execution of `receive(msg: OutInternal)`. Over time, this could lead to inconsistencies as storage fees accumulate.

```
nativeReserve(
    msg.out.amount,
    ReserveInvertSign | ReserveAddOriginalBalance |
    ReserveBounceIfActionFail
);
```

```
nativeReserve(
    msg.out.gas,
    ReserveInvertSign | ReserveAddOriginalBalance |
    ReserveBounceIfActionFail
);
```

Recommendation

The team is advised to ensure that the contract reserves the required fees from each incoming message to maintain balance integrity over the long term. This could be implemented by modifying the reserved amount in the call of the `nativeReserve` function.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	sign.tact#L26
Status	Unresolved

Description

The contract's functionality and behavior rely significantly on external parameters or configurations. While this external configuration provides flexibility, it introduces centralization risks that need careful consideration. These risks include a Single Point of Control, increased Vulnerability to Attacks, potential Operational Delays, Trust Dependencies, and the erosion of Decentralization. Specifically, the contracts function as a one-way bridge that does not update its state with records of user deposits. Withdrawal requests are assessed by a group of eligible signers responsible for validating these requests. These signers also have the authority to appoint new eligible signers and senders within the system. If control over the signers' private keys is compromised, it could lead to significant loss of funds.

```
fun verify(body: Cell, sender: Address, sign: Sign) {
 ...
foreach (idx, pubkey in signers) {
require(checkSignature(hash, sign.signatures.get(idx)!!.asSlice(),
pubkey),
"Invalid signature");
...
}
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

EFD - Existing Function Declaration

Criticality	Minor / Informative
Location	asm.tact#L18
Status	Unresolved

Description

The `asm.tact` file defines wrappers for common TVM commands that are not built-in. However, the declaration of `getForwardFee` may be redundant since the static function `getForwardFee` already exists in the latest Tact versions. This redundancy may lead to compiling issues.

```
@name(get_forward_fee)
native getForwardFee(cells: Int, bits: Int, isMc: Bool): Int;
```

Recommendation

To resolve this issue, the team is advised to remove the redundant declaration and rely on the built-in function provided by the latest releases of Tact language.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	bridge.tact#L82
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. Specificcaly, the contract uses the message `TokenTransfer` to notify the vault for an incoming deposit of gas. However the payment is made in nanoTons and not in the form of a jetton as the name of the message and the imported trait implies. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

```
let body = TokenTransfer{
    queryId: msg.out.queryId,
    amount: msg.out.amount,
    destination: msg.out.recipient,
    custom_payload: null,
    response_destination: msg.sender,
    forward_ton_amount: msg.out.gas,
    forward_payload: beginCell().storeUint(0, 1).asSlice(),
}.toCell();
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

NBM - Non Bounceable Messages

Criticality	Minor / Informative
Location	bridge.tact#L42,73,79
Status	Unresolved

Description

The bridge contract forwards funds using `bounce: false`. If an `OutInternal` message specifies the zero address for `msgOutInternal.out.vault`, the contract forwards funds to the recipient with `bounce: false`. This can lead to irretrievable loss of funds if the recipient cannot process the transfer, fails execution, or does not exist.

```
receive(msg: Pay) {
    ...
    send(SendParameters{
        to: ctx.sender,
        value: 0,
        mode: SendRemainingBalance | SendIgnoreErrors,
        bounce: false,
    });
  }
```

```
receive(msg: OutInternal) {
    ...
    send(SendParameters{
        to: msg.out.recipient,
        value: msg.out.amount,
        mode: SendPayGasSeparately | SendBounceIfActionFail,
        bounce: false,
    });
    send(SendParameters{
        to: msg.sender,
        value: 0,
        mode: SendRemainingBalance | SendBounceIfActionFail,
        bounce: false,
    });
}
```

Recommendation

Implementing bouncable messages and handling them elegantly is important. A robust solution involves receiving bounced messages and reversing the entire execution chain from the user's request in the relevant bridge contracts.

PISF - Potentially Inadequate Storage Fees

Criticality	Minor / Informative
Location	bridge.tact#L62,63
Status	Unresolved

Description

The contract calculates the necessary storage fees using the `storage_fees()` function, which wraps the `STORAGEFEES` opcode. This opcode retrieves the value of storage phase fees from the c7 register. However, accurately calculating storage fees requires storing the contract balance from the previous transaction, estimating gas usage, and comparing it to the current balance minus the message value. Storage fees however are dynamic and increase with each second the smart contract is stored on the blockchain. In this context, the contract assumes that the necessary storage fees can be accurately derived from the `storageFee()` call. However, this assumption can be inaccurate, especially if fees have accumulated since the last interaction. This inaccuracy can render statements like the following inadequate or irrelevant, potentially leading to insufficient fee coverage and transaction failures.

```
require(ctx.value > fwdFee * 2 + gasFee + storageFee, "Value too
low");
```

```
require(balance > MIN_FOR_STORAGE + msg.out.amount, "Balance too
low");
```

Recommendation

The team is advised to revise the calculation of the storage fees to ensure proper handling of funds. For more information please refer to the TON Documentation

RAR - Replay-Induced Access Restoration

Criticality	Minor / Informative
Location	sign.tact#L40,46
Status	Unresolved

Description

The contract exhibits a vulnerability to replay attacks, specifically allowing replay-induced privilege reversion. This vulnerability arises from the handling of signatures associated with signer and sender accounts that hold significant privileges. The bridge contract relies on the signatures of existing signers to authorize changes to the list of approved signers and senders. However, an active signer can unilaterally revert the list of signers to a previous state by replaying an old signature. Given that a deviant signer cannot be removed from the list of active signers without the vote of the absolute majority, including themselves, this vulnerability empowers such an entity to revert the system to an outdated state, potentially undermining the integrity and security of the contract's governance.

```
 receive(msg: SetSenders) {
    self.verify(msg.new_senders.asCell()!!, context().sender,
msg.sign);
    self.senders = msg.new_senders;
    self.notify(emptyCell());
  }
```

```
 receive(msg: SetSigners) {
    self.verify(msg.new_signers.asCell()!!, context().sender,
msg.sign);
    self.signers = msg.new_signers;
    self.notify(emptyCell());
  }
```

Recommendation

The team is advised to revise the implementation of the signer and sender authorization mechanism in the bridge contract to prevent the reuse of old signatures for reverting changes. A potential solution involves storing a 32-bit counter `cur-seqno` in the persistent data of the smart contract, and to expect a `req-seqno` value in (the signed part of) any inbound external messages. Then an external message is accepted only if both the signature is valid and `req-seqno` equals `cur-seqno`. After successful processing, the cur-seqno value in the persistent data is increased by one so the same external message will never be accepted again. For more information, please refer to the TON documentation.

UAW - Unverified Address Workchain

Criticality	Minor / Informative
Location	bridge.tact#L21,22,70,85,107, sign.tact#L40
Status	Unresolved

Description

The contract does not verify the workchain of provided addresses. The TON Blockchain consists of one masterchain and up to 2^32 workchains, each with its own rules. Currently, there are 2 workchains on TON, the MasterChain and the BaseChain. The BaseChain is the one used for everyday transactions between actors. Nevertheless, it is advisable to confirm that the provided addresses are on the same workchain as the contract to ensure consistency and security.

```
receive(msg: SetSenders) {
    self.verify(msg.new_senders.asCell()!!, context().sender,
msg.sign);
    self.senders = msg.new_senders;
    self.notify(emptyCell());
  }
```

Recommendation

The team is advised to ensure the contract verifies the workchain of the provided addresses by using the `parse_std_addr` primitive. This will help maintain transactions' integrity and security. For more information, please refer to the TON documentation.

UGC - Underestimated Gas Consumption

Criticality	Minor / Informative
Location	bridge.tact#L07 outreq.tact#L25
Status	Unresolved

Description

The contract needs to accurately estimate gas costs during the computation phase. A reliable approach would be to estimate gas usage through testing and then insert the worst-case scenario in the codebase. Using a function like `GETGASFEE` can then provide an estimation of the computational cost based on the current gas price. In the `outreq` contract, `30,000` gas is allocated for processing the `Send` message. This value appears to be within the necessary bounds. However, the `bridge` contract assumes a consumption of `50,000` gas for processing the `OutInternal` message on the bridge side, which, according to our tests, is significantly underestimated by nearly an order of magnitude. This can lead to inconsistencies, bypassing the necessary gas controls.

```
const OUT_INTERNAL_GAS: Int = 50000;
```

```
const SEND_GAS: Int = 30000;
```

Recommendation

The team is advised to monitor the gas consumption of all messages and computational operations to ensure the proper execution of the code. Accurate monitoring and adjustment of gas estimates are essential to prevent transaction failures and ensure efficient contract operation. For more information, please refer to the TON Documentation.

Summary

The bridge contract of DEFIWAY has been audited for security vulnerabilities, business concerns and overall performance. The audit identified an issue of medium severity and several issues of minor severity affecting the overall consistency of the contract. The team is suggested to take into account these considerations to improve the security and reliability of its application.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io