Competitive Security Assessment

Deviews

Feb 1st, 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	Deviews
Platform & Language	Solidity
Codebase	• https://github.com/ChaChingLabs/DVS/ • audit commit - da446993c0878f69175e7b8d9a0999be23089f52 • final commit - 98dfec882f9ccf092e55dbf829f14c463d470381
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

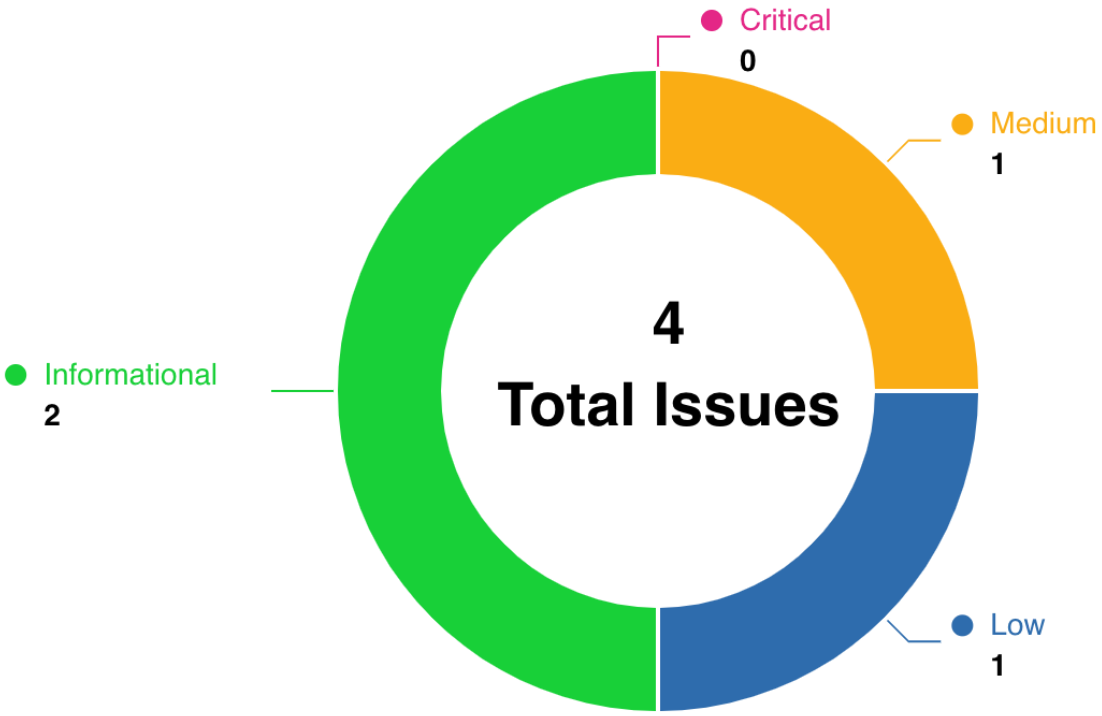
Code Vulnerability Review Summary

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

Audit Scope

File	Commit Hash
code/contracts/Deviews.sol	da446993c0878f69175e7b8d9a0999be23089f52

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
DVW-1	Missing Emit Events	Code Style	Informational	Fixed	Hellobloc, zzzix
DVW-2	Privileged Accounts `BUSINESS_ROLE` can Transfer Tokens of Any User	Privilege Related	Low	Fixed	Hellobloc
DVW-3	Tokenomics Implementation Inconsistency	Logical	Medium	Fixed	Hellobloc
DVW-4	Gas optimization in `Deviews` contract	Gas Optimization	Informational	Acknowledged	Hupixiong3

DVW-1:Missing Emit Events

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/Deviews.sol#L148-L195 • code/contracts/Deviews.sol#L219-L232 • code/contracts/Deviews.sol#L307-L333	Fixed	Hellobloc, zzzix

Code

```
148:     function updateTeam(address newTeam)
149:         public
150:         onlyRole(BUSINESS_ROLE)
151:         returns (bool)
152:     {
153:         _team = newTeam;
154:         return true;
155:     }
156:
157:     //Update the address of the treasury
158:     function updateTreasury(address newTreasury)
159:         public
160:         onlyRole(BUSINESS_ROLE)
161:         returns (bool)
162:     {
163:         _treasury = newTreasury;
164:         return true;
165:     }
166:
167:     //Update the address of the staker
168:     function updateStaker(address newStaker)
169:         public
170:         onlyRole(BUSINESS_ROLE)
171:         returns (bool)
172:     {
173:         _staker = newStaker;
174:         return true;
175:     }
176:
177:     //Update the address of the bonus pool
178:     function updateDefaultBonusPool(address newPool)
179:         public
180:         onlyRole(BUSINESS_ROLE)
181:         returns (bool)
182:     {
183:         _defaultBonusPool = newPool;
184:         return true;
185:     }
186:
187:     //Update the address of the signer
188:     function updateSignAddress(address newSignAddress)
189:         public
```

```
190:         onlyRole(BUSINESS_ROLE)
191:         returns (bool)
192:     {
193:         _signAddress = newSignAddress;
194:         return true;
195:     }

219:     function updateProportion(
220:         uint256 newTeamProportion,
221:         uint256 newTreasuryProportion,
222:         uint256 newStakerProportion,
223:         uint256 newBurnProportion,
224:         uint256 newPoolProportion
225:     ) public onlyRole(BUSINESS_ROLE) returns (bool) {
226:         teamProportion = newTeamProportion;
227:         treasuryProportion = newTreasuryProportion;
228:         stakerProportion = newStakerProportion;
229:         burnProportion = newBurnProportion;
230:         poolProportion = newPoolProportion;
231:         return true;
232:     }

307:     function rewards(
308:         address toAddress,
309:         uint256 value,
310:         uint256 deadline,
311:         uint8 v,
312:         bytes32 r,
313:         bytes32 s
314:     ) public virtual whenNotPaused {
315:         //check the current time whether reached the deadline
316:         require(block.timestamp <= deadline, "expired deadline");
317:         //verify the signature
318:         bytes32 structHash = keccak256(
319:             abi.encode(
320:                 _REWARDS_TYPEHASH,
321:                 _defaultBonusPool,
322:                 toAddress,
323:                 value,
324:                 _useNonce(toAddress),
325:                 deadline
326:             )
327:         );
```

```
328:         bytes32 hash = _hashTypedDataV4(structHash);
329:         address signer = ECDSAUpgradeable.recover(hash, v, r, s);
330:         require(signer == _signAddress, "invalid signature");
331:         //transfer from the bonus pool
332:         _transfer(_defaultBonusPool, toAddress, value);
333:     }
```

Description

Hellobloc : Function that affects the status of sensitive variables should be able to emit events as notifications to user. For example, the following function

- rewards()
- updateTeam()
- updateTreasury()
- updateStaker()
- updateDefaultBonusPool()
- updateSignAddress()
- updateProportion()

zzzix : For funtion that changes the contract state, it is best practice to emit event. The update functions do not have events emitted, `updateTeam()`, `updateTreasury()`, `updateStaker()`, `updateDefaultBonusPool()`, `updateSignAddress()`, `updateProportion()`.

Recommendation

Hellobloc : Consider adding events for sensitive actions, and emit it in the function.
zzzix : emit events at the end of the update functions.

Client Response

Fixed.

DVW-2:Privileged Accounts `BUSINESS_ROLE` can Transfer Tokens of Any User

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Low	• code/contracts/Deviews.sol#L307-L333	Fixed	Hellobloc

Code

```
307:    function rewards(
308:        address toAddress,
309:        uint256 value,
310:        uint256 deadline,
311:        uint8 v,
312:        bytes32 r,
313:        bytes32 s
314:    ) public virtual whenNotPaused {
315:        //check the current time whether reached the deadline
316:        require(block.timestamp <= deadline, "expired deadline");
317:        //verify the signature
318:        bytes32 structHash = keccak256(
319:            abi.encode(
320:                _REWARDS_TYPEHASH,
321:                _defaultBonusPool,
322:                toAddress,
323:                value,
324:                _useNonce(toAddress),
325:                deadline
326:            )
327:        );
328:        bytes32 hash = _hashTypedDataV4(structHash);
329:        address signer = ECDSAUpgradeable.recover(hash, v, r, s);
330:        require(signer == _signAddress, "invalid signature");
331:        //transfer from the bonus pool
332:        _transfer(_defaultBonusPool, toAddress, value);
333:    }
```

Description

Hellobloc : In the current project, there are privileged accounts that can perform the following privileged operations, which may lead to security risks.

- Arbitrary Transfer
  - `BUSINESS_ROLE` (by setting `_defaultBonusPool` to an arbitrary account and using the `reward` method to perform arbitrary `Transfer`)

Recommendation

Hellobloc : We recommend using `multi-signature` accounts to manage `BUSINESS_ROLE` and setting a time lock for privileged operations to prevent other risks such as private key leakage.

Client Response

Fixed.

DVW-3:Tokenomics Implementation Inconsistency

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/Deviews.sol#L76-L79	Fixed	Hellobloc

Code

```
76:         _mint(
77:             address(0xda8646145179Df22cED432BA5255f9B13D88343b),
78:             1000000 * 10**decimals()
79:         );
```

Description

Hellobloc : `Tokenomics` in the current code implementation does not match the whitepaper description. For example, the `total supply` is written as `1,000,000,000` in the white paper, but only `1,000,000` in the code implementation.

```
_mint(
            address(0xda8646145179Df22cED432BA5255f9B13D88343b),
            1_000_000 * 10**decimals()
    );
```

On the other hand, the token distribution percentage is also different from that in the whitepaper.

Recommendation

Hellobloc : We recommend rewriting the specific implementation of `Tokenomics` to match the `whitepaper`.

Client Response

Fixed.

DVW-4:Gas optimization in `Deviews` contract

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/Deviews.sol#L37-L46	Acknowledged	Hupixiong3

Code

```
37:    //The proportion team will get from the bonus
38:    uint256 private teamProportion;
39:    //The proportion treasury will get from the bonus
40:    uint256 private treasuryProportion;
41:    //The proportion stakers will get from the bonus
42:    uint256 private stakerProportion;
43:    //The proportion for burning from the bonus
44:    uint256 private burnProportion;
45:    //The proportion reviewers will get from the bonus
46:    uint256 private poolProportion;
```

Description

Hupixiong3 : According to the use case in the code, the proportions are small integers. Using `uint256` will increase gas consumption when deploying and calling the contract.

Recommendation

Hupixiong3 : Using integer types with smaller precision.

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.