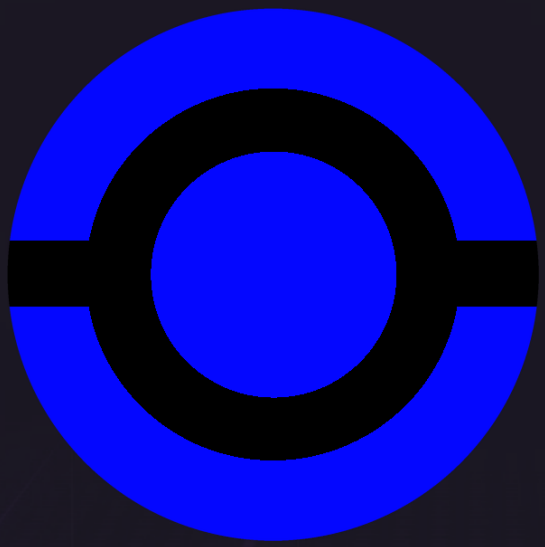
SHERLOCK

Security Review For TRUF.NETWORK by Truflation



Collaborative Audit Prepared For: **TRUF.NETWORK by Truflation**
Lead Security Expert(s): **blockace**
dobrevaleri
Date Audited: **January 14 - January 15, 2026**

Introduction

This audit covers the deposit and withdrawal smart contract on the Truf Network, which manages the movement of tokens between user wallets and the protocol. The review focuses on fund custody, access controls, input validation, and key vulnerability vectors including reentrancy, unauthorized withdrawals, and improper state management, with the goal of ensuring the contract operates securely and safeguards user assets.

Scope

Repository: trufnetwork/bridge-contracts

Audited Commit: d1fd61414817f3ea443d10d449bfed3c156911ff

Final Commit: 69925b1fa519248b864f4da93afcbd14cc2fc47b

Files:

- script/deploy.s.sol
- script/upgrade.s.sol
- src/interfaces/ITrufNetworkBridge.sol
- src/TrufNetworkBridge.sol

Final Commit Hash

69925b1fa519248b864f4da93afcbd14cc2fc47b

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	0	2

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: Unencrypted private key storage in environment variables [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-index-fun-bridge-contract-jan-2026/issues/5

Description

Both deployment scripts require private keys to be loaded from environment variables and used directly inside the script, e.g.:

```
## upgrade.s.sol

function run() public {
@>  uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");
    address proxyAddress = vm.envAddress("PROXY_ADDRESS");

    vm.startBroadcast(deployerPrivateKey);

    TrufNetworkBridge implementation = new TrufNetworkBridge();

    TrufNetworkBridge(payable(proxyAddress)).upgradeToAndCall(address(implementatio
    ↪  n), "");

    vm.stopBroadcast();
}


## deploy.s.sol
function run() public {
@>  uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");
    address owner = vm.envAddress("OWNER_ADDRESS");
    address bridgedToken = vm.envAddress("BRIDGED_TOKEN_ADDRESS");


    vm.startBroadcast(deployerPrivateKey);


    TrufNetworkBridge implementation = new TrufNetworkBridge();
    ERC1967Proxy proxy = new ERC1967Proxy(
        address(implementation), abi.encodeCall(TrufNetworkBridge.initialize,
        ↪  (owner, bridgedToken))
    );


    vm.stopBroadcast();
```

```
    // Log deployment addresses
    console.log("Implementation:", address(implementation));
    console.log("Proxy:", address(proxy));
}
```

This pattern stores sensitive credentials in plain text `.env` files, which creates multiple operational security vulnerabilities. Private keys stored this way can be exposed through version control systems (accidental commits), shell history, process environment dumps, log files, CI/CD pipeline outputs, or compromised developer machines.

Recommendation

Safer approach is to avoid embedding keys in scripts and use Foundry's wallet management and keystore support. Recommended pattern:

1. Import keys into an encrypted local keystore (once per machine) using cast:

```
cast wallet import deployerKey --interactive
```

2. Change scripts to use parameterless broadcasting so the signer is supplied by CLI:

```
// before: vm.startBroadcast(deployerPrivateKey);
vm.startBroadcast();
// ...
vm.stopBroadcast();
```

3. Run the script by providing the signer from the keystore.

Discussion

Dev-Doggo

Fixed using the recommendation.

Issue L-2: Payable modifier on withdraw function [RESOLVED]

Source: https://github.com/sherlock-audit/2026-01-index-fun-bridge-contract-jan-2026/issues/6

Description

The `TrufNetworkBridge::withdraw()` function is marked as `payable`, allowing users to send native tokens along with their withdrawal transactions. However, the function does not handle `msg.value` in any way - it only processes ERC20 token withdrawals through the `_bridgedToken.safeTransfer()` call.

```
## TrufNetworkBridge.sol

function withdraw(
    address recipient,
    uint256 amount,
    bytes32 kwilBlockHash,
    bytes32 root,
    bytes32[] calldata proof,
    Signature[] calldata signatures
@>) external payable {
    // ... signature and merkle proof verification ...

    // Only transfers ERC20 tokens, ignoring msg.value
    $._bridgedToken.safeTransfer(recipient, amount);

    emit Withdraw(recipient, amount, kwilBlockHash);
}
```

Any native tokens sent to this function will remain trapped in the contract permanently, as there is no mechanism to withdraw or refund them. Users might accidentally include `msg.value` when calling the function. The contract has no administrative function to rescue accidentally sent native tokens. This creates a permanent loss scenario for users who send native tokens alongside their withdrawal calls.

Recommendation

Remove the `payable` modifier from the `withdraw()` function entirely. This prevents users from sending native tokens in the first place, eliminating the risk of locked funds.

Discussion

Dev-Doggo

Fixed using the recommendation!

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.