Biconomy - GasTank

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 2nd, 2021 - November 11th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/10/2021	Ataberk Yavuzer
0.2	Document Edits	11/11/2021	Ataberk Yavuzer
0.3	Final Draft	11/13/2021	Ataberk Yavuzer
0.4	Draft Review	11/16/2021	Gabi Urrutia
1.0	Remediation Plan	11/26/2021	Ataberk Yavuzer
1.1	Remediation Plan Review	11/26/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Biconomy engaged Halborn to conduct a security assessment on their **Biconomy** GasTank Contract beginning on November 2nd and ending on November 11th, 2021.

The security assessment was scoped to the Github repository of Biconomy GasTank Contract. An audit of the security risk and implications regarding the changes introduced by the development team at Biconomy prior to its production release shortly following the assessments deadline.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the Biconomy team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (hardhat, Remix IDE, ganache-cli)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Biconomy GasTank Contracts
   (a) Repository: Biconomy GasTank
   (b) Commit ID: e2bb2f6dfe6fb2737475bbe28b341c316eefa4db
   (c) Contracts in scope:
      i. DappGasTank.sol
      ii. DappGasTankProxy.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	5	2

LIKELIHOOD

IMPACT

		(HAL-01)		
(HAL-03)		(HAL-02)		
	(HAL-05) (HAL-06) (HAL-07)			
(HAL-09)	(HAL-08)	(HAL-04)		

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CONTRACT UPGRADE/INITIALIZATION DROPS MINIMUM DEPOSIT VALUE TO ZERO	Medium	SOLVED - 11/26/2021
(HAL-02) MISSING ROLE-BASED ACCESS CONTROL	Medium	ACKNOWLEDGED
(HAL-03) OWNER CAN RENOUNCE OWNERSHIP	Low	ACKNOWLEDGED
(HAL-04) LACK OF ZERO ADDRESS CHECK	Low	SOLVED - 11/26/2021
(HAL-05) PRAGMA VERSION IS TOO PRIOR	Low	ACKNOWLEDGED
(HAL-06) FLOATING PRAGMA	Low	SOLVED - 11/26/2021
(HAL-07) MISSING REENTRANCY PROTECTION	Low	SOLVED - 11/26/2021
(HAL-08) DEPOSIT FUNCTION DOES NOT CONTROL ALLOWED TOKENS	Informational	NOT APPLICABLE
(HAL-09) UNUSED PRICE ORACLE	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) CONTRACT UPGRADE/INITIALIZATION DROPS MINIMUM DEPOSIT VALUE TO ZERO - MEDIUM

Description:

There is a mapping on the GasTank contract where deposited asset amounts are mapped with addresses. The amount of assets deposited on the contract is also kept on this variable. While defining the minDeposit variable, which controls the amount of assets to be sent to the MasterAccount, this value was determined as 1e18 at the contract itself. However, when this value is checked after the contract is initialized or upgraded, it is seen that minDeposit value is dropped to **0**.

Code Location:

Listing 1: DappGasTank.sol (Lines 58)

```
57 address payable public masterAccount;
58 uint256 public minDeposit = 1e18;
```

Listing 2: DappGasTank.sol

```
function initialize(address trustedForwarder) public initializer {
        __ERC2771Context_init(trustedForwarder);
        __Ownable_init();
        _initializedVersion = 0;
    }
```

PoC Code:

Hardhat Test Script

It is recommended to run the following test script after contract initialization.

Listing 3: scripts/halborn-minDeposit.js

```
async function main() {
    try {

    const owner = ""; // Owner Address
    const proxyAdmin = ""; // Proxy Admin Address
    const relayerMasterAccount = ""; // MasterAccount Address
    const proxyAddress = ""; // Proxy Address

    let gasTankProxy = await hre.ethers.getContractAt("contracts
        /7/gas-manager/gas-tank/DappGasTank.sol:DappGasTank",
        proxyAddress);

    console.log("MinDeposit Value: " + await gasTankProxy.
        minDeposit());

  } catch(error) {
    console.log(error);
  }
}

main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

Output:

```
➜ mexa npx hardhat run scripts/halborn-minDepositTest.js --network ganache
MinDeposit Value: 0
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendations:

It is recommended to define a valid number for minDeposit variable while initializing or upgrading the contract.

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by controlling the minDeposit variable.

Commit ID: **4e2a4ada3f1629b51018dc45c4b9f1af6c2a02c4**

3.2 (HAL-02) MISSING ROLE-BASED ACCESS CONTROL - MEDIUM

Description:

In smart contracts, implementing a correct Access Control policy is an essential step to maintain security and decentralization of permissions on a token. All the features of the smart contract , such as mint/burn tokens and pause contracts are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Nevertheless, other authorization levels are required to follow the principle of least privilege, also known as least authority. Briefly, any process, user or program only can access to the necessary resources or information. Otherwise, the ownership role is useful in a simple system, but more complex projects require the use of more roles by using Role-based access control.

There are multiple important functionalities on DappGasTank.sol, contract such as allowing/disabling tokens for transfer, adjusting minimum deposit amount and withdrawing assets. It is important to divide these functionalities into multiple roles.

Code Location:

Listing 4: Centralized Functions

```
function setMinDeposit(uint256 _newMinDeposit) external onlyOwner
function setTrustedForwarder(address payable _forwarder) external
    onlyOwner
function setMasterAccount(address payable _newAccount) external
    onlyOwner
function setTokenAllowed(address token, bool allowed) external
    onlyOwner
function withdraw(uint256 _amount) public onlyOwner
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendations:

RESOURCE_SETTER role and onlyResourceSetter modifier should be implemented for the following functions to avoid centralization on the contract.

Listing 5: Asset-Related Functions

```
function setMinDeposit(uint256 _newMinDeposit) external onlyOwner
function setTokenAllowed(address token, bool allowed) external
    onlyOwner
function withdraw(uint256 _amount) public onlyOwner
```

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

FINDINGS & TECH DETAILS

3.3 (HAL-03) OWNER CAN RENOUNCE OWNERSHIP - LOW

Description:

Owner of the contract is usually the account which deploys the contract. As a result, the Owner can perform some privileged functions like transferOwnership(). In DappGasTank.sol smart contract, the renounceOwnership function is used to renounce being Owner. Otherwise, if the ownership was not transferred before, the contract will never have an Owner, which is dangerous.

Code Location:

Listing 6: DappGasTank.sol (Lines 55)

```
contract DappGasTank is Initializable, OwnableUpgradeable,
    ERC2771ContextUpgradeable {
```

PoC Code:

Hardhat Test Script

Listing 7: scripts/halborn-renounceOwnership.js

```
async function main() {
    try {

    const owner = ""; // Owner Address
    const proxyAdmin = ""; // ProxyAdmin Address
    const relayerMasterAccount = ""; // MasterAccount Address
    const proxyAddress = ""; // Proxy Address

    let gasTankProxy = await hre.ethers.getContractAt("contracts
        /7/gas-manager/gas-tank/DappGasTank.sol:DappGasTank",
        proxyAddress);

```

```javascript
    console.log("Owner address before renounceOwnership() method: " + await gasTankProxy.owner());
    tx = await gasTankProxy.renounceOwnership();
    console.log("Owner address after renounceOwnership() method: " + await gasTankProxy.owner());

  } catch(error) {
    console.log(error);
  }
}

main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

Output:

```
➜ mexa npx hardhat run scripts/halborn-renounceOwnership.js --network ganache
Owner address before renounceOwnership() method: 0x27D635e6ac78be4d37F93275Fd1299D10703D3a6
Owner address after renounceOwnership() method: 0x0000000000000000000000000000000000000000
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

It is recommended that the Owner cannot call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users.

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.4 (HAL-04) LACK OF ZERO ADDRESS CHECK - LOW

Description:

The DappGasTank.sol contract have multiple input fields on their both public and private functions. Some of these inputs are required as address variable. During the test, it has seen all of these inputs are not protected against using the address(0) as the target address. It is not recommended to use zero address as target addresses on the contracts.

Code Location:

Listing 8: DappGasTank.sol (Lines 79,80)

```
79 function initialize(address trustedForwarder) public initializer {
80         __ERC2771Context_init(trustedForwarder);
81         __Ownable_init();
82         _initializedVersion = 0;
83     }
```

Listing 9: DappGasTank.sol (Lines 152,153)

```
152 function setMasterAccount(address payable _newAccount) external
        onlyOwner{
153         masterAccount = _newAccount;
154         emit MasterAccountChanged(_newAccount, msg.sender);
155     }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendations:

It is recommended to implement additional address check to detect is current contract getting used as a target address.

Listing 10: DappGasTank.sol

```
function initialize(address trustedForwarder) public initializer {
        require(trustedForwarder != address(0), "Trusted Forwarder can not be zero address.")
        __ERC2771Context_init(trustedForwarder);
        __Ownable_init();
        _initializedVersion = 0;
    }
```

Listing 11: DappGasTank.sol

```
function setMasterAccount(address payable _newAccount) external onlyOwner{
        require(_newAccount != address(0), "Master Account can not be zero address.")
        masterAccount = _newAccount;
        emit MasterAccountChanged(_newAccount, msg.sender);
    }
```

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by implementing zero address checks.

Commit ID: **4e2a4ada3f1629b51018dc45c4b9f1af6c2a02c4**

3.5 (HAL-05) PRAGMA VERSION IS TOO PRIOR - LOW

Description:

The project uses one of the latest pragma version (0.8.0) which was released on 16th of December, 2020. The latest pragma version (0.8.9) was released in October 2021. Many pragma versions have been lately released, going from version 0.7.x to the recently released version 0.8.x. in just 6 months.

In the Solitidy Github repository, there is a JSON file where are all bugs finding in the different compiler versions. It should be noted that pragma 0.6.12 and 0.7.6 are widely used by Solidity developers and have been extensively tested in many security audits.

Code Location:

Listing 12: DappGasTank.sol (Lines 1)

```
pragma solidity ^0.8.0;
// SPDX-License-Identifier: MIT
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

If possible, consider using the latest stable pragma version that has been thoroughly tested to prevent potential undiscovered vulnerabilities such as pragma between 0.6.12 - 0.7.6.

References:

- Solidity Releases
- Solidity Bugs By Version

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.6 (HAL-06) FLOATING PRAGMA - LOW

Description:

The project contains many instances of floating pragma. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, either an outdated compiler version that might introduce bugs that affect the contract system negatively or a pragma version too recent which has not been extensively tested.

Code Location:

Listing 13: DappGasTank.sol (Lines 1)

```
pragma solidity ^0.8.0;
// SPDX-License-Identifier: MIT
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

Consider locking the pragma version with known bugs for the compiler version by removing the caret (^) symbol. When possible, do not use floating pragma in the final live deployment. Specifying a fixed compiler version ensures that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by locking the pragma version.

Commit ID: **4e2a4ada3f1629b51018dc45c4b9f1af6c2a02c4**

3.7 (HAL-07) MISSING REENTRANCY PROTECTION - LOW

Description:

To protect against cross-function re-entrancy attacks, it may be necessary to use a mutex. By using this lock, an attacker can no longer exploit the withdrawal function with a recursive call. OpenZeppelin has its own mutex implementation called ReentrancyGuard which provides a modifier to any function called nonReentrant that guards the function with a mutex against re-entrancy attacks.

Code Location:

Listing 14: Missing Re-Entrancy Guard

```
function depositFor(uint256 _fundingKey) public payable
function withdraw(uint256 _amount) public onlyOwner
```

Risk Level:

Likelihood - 2
Impact - 2

FINDINGS & TECH DETAILS

Recommendations:

In the DappGasTank.sol contract, functions above are missing a nonReentrant modifier. It is recommended to add OpenZeppelin ReentrancyGuard library to the project and use the nonReentrant modifier to avoid introducing future re-entrancy vulnerabilities.

Listing 15: nonReentrant Modifier

```
function depositFor(uint256 _fundingKey) public nonReentrant
    payable
function withdraw(uint256 _amount) public onlyOwner nonReentrant
```

Remediation Plan:

SOLVED: The Biconomy Team solved this issue by implementing OpenZeppelin's ReentrancyGuard.

Commit ID: **4e2a4ada3f1629b51018dc45c4b9f1af6c2a02c4**

3.8 (HAL-08) DEPOSIT FUNCTION DOES NOT CONTROL ALLOWED TOKENS - INFORMATIONAL

Description:

During the manual code review step, it has seen that allowedToken variable has implemented to the contract to control which token will be allowed to be deposited to the contract. There is also setTokenAllowed method on the contract to enable or disable tokens. Although, the depositFor() method does not check the value of allowedToken variable to decide which token is allowed to be deposited and which is not allowed.

Code Location:

Listing 16: DappGasTank.sol (Lines 69)

```
69 mapping(address => bool) public allowedTokens;
```

Listing 17: DappGasTank.sol (Lines 162)

```
160 function setTokenAllowed(address token, bool allowed) external
      onlyOwner{
161         require(token != address(0), "Token address cannot be 0");
162         allowedTokens[token] = allowed;
163         emit DepositTokenAdded(token,msg.sender);
164     }
```

Listing 18: DappGasTank.sol

```
175 function depositFor(uint256 _fundingKey) public payable {
176         require(msg.sender == tx.origin || msg.sender ==
                _trustedForwarder, "sender must be EOA or trusted
                forwarder");
177         require(msg.value > 0, "No value provided to depositFor.")
                ;
```

```
        require(msg.value >= minDeposit, "Must be grater than minimum deposit for this network");
        masterAccount.transfer(msg.value);
        dappBalances[_fundingKey] = dappBalances[_fundingKey] + msg.value;
        //review
        depositorBalances[msg.sender][_fundingKey] = depositorBalances[msg.sender][_fundingKey] + msg.value;
        emit Deposit(msg.sender, msg.value, _fundingKey);
    }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendations:

It is suggested to check allowed tokens on `depositFor()` method.

Listing 19: DappGasTank.sol (Lines 175,177)

```
175 function depositFor(uint256 _fundingKey, address tokenAddress)
        public payable {
176         require(msg.sender == tx.origin || msg.sender ==
                _trustedForwarder, "sender must be EOA or trusted
                forwarder");
177         require(allowedTokens[tokenAddress], "This token is not
                allowed.");
178         require(msg.value > 0, "No value provided to depositFor.")
                ;
179         require(msg.value >= minDeposit, "Must be grater than
                minimum deposit for this network");
180         masterAccount.transfer(msg.value);
181         dappBalances[_fundingKey] = dappBalances[_fundingKey] +
                msg.value;
182         //review
183         depositorBalances[msg.sender][_fundingKey] =
                depositorBalances[msg.sender][_fundingKey] + msg.value;
184         emit Deposit(msg.sender, msg.value, _fundingKey);
185     }
```

Remediation Plan:

NOT APPLICABLE: This issue is not applicable in the current version. However, this will be applicable for extended `depositFor()` method in a future release.

3.9 (HAL-09) UNUSED PRICE ORACLE - INFORMATIONAL

Description:

During the test, it was determined that a variable on the contract was not used for any purpose, although it was defined on the contract. This situation does not pose any risk in terms of security. But it is important for the readability and applicability of the code.

Code Location:

Listing 20: DappGasTank.sol (Lines 79)

```
78 //Pricefeeds info should you require to calculate Token/ETH
79 mapping(address => address) public tokenPriceFeed;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to review the unused variables, and to delete it from the contract if it will be remained unused in the future.

Remediation Plan:

ACKNOWLEDGED: The Biconomy Team acknowledged this issue.

3.10 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

```
OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#77) shadows:
        - ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

DappGasTank.setMasterAccount(address)._newAccount (contracts/7/gas-manager/gas-tank/DappGasTank.sol#152) lacks a zero-check on :
                - masterAccount = _newAccount (contracts/7/gas-manager/gas-tank/DappGasTank.sol#153)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

ERC2771ContextUpgradeable._msgSender() (contracts/7/gas-manager/gas-tank/DappGasTank.sol#28-37) uses assembly
        - INLINE ASM (contracts/7/gas-manager/gas-tank/DappGasTank.sol#31-33)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#17-19) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
ContextUpgradeable._msgSender() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#23-25) is never used and should be removed
DappGasTank._msgData() (contracts/7/gas-manager/gas-tank/DappGasTank.sol#119-126) is never used and should be removed
ERC2771ContextUpgradeable._msgData() (contracts/7/gas-manager/gas-tank/DappGasTank.sol#39-45) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/7/gas-manager/gas-tank/DappGasTank.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function ERC2771ContextUpgradeable.__ERC2771Context_init(address) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#16-18) is not in mixedCase
Function ERC2771ContextUpgradeable.__ERC2771Context_init_unchained(address) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#20-22) is not in mixedCase
Variable ERC2771ContextUpgradeable._trustedForwarder (contracts/7/gas-manager/gas-tank/DappGasTank.sol#14) is not in mixedCase
Variable ERC2771ContextUpgradeable.__gap (contracts/7/gas-manager/gas-tank/DappGasTank.sol#46) is not in mixedCase
Parameter DappGasTank.setMinDeposit(uint256)._newMinDeposit (contracts/7/gas-manager/gas-tank/DappGasTank.sol#133) is not in mixedCase
Parameter DappGasTank.setTrustedForwarder(address)._forwarder (contracts/7/gas-manager/gas-tank/DappGasTank.sol#143) is not in mixedCase
Parameter DappGasTank.setMasterAccount(address)._newAccount (contracts/7/gas-manager/gas-tank/DappGasTank.sol#152) is not in mixedCase
Parameter DappGasTank.depositFor(uint256)._fundingKey (contracts/7/gas-manager/gas-tank/DappGasTank.sol#175) is not in mixedCase
Parameter DappGasTank.withdraw(uint256)._amount (contracts/7/gas-manager/gas-tank/DappGasTank.sol#203) is not in mixedCase
Variable DappGasTank._initializedVersion (contracts/7/gas-manager/gas-tank/DappGasTank.sol#59) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#28-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#33-35) is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#77) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#17-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#30) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Reentrancy in DappGasTank.depositFor(uint256) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#175-184):
        External calls:
        - masterAccount.transfer(msg.value) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#179)
        State variables written after the call(s):
        - dappBalances[_fundingKey] = dappBalances[_fundingKey] + msg.value (contracts/7/gas-manager/gas-tank/DappGasTank.sol#180)
        - depositorBalances[msg.sender][_fundingKey] = depositorBalances[msg.sender][_fundingKey] + msg.value (contracts/7/gas-manager/gas-tank/DappGasTank.sol#182)
        Event emitted after the call(s):
        - Deposit(msg.sender,msg.value,_fundingKey) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#183)
Reentrancy in DappGasTank.withdraw(uint256) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#203-206):
        External calls:
        - masterAccount.transfer(_amount) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#204)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_amount,masterAccount) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#205)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

ERC2771ContextUpgradeable.__gap (contracts/7/gas-manager/gas-tank/DappGasTank.sol#46) is never used in DappGasTank (contracts/7/gas-manager/gas-tank/DappGasTank.sol#55-208)
DappGasTank.NATIVE (contracts/7/gas-manager/gas-tank/DappGasTank.sol#60) is never used in DappGasTank (contracts/7/gas-manager/gas-tank/DappGasTank.sol#55-208)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

initialize(address) should be declared external:
        - DappGasTank.initialize(address) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#79-83)
depositFor(uint256) should be declared external:
        - DappGasTank.depositFor(uint256) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#175-184)
withdraw(uint256) should be declared external:
        - DappGasTank.withdraw(uint256) (contracts/7/gas-manager/gas-tank/DappGasTank.sol#203-206)
renounceOwnership() should be declared external:
        - OwnableUpgradeable.renounceOwnership() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#59-61)
transferOwnership(address) should be declared external:
        - OwnableUpgradeable.transferOwnership(address) (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#67-70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
contracts/7/gas-manager/gas-tank/DappGasTank.sol analyzed (5 contracts with 75 detectors), 38 result(s) found
```

Possible Findings and Results:

According to these test results, **some of the findings found by Slither were considered as false positives while some findings were real security concerns.** All relevant findings were reviewed by the auditors and relevant findings addressed in the report as security concerns.

Lack of Zero Address Check This issue has been declared as **valid** vulnerability. During the manual code review step, a lack of zero address check vulnerability was also detected. That issue has addressed in the report.

Reference below:
(HAL-04) LACK OF ZERO ADDRESS CHECK

Reentrancy Vulnerability This vulnerability has been declared as **False-Positive** since it is not possible to trigger the Reentrancy vulnerability.

THANK YOU FOR CHOOSING

HALBORN