Solidity Smart Contract Audit Report for Force Bridge Contracts

Document Properties

Client	Tunnel Vision Labs
Title	Solidity Smart Contract Audit Report
Version (code freeze hash)	`69193f9a6f684bf32742d47052402c094de31c00`
Author	Manmeet Singh Parmar
Auditor	Manmeet Singh Parmar

Overview

Before discussing the list of issues identified during the audit of the commit `69193f9a6f684bf32742d47052402c094de31c00`, which included all Solidity contracts and libraries in the `eth-contracts` folder, it is important to provide some context on the current status of the project.

Project Status

Force Bridge is a cross-chain protocol that connects the CKB blockchain with other blockchain systems. It is a newly designed bridge that is capable of connecting to any chain that supports multiple signature accounts and non-fungible token transfers. In it's first stage, Force Bridge plans to support EOS, TRON, BTC, Cardano, and Polkadot. It is important to note that trust in the committee running the bridge is required to use it.

Security Assessment

Classification of issues

1. CRITICAL: Bugs leading to Ether or token theft, fund access locking, or any other loss of Ether/tokens to be transferred to any party

2. HIGH: Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement

3. WARNINGS: Bugs that can break the intended contract logic or expose it to DoS attacks.

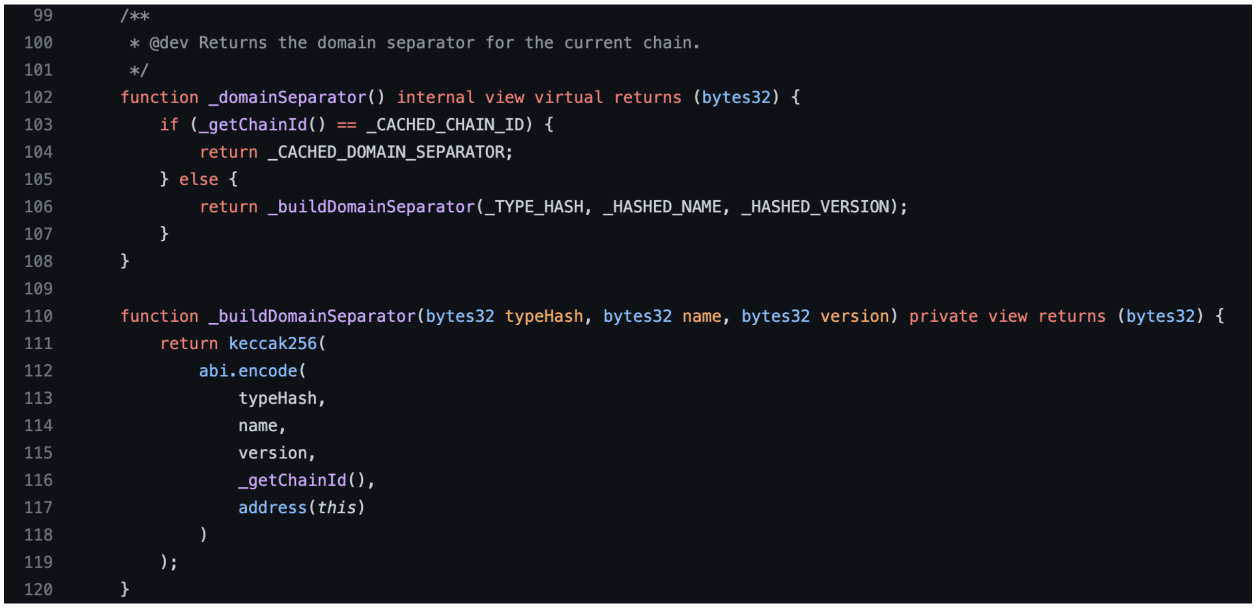
4. COMMENTS: Other issues and recommendations

Detected Issues

CRITICAL

[C01] Delegate call is not supported safely

The ForceBridge contract is based on EIP-712 (Typed structured data hashing and signing), a procedure for hashing and signing of typed structured data as opposed to just bytestrings. A caching system is used instead whereby the domain separator is re-calculated on a need-to basis by comparing the current `chainid` and the one that the original domainSeparator was calculated in, allowing the separator to be re-calculated in case the chain IDs do not match (i.e. due to the PoW chain of Ethereum being used instead of the PoS one).



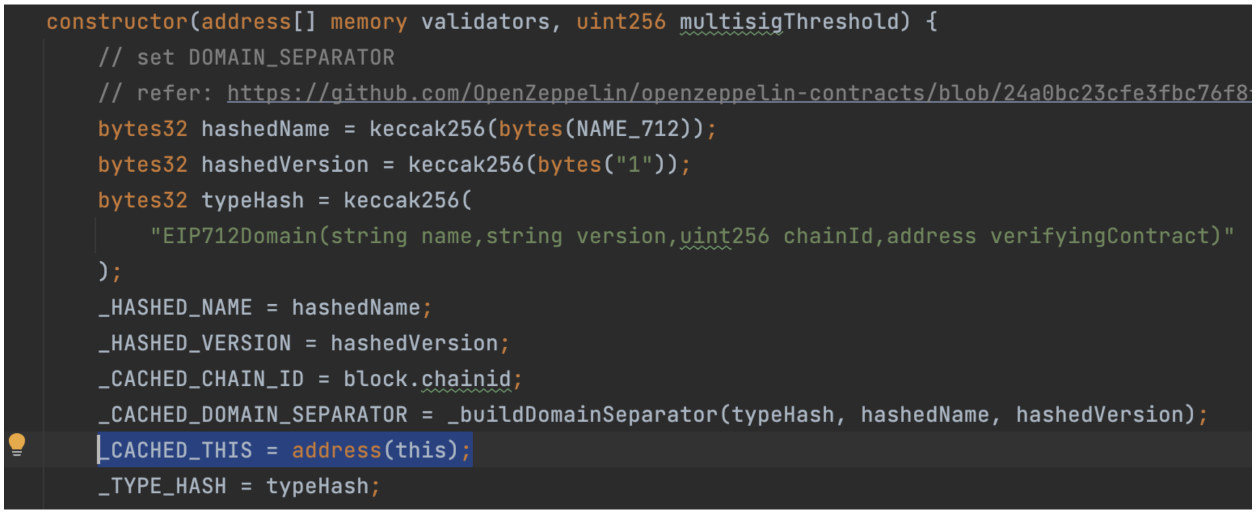
```
    /**
     * @dev Returns the domain separator for the current chain.
     */
    function _domainSeparator() internal view virtual returns (bytes32) {
        if (_getChainId() == _CACHED_CHAIN_ID) {
            return _CACHED_DOMAIN_SEPARATOR;
        } else {
            return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
        }
    }

    function _buildDomainSeparator(bytes32 typeHash, bytes32 name, bytes32 version) private view returns (bytes32) {
        return keccak256(
            abi.encode(
                typeHash,
                name,
                version,
                _getChainId(),
                address(this)
            )
        );
    }
```

Ref : https://github.com/nervosnetwork/force-bridge/blob/main/eth-contracts/contracts/ForceBridge.sol#L102

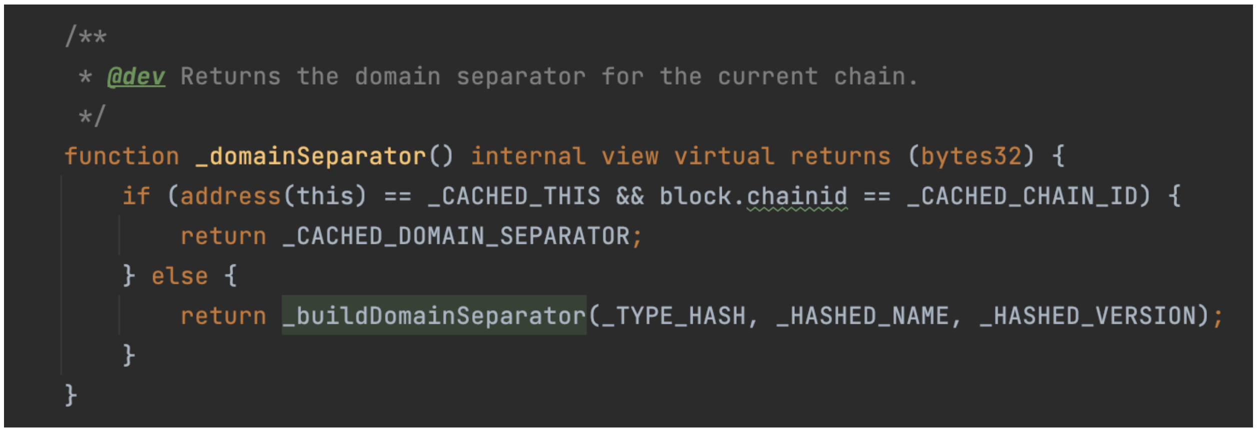
- There should be an additional check along with the chain id check in the `_domainSeparator()` function
- `address(this)` should be cached to an immutable storage to avoid potential issues if a vanilla contract is used in delegatecall context.

- As a cross-chain bridge it is preferred to have this check as extra security to avoid misuse. Misuse could lead to a double spend.
- The `_domainSeparator()` currently has a discrepancy in the `` `this` `` value that is used in each branch of the `if` statement (cached vs recreated).

Recommendation : `address(this)` should be cached to an immutable storage to avoid potential issues if a vanilla contract is used in delegatecall context.



```
constructor(address[] memory validators, uint256 multisigThreshold) {
    // set DOMAIN_SEPARATOR
    // refer: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/24a0bc23cfe3fbc76f8
    bytes32 hashedName = keccak256(bytes(NAME_712));
    bytes32 hashedVersion = keccak256(bytes("1"));
    bytes32 typeHash = keccak256(
        "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
    );
    _HASHED_NAME = hashedName;
    _HASHED_VERSION = hashedVersion;
    _CACHED_CHAIN_ID = block.chainid;
    _CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(typeHash, hashedName, hashedVersion);
    _CACHED_THIS = address(this);
    _TYPE_HASH = typeHash;
```



```
/**
 * @dev Returns the domain separator for the current chain.
 */
function _domainSeparator() internal view virtual returns (bytes32) {
    if (address(this) == _CACHED_THIS && block.chainid == _CACHED_CHAIN_ID) {
        return _CACHED_DOMAIN_SEPARATOR;
    } else {
        return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
    }
}
```

💡 Note : If there is some other contract that makes delegateCalls to our ForceBridge contract, this affects only the delegate-calling contract's storage. In that scenario, it can easy happen that address(this) == _CACHED_THIS is false, but this still does not affect our contracts. It is just a decision we have to make if we care to support delegatecall at our own expense.

HIGH

Not Found

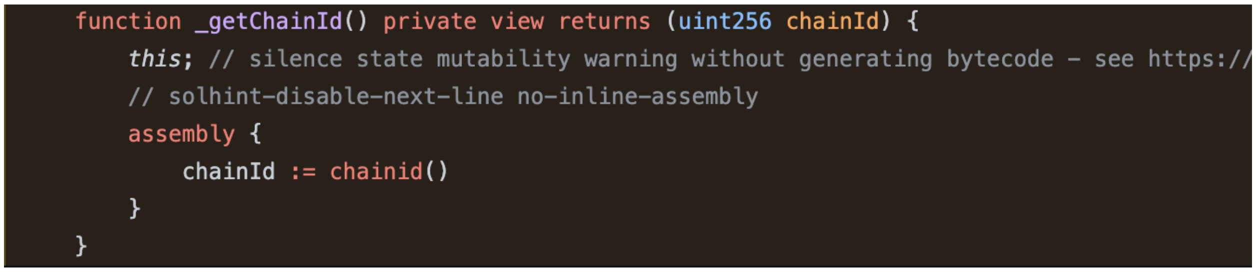
WARNING

Not Found

COMMENTS

[Co01] Use `block.chainid` in place of fetching chain id using inline ASM

`block.chainid` is now a special variable in solidity's global namespace. This can be used in place of



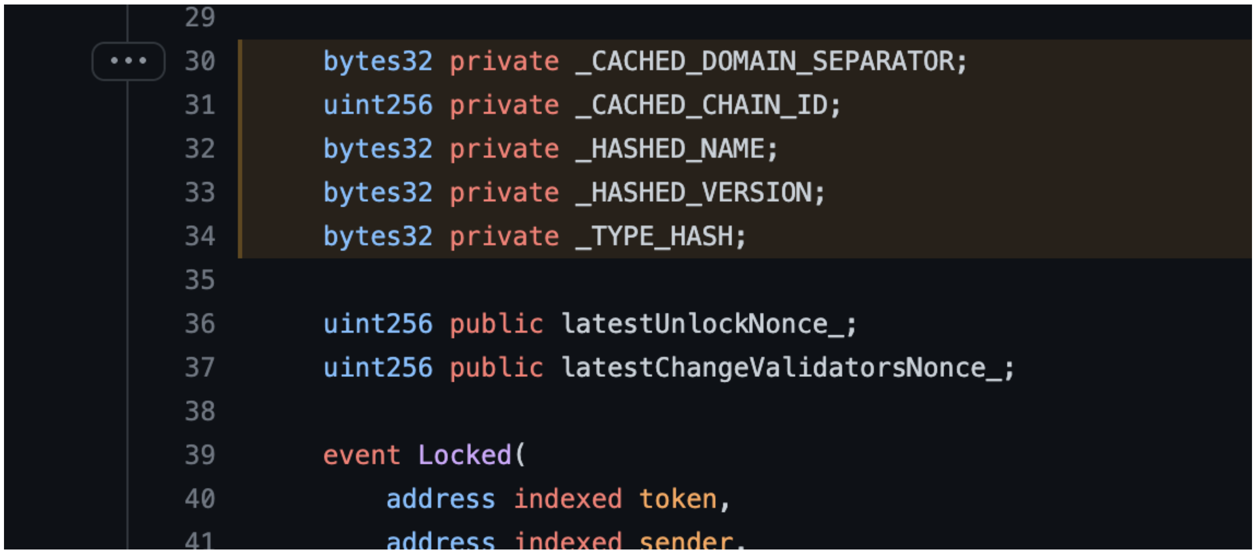
```
    function _getChainId() private view returns (uint256 chainId) {
        this; // silence state mutability warning without generating bytecode - see https://
        // solhint-disable-next-line no-inline-assembly
        assembly {
            chainId := chainid()
        }
    }
```

Ref : https://github.com/nervosnetwork/force-bridge/blob/main/eth-contracts/contracts/ForceBridge.sol#L122-L128

Recommendation : Use `block.chainid` for fetching chain id.

[Co02] Cached and Hashed Variables can be made immutable

The big advantage of immutables is that reading them is significantly cheaper than reading from regular state variables, since immutables will not be stored in storage, but their values will be directly inserted into the runtime code.

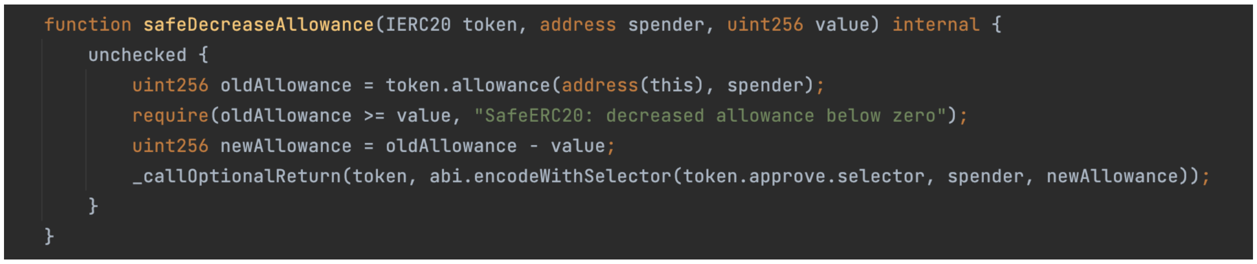


```

    bytes32 private _CACHED_DOMAIN_SEPARATOR;
    uint256 private _CACHED_CHAIN_ID;
    bytes32 private _HASHED_NAME;
    bytes32 private _HASHED_VERSION;
    bytes32 private _TYPE_HASH;

    uint256 public latestUnlockNonce_;
    uint256 public latestChangeValidatorsNonce_;

    event Locked(
        address indexed token,
        address indexed sender,
```

Recommendation : Make these variables `immutable`

[Co03] SafeERC20#safeDecreaseAllowance - enable Safemath revert and use unchecked

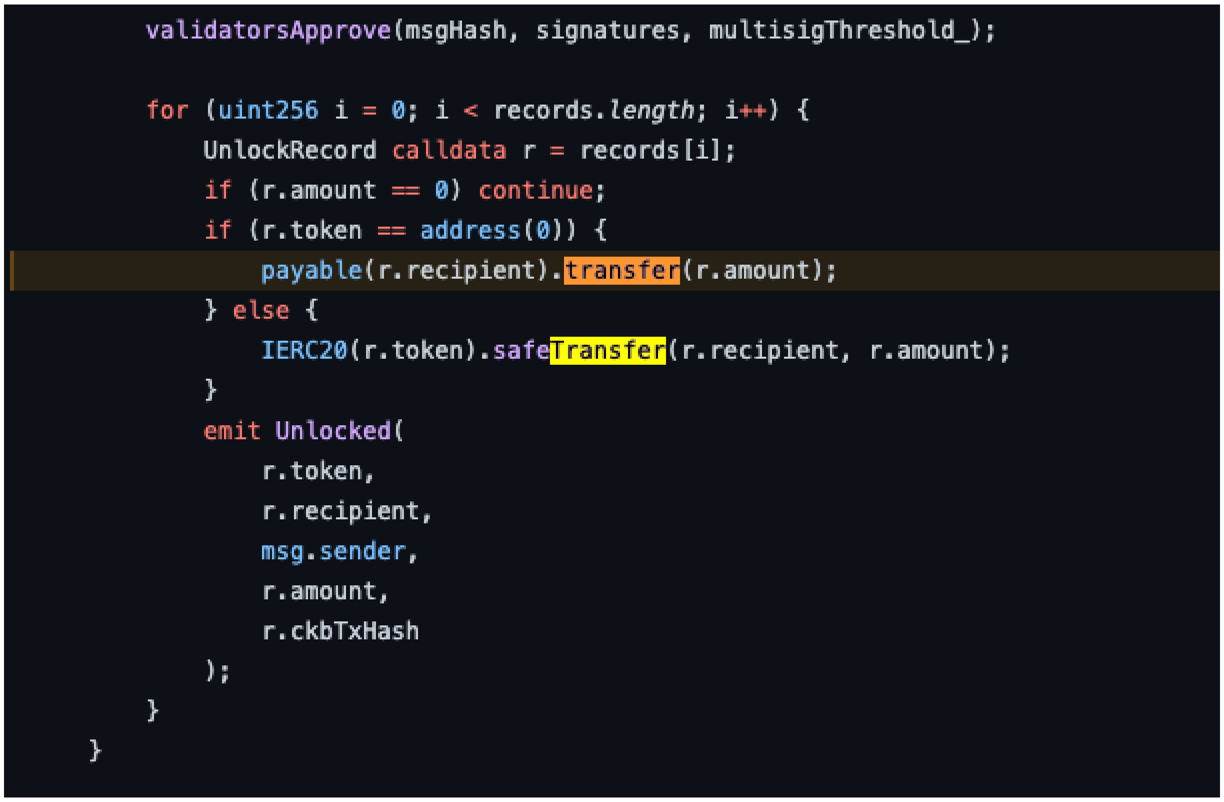
Recommendation : SafeDecreaseAllowance can be implemented the following way



```
function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
    unchecked {
        uint256 oldAllowance = token.allowance(address(this), spender);
        require(oldAllowance >= value, "SafeERC20: decreased allowance below zero");
        uint256 newAllowance = oldAllowance - value;
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }
}
```

[Co04] Solidity’s transfer being used to transfer eth

https://github.com/nervosnetwork/force-bridge/blob/main/eth-contracts/contracts/ForceBridge.sol#L284



```
        validatorsApprove(msgHash, signatures, multisigThreshold_);

        for (uint256 i = 0; i < records.length; i++) {
            UnlockRecord calldata r = records[i];
            if (r.amount == 0) continue;
            if (r.token == address(0)) {
                payable(r.recipient).transfer(r.amount);
            } else {
                IERC20(r.token).safeTransfer(r.recipient, r.amount);
            }
            emit Unlocked(
                r.token,
                r.recipient,
                msg.sender,
                r.amount,
                r.ckbTxHash
            );
        }
    }
```

```
https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
* of certain opcodes, possibly making contracts go over the 2300 gas limit
* imposed by `transfer`, making them unable to receive funds via
* `transfer`. {sendValue} removes this limitation.
```

Suggestion : use Address.sendValue to send eth

Closing Summary

Findings List

Level	Amount
CRITICAL	01 (potential)
HIGH	00
WARNING	00
COMMENTS	03