Competitive Security Assessment

HibikiRun

Aug 15th, 2023

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 Detail

Project Name	HibikiRun
Platform & Language	Solidity
Codebase	• https://polygonscan.com/address/0x3e15cd00b456b0fb33827e3c9b49952bb0ec126c • audit commit - https://polygonscan.com/address/0x3e15cd00b456b0fb33827e3c9b49952bb0ec126c • final commit - https://polygonscan.com/address/0x3e15cd00b456b0fb33827e3c9b49952bb0ec126c
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

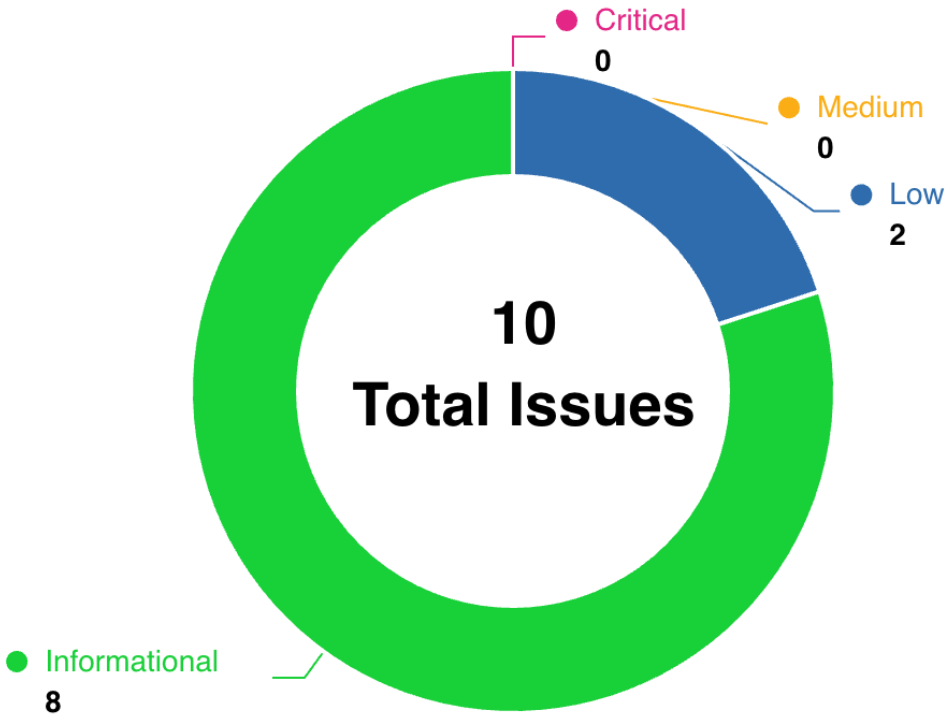
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	2	0	1	0	1	0
Informational	8	0	5	0	0	3

Audit Scope

File	SHA256 Hash
./contract.sol	d60b789d7171f4b06d74989cbbef3ac515a5fcac82e5b1551ac42f3083d06166

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
HIB-1	Privileged Ownership risk in `HUTToken::mint()` function	Privilege Related	Low	Mitigated	helookslikeme, SAir, hunya, 0xxm
HIB-2	Transaction order dependency risk in `approve()` function	Logical	Low	Acknowledged	SAir
HIB-3	Functions not used internally could be marked external in `HUTToken::mint()` function	Gas Optimization	Informational	Acknowledged	hunya

HIB-4	Gas Optimization - Parameters in External Function Should Be Declared as Calldata	Gas Optimization	Informational	Acknowledged	0xxm
HIB-5	Missing "virtual" Modifier in `ERC20Metadata` interface	Code Style	Informational	Acknowledged	SAir
HIB-6	Gas Optimization - Use CustomError Instead of String for `require`	Gas Optimization	Informational	Acknowledged	0xxm
HIB-7	Suggest to add function to withdraw Ether or Token locked in contract	Logical	Informational	Acknowledged	0xxm
HIB-8	Solidity version inconsistency in `contract.sol` smart contract	Language Specific	Informational	Declined	hunya, SAir
HIB-9	Unnamed function parameter in `HUTToken` contract and `HBKToken` contract	Code Style	Informational	Declined	SAir
HIB-10	Lack of marking in `contract.sol` contract	Code Style	Informational	Declined	SAir

HIB-1:Privileged Ownership risk in `HUTToken::mint()` function

Category	Severity	Client Response	Contributor
Privilege Related	Low	Mitigated	helookslikeme, SAir, hunya, 0xxm

Code Reference

- code/contract.sol#L647-L655
- code/contract.sol#L652-L654
- code/contract.sol#L652 code/contract.sol#L653

```
647:contract HUTToken is ERC20, Ownable {
648:    constructor(uint256 initialSupply) ERC20("Hibiki Utility Token", "HUT") {
649:        _mint(msg.sender, initialSupply);
650:    }
651:
652:    function mint(address to, uint256 amount) public onlyOwner {
653:        _mint(to, amount);
654:    }
655:}

652:function mint(address to, uint256 amount) public onlyOwner {
653:        _mint(to, amount);
654:    }

652:function mint(address to, uint256 amount) public onlyOwner {

653:_mint(to, amount);
```

Description

helookslikeme : The administrator can arbitrarily mint, or if the administrator's authority is stolen, the hacker can issue additional tokens arbitrarily
SAir : The `mint()` function can only be called by `Owner`, and `Owner` is free to call the `mint()` function to mint token for any `to` address. Project owners can mint token maliciously to influence the market.
hunya : The owner of contracts is able to mint more tokens casually without obtaining consensus from the community.
0xxm : The total supply of both `HUTToken` and `HBKToken` are not limited, especially the owner of `HUTToken` has the

privilege to mint unlimited token at any time, which results in a centralized risk as the owner is able to inflate the token with large amount of token mint.

```
contract HUTToken is ERC20, Ownable {
    constructor(uint256 initialSupply) ERC20("Hibiki Utility Token", "HUT") {
        _mint(msg.sender, initialSupply);
    }

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
}

contract HBKToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("Hibiki Token", "HBK") {
        _mint(msg.sender, initialSupply);
    }
}
```

Recommendation

helookslikeme : Use multi-signature management
SAir : It is recommended that the `mint()` function be removed.
Consider below fix in `HUTToken` contract:

```
contract HUTToken is ERC20, Ownable {
    uint256 private _initialSupply;

    constructor(uint256 _initialSupply) ERC20("Hibiki Utility Token", "HUT") {
        _mint(msg.sender, _initialSupply);
    }
}
```

hunya : Renounce ownership when it is the right time to do so, or gradually migrate to a timelock plus multisig governing procedure and let the community monitor in respect to transparency considerations.
0xxm : It is recommented to restrict the total supply of tokens, eg. by adpoting openzepplin's ERC20Capped.sol.

Client Response

Mitigated. The contract owner is a multi-sign wallet.

HIB-2:Transaction order dependency risk in `approve()` function

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	SAir

Code Reference

- code/contract.sol#L476-L486

```
476:function _approve(
477:        address owner,
478:        address spender,
479:        uint256 amount
480:    ) internal virtual {
481:        require(owner != address(0), "ERC20: approve from the zero address");
482:        require(spender != address(0), "ERC20: approve to the zero address");
483:
484:        _allowances[owner][spender] = amount;
485:        emit Approval(owner, spender, amount);
486:    }
```

Description

SAir : Since miners always get a gas fee via a code representing an Externally Owned Address (EOA), users can specify a higher fee in order to carry out transactions faster. Since the ethereum blockchain is public, everyone can see the contents of everyone else's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal that solution and copy its transactions at a higher fee to grab the original solution. Detection result: It is detected that there is a risk of transaction order dependency attack in the approve function of the contract. Although the functions increaseApprove and decreaseApprove are used later, there is no checking of _allowances[owner][spender] and AMOUNT in the _approve function, and all the three functions are checked. and amount are not checked in the _approve function, and all three functions are public, there is still a risk of transactional order dependency.

Recommendation

SAir : 1. Front-end restrictions, when user A will modify the amount from N to M, can be modified first from N to 0, and then from 0 to M. 2. Add the following code at the beginning of the approve function:

```
require((amount== 0) || (_allowances[owner][spender] = 0))
```

Client Response

Acknowledged

HIB-3:Functions not used internally could be marked external in `HUTToken::mint()` function

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	hunya

Code Reference

- code/contract.sol#L652

```
652:function mint(address to, uint256 amount) public onlyOwner {
```

Description

hunya : function `mint` has public visibility and it is not used in the contract internally. Best practice is to mark external which is not used internally.

Recommendation

hunya : Change public to external for the `mint` function.

Client Response

Acknowledged. Would not change it as it's not a security risk.

HIB-4:Gas Optimization - Parameters in External Function Should Be Declared as Calldata

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	0xxm

Code Reference

- code/contract.sol#L197

```
197:constructor(string memory name_, string memory symbol_) {
```

Description

0xxm : When the compiler parses the external function, it can directly read the function parameters from calldata. Setting it to other storage locations may waste gas.

Generally, about 300-400 gas can be saved with compiler optimization turned off for each calldata parameter, while 120-150 gas can be saved with compiler optimization turned on.

```
constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
}
```

Recommendation

0xxm : Change the parameter storage to `calldata` instead of `memory`.

Client Response

Acknowledged. Would not change it as it's not a security risk.

HIB-5:Missing "virtual" Modifier in `ERC20Metadata` interface

Category	Severity	Client Response	Contributor
Code Style	Informational	Acknowledged	SAir

Code Reference

- code/contract.sol#L104-L119

```
104:interface IERC20Metadata is IERC20 {
105:    /**
106:     * @dev Returns the name of the token.
107:     */
108:    function name() external view returns (string memory);
109:
110:    /**
111:     * @dev Returns the symbol of the token.
112:     */
113:    function symbol() external view returns (string memory);
114:
115:    /**
116:     * @dev Returns the decimals places of the token.
117:     */
118:    function decimals() external view returns (uint8);
119:}
```

Description

SAir : A `virtual` modifier should be added before `name()`, `symbol()`, and `decimals()` function declarations in the `ERC20Metadata` interface to maintain consistency of the interface.

Recommendation

SAir : Updates to the ERC20Metadata interface are required for consistency.

Consider below fix:

```
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view virtual returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view virtual returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view virtual returns (uint8);
}
```

Client Response

Acknowledged. Would not change it as it's not a security risk.

HIB-6:Gas Optimization - Use CustomError Instead of String for `require`

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Acknowledged	0xxm

Code Reference

- code/contract.sol#L394
- code/contract.sol#L400
- code/contract.sol#L364
- code/contract.sol#L625
- code/contract.sol#L395
- code/contract.sol#L481

```
364:require(

394:require(from != address(0), "ERC20: transfer from the zero address");

395:require(to != address(0), "ERC20: transfer to the zero address");

400:require(

481:require(owner != address(0), "ERC20: approve from the zero address");

625:require(
```

Description

0xxm : When using `require` or `revert`, CustomError is more gas efficient than string description, especially when the string exceeds 32 bytes, more gas will be consumed. The error message described using CustomError is only compiled into four bytes. Generally, around 250-270 gas can be saved for one CustomError replacement when compiler optimization is turned off, 60-80 gas can be saved even if compiler optimization is turned on.

Recommendation

0xxm : Use CustomError instead of string for `require` description.

Client Response

Acknowledged. Would not change it as it's not a security risk.

HIB-7:Suggest to add function to withdraw Ether or Token locked in contract

Category	Severity	Client Response	Contributor
Logical	Informational	Acknowledged	0xxm

Code Reference

- code/contract.sol#L661

```
661:}
```

Description

0xxm : Occasionally, ether or tokens might be sent to the contract by mistake. However, current contract is not able to withdraw Ether or transfer out any token when it happens, resulting in ether/token being permanently locked in this contract.

Recommendation

0xxm : It is suggested to add a `rescue` function as below:

```
function rescue(address token, uint256 amount) external onlyOwner{
    address receiver = owner();
    if (token == address(0)) Address.sendValue(payable(receiver), amount);
    else IERC20(token).safeTransfer(receiver, amount);
}
```

Client Response

Acknowledged. Would not change it as it's not a security risk.

HIB-8:Solidity version inconsistency in `contract.sol` smart contract

Category	Severity	Client Response	Contributor
Language Specific	Informational	Declined	hunya, SAir

Code Reference

- code/contract.sol#L6
- code/contract.sol#L97
- code/contract.sol#L125
- code/contract.sol#L151
- code/contract.sol#L558
- code/contract.sol#L645

```
6:pragma solidity ^0.8.0;

97:pragma solidity ^0.8.0;

125:pragma solidity ^0.8.0;

151:pragma solidity ^0.8.0;

558:pragma solidity ^0.8.0;

645:pragma solidity ^0.8.15;
```

Description

hunya : contract.sol solidity files have a pragma solidity version number with ^0.8.15 . The caret(^) points to unlocked pragma, meaning the compiler will use the specified version or above.
SAir : Different contract files declare different versions of the Solidity compiler. To avoid potential compilation errors and incompatibility issues, the same version of Solidity should be used consistently.

Recommendation

hunya : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contract.
SAir : Try to standardize the declarations across all contract files.

Consider below fix:

```
pragma solidity ^0.8.15;
```

Client Response

Declined

HIB-9:Unnamed function parameter in `HUTToken` contract and `HBKToken` contract

Category	Severity	Client Response	Contributor
Code Style	Informational	Declined	SAir

Code Reference

- code/contract.sol#L611-L647

```
611:* `onlyOwner` functions anymore. Can only be called by the current owner.
612:     *
613:     * NOTE: Renouncing ownership will leave the contract without an owner,
614:     * thereby removing any functionality that is only available to the owner.
615:     */
616:    function renounceOwnership() public virtual onlyOwner {
617:        _transferOwnership(address(0));
618:    }
619:
620:    /**
621:     * @dev Transfers ownership of the contract to a new account (`newOwner`).
622:     * Can only be called by the current owner.
623:     */
624:    function transferOwnership(address newOwner) public virtual onlyOwner {
625:        require(
626:            newOwner != address(0),
627:            "Ownable: new owner is the zero address"
628:        );
629:        _transferOwnership(newOwner);
630:    }
631:
632:    /**
633:     * @dev Transfers ownership of the contract to a new account (`newOwner`).
634:     * Internal function without access restriction.
635:     */
636:    function _transferOwnership(address newOwner) internal virtual {
637:        address oldOwner = _owner;
638:        _owner = newOwner;
639:        emit OwnershipTransferred(oldOwner, newOwner);
640:    }
641:}
642:
643:// File: contracts/Tokens.sol
644:
645:pragma solidity ^0.8.15;
646:
647:contract HUTToken is ERC20, Ownable {
```

Description

SAir : In the constructor of the HUTToken `contract and` HBKToken `contract, there is an` initialSupply `parameter, but it is not explicitly named` initialSupply` and is used directly.

Recommendation

SAir : Good coding practices should explicitly name all function parameters to improve code readability.

Consider below fix in `HUTToken` contract and `HBKToken` contract:

```
contract HUTToken is ERC20, Ownable {
    uint256 private _initialSupply;

    constructor(uint256 _initialSupply) ERC20("Hibiki Utility Token", "HUT") {
        _mint(msg.sender, _initialSupply);
    }

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }
}

contract HBKToken is ERC20 {
    uint256 private _initialSupply;

    constructor(uint256 _initialSupply) ERC20("Hibiki Token", "HBK") {
        _mint(msg.sender, _initialSupply);
    }
}
```

Client Response

Declined

HIB-10:Lack of marking in `contract.sol` contract

Category	Severity	Client Response	Contributor
Code Style	Informational	Declined	SAir

Code Reference

- code/contract.sol#L93
- code/contract.sol#L121
- code/contract.sol#L147
- code/contract.sol#L558

```
93:// File: @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol

121:// File: @openzeppelin/contracts/utils/Context.sol

147:// File: @openzeppelin/contracts/token/ERC20/ERC20.sol

558:pragma solidity ^0.8.0;
```

Description

SAir : There are several contracts in the code that do not have the SPDX-License-Identifier identifier at the beginning of the contract.SPDX-License-Identifier is an SPDX License identifier that specifies the contract's license so that other developers and users are aware of the code's license. The absence of this identifier may lead to uncertainty or inconvenience when using or distributing the code.

Recommendation

SAir : In order to follow best practices, it is recommended to add the SPDX-License-Identifier identifier at the beginning of each contract and specify the license type of the contract. Typically, the MIT license is one of the common open source licenses, so the identifier can be added at the beginning of each contract as shown below:

```
// SPDX-License-Identifier: MIT
```

Client Response

Declined

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.