Competitive Security Assessment

HashKey

Nov 12th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

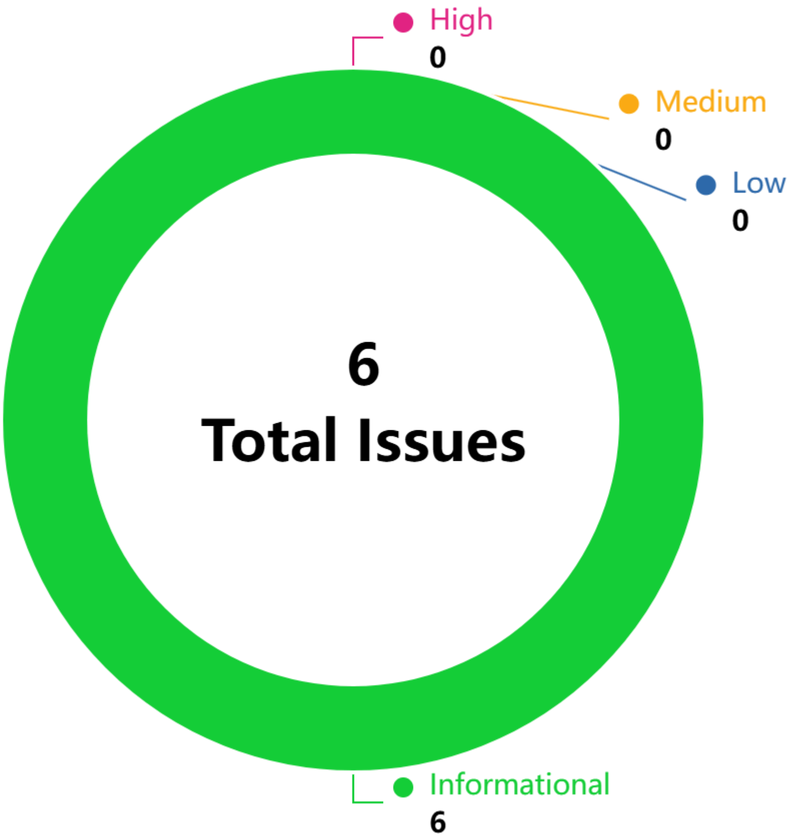
Overview

Project Name	HashKey
Language	solidity
Codebase	- https://etherscan.io/token/0x557683a5fa469d00516dee63fbf345c450cf647a#code - https://etherscan.io/address/0xE7C6BF469e97eEB0bFB74C8dbFF5BD47D4C1C98a#code

Audit Scope

File	SHA256 Hash
HSKImplementationV1/contracts/ERC20.sol	c329fbf2c4a7ee50137f0c0a6e66eea6d66d60c638a4dcde9bd97e96a7efff09
HSKProxy/contracts/Proxy.sol	520654d490e6a4900676bd1f927c63f7e4cc150252db77dedd0a92cf234b792c
HSKImplementationV1/contracts/BurnPermit.sol	ea9e150198c385d60c8b10ef8cfda9867037474f6dc1a704c9a86cc935adc199
HSKImplementationV1/contracts/library/Vesting.sol	1610326cef7f95da2bbde82d8f675b7157439632ef7c9d3d9e4cf15bddf888b5
HSKImplementationV1/contracts/ERC20Permit.sol	be0d5c24ecccee42672e56d99b96d426c75ab54db7e5f93d3cb49e2e344c6ec0
HSKImplementationV1/contracts/library/CheckSig.sol	f0eec8ea55b30faeef6b704b1db037abdfd0e514ad98ead10b870b00e32bb5f1
HSKImplementationV1/contracts/Access.sol	4020d6bd12b85cfa8cb7257fe2c71def243f3e668bcad1bf2b2809f7d72c4e72
HSKImplementationV1/contracts/BlackList.sol	1e0e1da075ffb10f993aceb53c9e2e0e54a9b26f348f488d3e51c8ec761c42d5
HSKImplementationV1/contracts/HSK.sol	744568f6ef8f9d3fa8cd0f1fd5bb2a68952da4ee693a2dd560eeb94799c5298d
HSKImplementationV1/contracts/library/Mint.sol	a134cd3094a7e77a3535c5e4f395dfd4db5040566f8942bcc40f06ff973c8f5c

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
HKY-1	Unused Imports	Gas Optimization	Informational	Acknowledged	***
HKY-2	Unnecessary Checked in Loop	Gas Optimization	Informational	Acknowledged	***
HKY-3	State Variable Could be Cached in Memory	Gas Optimization	Informational	Acknowledged	***
HKY-4	Compiler Version Optimization	Logical	Informational	Acknowledged	***
HKY-5	Cheaper Conditional Operators	Language Specific	Informational	Acknowledged	***
HKY-6	Avoid Re-storing Same Values	Code Style	Informational	Acknowledged	***

HKY-1:Unused Imports

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	***

Code Reference

- code/HSKImplementationV1/contracts/BurnPermit.sol#L5

```
5: import "@openzeppelin/contracts/access/Ownable.sol";
```

Description

***: In the contract `BurnPermit.sol`, where was found to be importing the file `@openzeppelin/contracts/access/Ownable.sol` which is not used anywhere in the code:

```
import "@openzeppelin/contracts/access/Ownable.sol";
```

And Solidity is a Gas-constrained language. Having unused code or import statements incurs extra gas usage when deploying the contract.

Recommendation

***: Remove the unused import statement `@openzeppelin/contracts/access/Ownable.sol` if it's not utilized anywhere in the code to save on deployment gas.

```
- import "@openzeppelin/contracts/access/Ownable.sol";
```

Client Response

client response : Acknowledged.

HKY-2:Unnecessary Checked in Loop

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	***

Code Reference

- code/HSKImplementationV1/contracts/ERC20.sol#L212-L218
- code/HSKImplementationV1/contracts/ERC20.sol#L221-L231

```
212: function batchTransfer(address[] memory tos, uint256[] memory amounts) external {
213:         require(tos.length == amounts.length, "ERC20: Unmatched array length");
214:
215:         for (uint256 i = 0; i < tos.length; i++) {
216:             transfer(tos[i], amounts[i]);
217:         }
218:     }
```

```
221: function batchTransferFrom(
222:         address from,
223:         address[] memory tos,
224:         uint256[] memory amounts
225:     ) external {
226:         require(tos.length == amounts.length, "ERC20: Unmatched array length");
227:
228:         for (uint256 i = 0; i < tos.length; i++) {
229:             transferFrom(from, tos[i], amounts[i]);
230:         }
231:     }
```

Description

*****:** Increments inside a loop could never overflow due to the fact that the transaction will run out of gas before the variable reaches its limits. Therefore, it makes no sense to have checked arithmetic in such a place:

```
   function batchTransfer(address[] memory tos, uint256[] memory amounts) external {
        require(tos.length == amounts.length, "ERC20: Unmatched array length");

>@      for (uint256 i = 0; i < tos.length; i++) {
            transfer(tos[i], amounts[i]);
        }
    }

    /// @dev batch execute transferFrom tokens.
    function batchTransferFrom(
        address from,
        address[] memory tos,
        uint256[] memory amounts
    ) external {
        require(tos.length == amounts.length, "ERC20: Unmatched array length");

>@      for (uint256 i = 0; i < tos.length; i++) {
            transferFrom(from, tos[i], amounts[i]);
        }
    }
```

Recommendation

*****: It is recommended to have the increment value inside the unchecked block to save some gas.

```
function batchTransfer(address[] memory tos, uint256[] memory amounts) external {
        require(tos.length == amounts.length, "ERC20: Unmatched array length");

        for (uint256 i = 0; i < tos.length;) {
            transfer(tos[i], amounts[i]);
            unchecked {i++};
        }
    }

    /// @dev batch execute transferFrom tokens.
    function batchTransferFrom(
        address from,
        address[] memory tos,
        uint256[] memory amounts
    ) external {
        require(tos.length == amounts.length, "ERC20: Unmatched array length");

        for (uint256 i = 0; i < tos.length;) {
            transferFrom(from, tos[i], amounts[i]);
            unchecked {i++};
        }
    }
```

Client Response

client response : Acknowledged.

HKY-3:State Variable Could be Cached in Memory

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	***

Code Reference

- code/HSKImplementationV1/contracts/ERC20.sol#L540-L553

```
540: function _decreaseShare(Mint.Bucket bucket, uint256 amount) internal {
541:         if (bucket == Mint.Bucket.EcoGrowth) {
542:             require(amount <= _ecoGrowthShare, "ERC20: Insufficient share");
543:             _ecoGrowthShare -= amount;
544:         } else if (bucket == Mint.Bucket.Team) {
545:             require(amount <= _teamShare, "ERC20: Insufficient share");
546:             _teamShare -= amount;
547:         } else if (bucket == Mint.Bucket.Reserve) {
548:             require(amount <= _reserveShare, "ERC20: Insufficient share");
549:             _reserveShare -= amount;
550:         } else {
551:             revert("ERC20: Invalid mint bucket");
552:         }
553:     }
```

Description

***: The contract `ERC20.sol` is using the state variables `_ecoGrowthShare`, `_teamShare` and `_reserveShare` multiple times in the function `_decreaseShare`.
`SLOAD` are expensive (2100 gas to 1st access and 100 gas for each subsequent access.) compared to `MLOAD/MSTORE` (3 gas each).

Recommendation

***: Cache storage variables in memory to minimize `SLOAD` operations and reduce gas costs.

```
function _decreaseShare(Mint.Bucket bucket, uint256 amount) internal {
    if (bucket == Mint.Bucket.EcoGrowth) {
        uint256 ecoGrowthShare = _ecoGrowthShare;
        require(amount <= ecoGrowthShare, "ERC20: Insufficient share");
        _ecoGrowthShare = ecoGrowthShare - amount;
    } else if (bucket == Mint.Bucket.Team) {
        uint256 teamShare = _teamShare;
        require(amount <= teamShare, "ERC20: Insufficient share");
        _teamShare = teamShare - amount;
    } else if (bucket == Mint.Bucket.Reserve) {
        uint256 reserveShare = _reserveShare;
        require(amount <= reserveShare, "ERC20: Insufficient share");
        _reserveShare = reserveShare - amount;
    } else {
        revert("ERC20: Invalid mint bucket");
    }
}
```

Client Response

client response : Acknowledged.

HKY-4:Compiler Version Optimization

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	***

Code Reference

- code/HSKImplementationV1/contracts/Access.sol#L2

```
2: pragma solidity ^0.8.0;
```

- code/HSKImplementationV1/contracts/BlackList.sol#L2

```
2: pragma solidity ^0.8.0;
```

- code/HSKImplementationV1/contracts/BurnPermit.sol#L2

```
2: pragma solidity ^0.8.0;
```

- code/HSKImplementationV1/contracts/ERC20.sol#L2

```
2: pragma solidity ^0.8.0;
```

- code/HSKImplementationV1/contracts/ERC20Permit.sol#L2

```
2: pragma solidity ^0.8.0;
```

- code/HSKImplementationV1/contracts/HSK.sol#L2

```
2: pragma solidity ^0.8.0;
```

- code/HSKProxy/contracts/Proxy.sol#L2

```
2: pragma solidity ^0.8.0;
```

Description

*****:** Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the floating pragma, i.e. by not using ^ in pragma solidity ^0.8.0, ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.
For reference, see https://swcregistry.io/docs/SWC-103

Recommendation

*****:** It is recommended to use a recent version of the Solidity compiler and lock the pragma version.

```
pragma solidity 0.8.25;
```

Client Response

client response : Acknowledged.

HKY-5:Cheaper Conditional Operators

Category	Severity	Client Response	Contributor
Language Specific	Informational	Acknowledged	***

Code Reference

- code/HSKProxy/contracts/Proxy.sol#L68-L76

```
68: function _setImplementation(address _impl) internal {
69:         require(_impl.code.length > 0, "Proxy: not a contract address");
70:
71:         bytes32 slot = IMPLEMENTATION_SLOT;
72:         // solhint-disable-next-line no-inline-assembly
73:         assembly {
74:             sstore(slot, _impl)
75:         }
76:     }
```

Description

***: During compilation, `x != 0` is cheaper than `x > 0` for `uint` in solidity inside conditional statements:

```
contract HSKProxy is Proxy, IERC897Proxy {

    // code snippet
    function _setImplementation(address _impl) internal {
>@      require(_impl.code.length > 0, "Proxy: not a contract address");

        bytes32 slot = IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, _impl)
        }
    }

   // code snippet
}
```

Recommendation

***: Use x != 0 instead of x > 0 for unsigned integer checks to optimize gas cost.

```
contract HSKProxy is Proxy, IERC897Proxy {

    // code snippet
    function _setImplementation(address _impl) internal {
+       require(_impl.code.length != 0, "Proxy: not a contract address");

        bytes32 slot = IMPLEMENTATION_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, _impl)
        }
    }

   // code snippet
}
```

Client Response

client response : Acknowledged.

HKY-6:Avoid Re-storing Same Values

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	***

Code Reference

- code/HSKImplementationV1/contracts/BlackList.sol#L25-L28

```
25: function setBlackList(address account, bool blacklisted) external accessible(BLACKLIST_ROLE) {
26:         require(account != address(0), "Blacklist: zero address");
27:         _blacklist[account] = blacklisted;
28:         emit SetupBlackList(account, blacklisted);
```

Description

***: The function `setBlackList` is found to be allowing re-storing the value in the contract's state variable even when the old value is equal to the new value.
This practice results in unnecessary gas consumption due to the `Gsreset` operation (2900 gas), which could be avoided.
If the old value and the new value are the same, not updating the storage would avoid this cost and could instead incur a `Gcoldsload` (2100 gas) or a `Gwarmaccess` (100 gas), potentially saving gas.

```
    function setBlackList(address account, bool blacklisted) external accessible(BLACKLIST_ROLE) {
        require(account != address(0), "Blacklist: zero address");
@>      _blacklist[account] = blacklisted;
        emit SetupBlackList(account, blacklisted);
    }
```

Recommendation

***: Add a condition to compare the old value with the new value. Only update the state variable if the values differ, preventing unnecessary writes and saving gas.

```
function setBlackList(address account, bool blacklisted) external accessible(BLACKLIST_ROLE) {
    require(account != address(0), "Blacklist: zero address");
+   if (_blacklist[account] != blacklisted) {
        _blacklist[account] = blacklisted;
        emit SetupBlackList(account, blacklisted);
    }
}
```

Client Response

client response : Acknowledged.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. 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 Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.