Competitive Security Assessment

Holonym-P1

Sep 25th, 2023

Secure3

secure3.io

Summary

Holonym is a privacy-preserving identity protocol. It augments standard identity tools such as phone number or government ID verification with a privacy layer based on zkSNARKs. Currently it is used for privacy-preserving Sybil resistance and ID verification. Holonym is further developing tools for identity-based zkAccounts including noncustodial account creation and recovery primitives.

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	Holonym-P1
Platform & Language	Solidity
Codebase	• https://github.com/holonym-foundation/id-hub-contracts • audit commit - 3ad8ea611226343c337bdc4247ae5124f679a2f8 • final commit - 4763f90dadcee434a3da0fd6e031a5a12ebe3f00
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

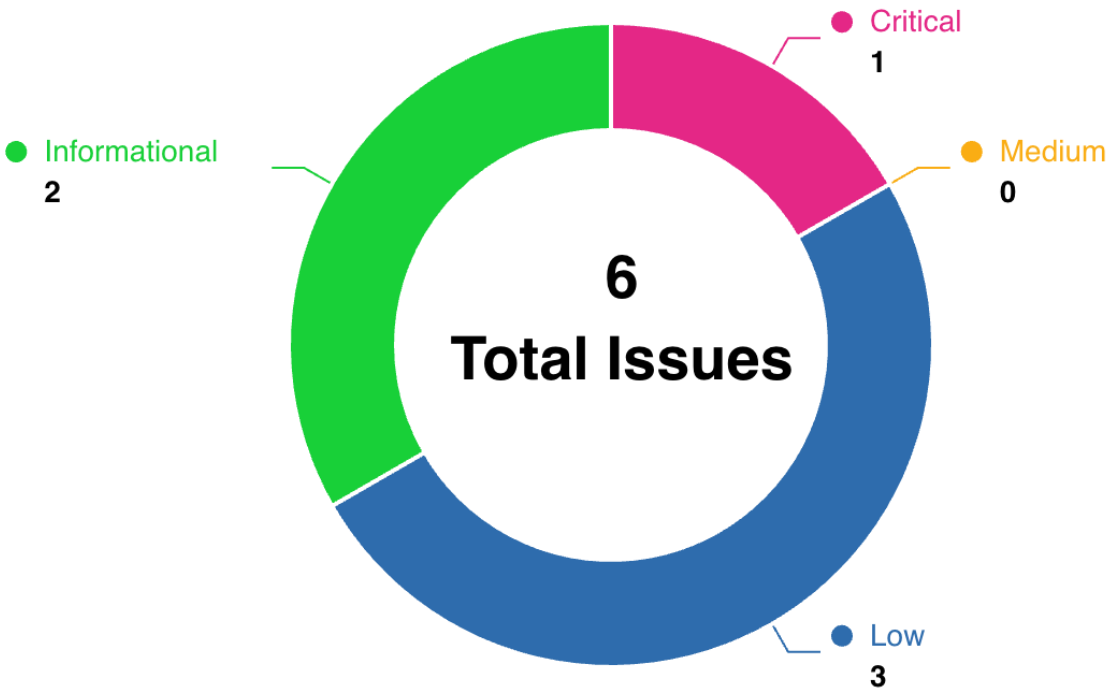
Code Vulnerability Review Summary

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

Audit Scope

File	SHA256 Hash
./contracts/custom-proofs/uniqueness/SybilGovID.sol	58efe1a5eed08ce902aaa2a8190eeef42e3cddf606cbc9b54466f6e03a4be60c
./contracts/custom-proofs/PaidProof.sol	3b27a8bc54d130d37d9bc29507b3ad72381e21bb24634b4b646fdb269c8ffa47

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
HYM-1	One Person can verify one more time at each new contract redeployment in `SybilGovID.sol`	Logical	Critical	Mitigated	0xtruthfulmonkey
HYM-2	use `call()` instead of `transfer` to transfer ETH	Language Specific	Low	Fixed	crjr0629, 0xtruthfulmonkey

HYM-3	User overpaying for price in `SybilGovID::prove()`	Logical	Low	Fixed	crjr0629, 0xtruthfulmonkey, infinityhacker
HYM-4	Using address.transfer to trasnfer Ether is not suggested	DOS	Low	Fixed	infinityhacker
HYM-5	Remove unnecessary import	Code Style	Informational	Fixed	crjr0629
HYM-6	Lack of proper notification for important state changes in contract `PaidProof.sol`	Code Style	Informational	Fixed	0xtruthfulmonkey

HYM-1:One Person can verify one more time at each new contract redeployment in `SybilGovID.sol`

Category	Severity	Client Response	Contributor
Logical	Critical	Mitigated	0xtruthfulmonkey

Code Reference

- code/contracts/custom-proofs/uniqueness/SybilGovID.sol#L37-L65

```
37:function isUniqueForAction(address addr, uint actionId) public view returns (bool unique) {
38:        return (
39:            verifications[keccak256(abi.encodePacked(addr, actionId))]) ||
40:            (legacySupport && oldContract.isUniqueForAction(addr, actionId)
41:        );
42:    }
43:
44:    // It is useful to separate this from the prove() function which is changes state, so that so
mebody can call this off-chain as a view function.
45:    // Then, they can maintain their own off-chain list of footprints and verified address
46:    function proofIsValid(Proof calldata proof, uint[5] memory input) public view returns (bool i
sValid) {
47:        require(roots.rootIsRecent(input[0]), "The root provided was not found in the Merkle tre
e's recent root list");
48:        // Checking msg.sender no longer seems very necessary and prevents signature-free interac
tions. Without it, relayers can submit cross-chain transactions without the user signature. Thus, we
are deprecating this check:
49:        // require(uint256(uint160(msg.sender)) == input[1], "Second public argument of proof mus
t be your address");
50:
51:        require(isValidIssuer(input[2]), "Proof must come from correct issuer's address");
52:        require(!masalaWasUsed[input[4]], "One person can only verify once");
53:        require(verifier.verifyTx(proof, input), "Failed to verify ZKP");
54:        return true;
55:    }
56:
57:    /// @param proof PairingAndProof.sol Proof struct
58:    /// @param input The public inputs to the proof, in ZoKrates' format
59:    function prove(Proof calldata proof, uint256[5] calldata input) public payable needsPayment {
60:        require(proofIsValid(proof, input));
61:        masalaWasUsed[input[4]] = true; //input[4] is address
62:        bytes32 commit = keccak256(abi.encodePacked(uint160(input[1]), input[3])); //input[1] is
address of user to be registered for actionId, input[3] is actionId
63:        verifications[commit] = true;
64:        emit Uniqueness(msg.sender, input[3]);
65:    }
```

Description

0xtruthfulmonkey : `masalaWasUsed` can be reused on new deployments of sybilGovID, it does not check the legacy contract if this has been used unlike `verifications` array, this is important as each new redeployed contract is an extension of the storage life of the previous. a new deployment invalidates the `masalaWasUsed` of the previous.

Recommendation

0xtruthfulmonkey : Check legacy contract to validate that the current footprint has been used before.

Alternatively,

decouple the state from the logic for `SybilGovID.sol`, that way when there is a new logical change the contract does not have to make multiple nested calls to validate state. Example of decoupling : Using external storage pattern

Client Response

Mitigated, Thanks. The function that has been used in production for sybil resistance is isUniqueForAction so this does not harm any production contracts. However, the proofIsValid was also meant to be sybil-resistant, which this finding shows it is not, at least when used with legacy contracts. Thus, I changed the name to proofIsValidNonLegacy to avoid circumstances where it could be unintentionally used in the future in dangerous ways. Our future protocol upgrades will likely change the nullifier scheme, and we will update this method accordingly when those future changes are done.

HYM-2:use `call()` instead of `transfer` to transfer ETH

Category	Severity	Client Response	Contributor
Language Specific	Low	Fixed	crjr0629, 0xtruthfulmonkey

Code Reference

- code/contracts/custom-proofs/PaidProof.sol#L15-L17
- code/contracts/custom-proofs/PaidProof.sol#L16

```
15:function collectPayments() public onlyOwner {
16:         payable(owner()).transfer(address(this).balance);
17:     }

16:payable(owner()).transfer(address(this).balance);
```

Description

crjr0629 : the contract `PaidProof.sol` uses `transfer()` to transfer eth to the owner, this should be avoided as stated here. Gas might not be sufficient in future updates of the network.
0xtruthfulmonkey : When sending ETH, use call() instead of transfer().

The transfer() function only allows the recipient to use 2300 gas. If the recipient uses more than that, transfers will fail. In the future gas costs might change increasing the likelihood of that happening.

There is no risk associated with using call here as the function sends the entire balance to the owner.

Recommendation

crjr0629 : Use `call()` instead to transfer ETH, since the owner is the only one who can call this function
0xtruthfulmonkey : Swap Transfer for call.

```
(bool success, ) = payable(owner()).call{value: address(this).balance}("");
require(success, "Transfer failed.")
```

Client Response

Fixed

HYM-3:User overpaying for price in `SybilGovID::prove()`

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	crjr0629, 0xtruthfulmonkey, infinityhacker

Code Reference

- code/contracts/custom-proofs/PaidProof.sol#L19-L22
- code/contracts/custom-proofs/uniqueness/SybilGovID.sol#L59-L65
- code/contracts/custom-proofs/uniqueness/SybilGovID.sol#L59-L64

```
19:modifier needsPayment() {
20:         require(msg.value >= price, "Missing payment");
21:         _;
22:     }

59:function prove(Proof calldata proof, uint256[5] calldata input) public payable needsPayment {
60:         require(proofIsValid(proof, input));
61:         masalaWasUsed[input[4]] = true; //input[4] is address
62:         bytes32 commit = keccak256(abi.encodePacked(uint160(input[1]), input[3])); //input[1] is
address of user to be registered for actionId, input[3] is actionId
63:         verifications[commit] = true;
64:         emit Uniqueness(msg.sender, input[3]);
65:     }
```

```
59:function prove(Proof calldata proof, uint256[5] calldata input) public payable needsPayment {
60:         require(proofIsValid(proof, input));
61:         masalaWasUsed[input[4]] = true; //input[4] is address
62:         bytes32 commit = keccak256(abi.encodePacked(uint160(input[1]), input[3])); //input[1] is
address of user to be registered for actionId, input[3] is actionId
63:         verifications[commit] = true;
64:         emit Uniqueness(msg.sender, input[3]);
```

Description

crjr0629 : The idea of `price` is to pay a fixed amount of ETH in order to perform an action. The function `prove()` allows a user to send a greater amount than `price`, this shouldn't be allowed, or `price` should be changed to `minAmo unt` or similar naming.

0xtruthfulmonkey : The validation on the approve function checks that the msg.value is `require(msg.value >= p r-`

`ice, "Missing payment")`, this can be problematic in a number of scenarios. This means that when a user pays more than the price stipulated in the contract, the contract accepts it and makes no provision to send the excess of the price `(excess = msg.value - price)` to the user.

The likelihood of this occurring is above average being that the contract does not send proper notification when the price is updated on the contract, in the chance that for example the price was 20eth and then updated to 10eth, the user unaware will send 20eth still and the contract would have no way of reimbursing the user the excess 10eth.

infinityhacker : According to the logic in contract `SybilGovID`, function `prove`, it receive ether payment and check if a proof is valid. And the `needsPayment` modifer checks if the payment is greater than the price. But after verifying the prove, it does not return the extra ether payment as the `needsPayment` modifier only checks `msg.value >= price`, which make prover lost

Recommendation

crjr0629 : slightly modify the modifier `needsPayment()` in `PaidProof.sol` to `require(msg.value == price, "Missing payment");`

Check for equality of `msg.value` and `price`.

0xtruthfulmonkey : Either validate the msg.value to be `require(msg.value == price, "message here")` or reimburse the user the excess cost after deducting the price from msg.value `(excess = msg.value - price)`.

infinityhacker : Return the extra ether payment using openzeppelin's safeTransfer function and also add nonReentrance modifier

Client Response

Fixed

HYM-4:Using address.transfer to trasnfer Ether is not suggested

Category	Severity	Client Response	Contributor
DOS	Low	Fixed	infinityhacker

Code Reference

- code/contracts/custom-proofs/PaidProof.sol#L15-L17

```
15:function collectPayments() public onlyOwner {
16:         payable(owner()).transfer(address(this).balance);
17:     }
```

Description

infinityhacker : As the logic in contract `PaidProof`, function `collectPayments`, it aims to transfer ether to contract owner, but after Ethereum istanbul upgrade, the cost of `sload` has increased, so for some Ether transfer, if the receiver is a proxy contract, it may not receive the ether correctly, as default gas in transfer is only 2300 and the `sload` opcode already cost 800 of it.

Recommendation

infinityhacker : Refer to Openzeppelin suggestion, we suggest using openzeppelin's `safeTransfer` function

Client Response

Fixed

HYM-5:Remove unnecessary import

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	crjr0629

Code Reference

- code/contracts/Roots.sol#L3

```
3:import "hardhat/console.sol";
```

Description

crjr0629 : The contract `Roots.sol` imports `hardhat/console.sol` which is not used

Recommendation

crjr0629 : rename this import from the code.

Client Response

Fixed

HYM-6:Lack of proper notification for important state changes in contract `PaidProof.sol`

Category	Severity	Client Response	Contributor
Code Style	Informational	Fixed	0xtruthfulmonkey

Code Reference

- code/contracts/custom-proofs/PaidProof.sol#L11-L13
- code/contracts/custom-proofs/PaidProof.sol#L24-L36

```
11:function setPrice(uint newPrice) public onlyOwner {
12:            price = newPrice;
13:    }

24:function allowIssuers(uint[] memory issuerAddresses) public onlyOwner {
25:        uint8 i;
26:        for (i = 0; i < issuerAddresses.length; i++) {
27:            allowedIssuers[issuerAddresses[i]] = true;
28:        }
29:    }
30:
31:    function revokeIssuers(uint[] memory issuerAddresses) public onlyOwner {
32:        uint8 i;
33:        for (i = 0; i < issuerAddresses.length; i++) {
34:            allowedIssuers[issuerAddresses[i]] = false;
35:        }
36:    }
```

Description

0xtruthfulmonkey : For functions `setPrice()`, `revokeIssuers()`, `allowIssuers()` that change important states in the contract, there should be proper notification for them. This is more important in the `setPrice` function as it a function that sets an important state that the user needs to be aware of both onchain and offchain, this is also pertinent for offchain systems that monitor the contract, for them to know when critical aspects of the contract has been updated.

Ex of such security issues: Events not emitted for important state changes Missing events for critical changes Lack of event emission after rebalancing fixed borrow rate

Recommendation

0xtruthfulmonkey : Emit events when price and issuers are updated.

Client Response

Fixed

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.