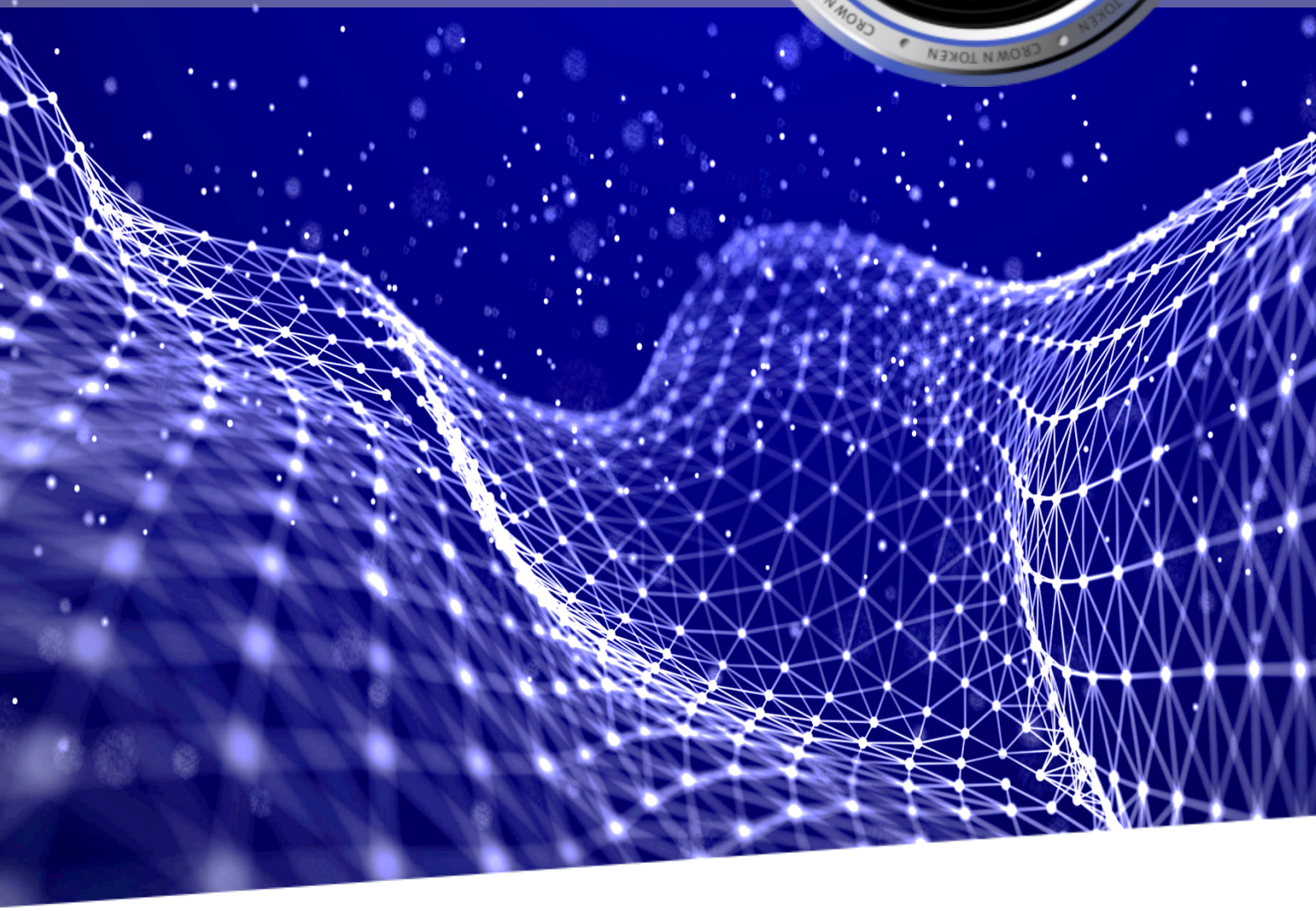
Vega Investment Group Limited

CrownToken

Smart Contract Audit Report



Date Issued: 8 Nov 2022

Version: Final v1.0

ValiX Consulting

Public

Table of Contents

Executive Summary

Overview

Valix conducted a smart contract audit to evaluate potential security issues of the **CrownToken feature**. This audit report was published on `8 Nov 2022`. The audit scope is limited to the **CrownToken feature.** Our security best practices strongly recommend that the **Vega Investment Group Limited** conduct a full security audit for both on-chain and off-chain components of its infrastructure and their interaction. A comprehensive examination has been performed during the audit process utilizing Valix's Formal Verification, Static Analysis, and Manual Review techniques.

About CrownToken

CROWN token is an entertainment token that bridges traditional IPs with blockchain technology to enhance the core business and create additional value for both IP owners and the community. The token is supported by high-quality IP projects, including animated movies, series, and platforms.

Scope of Work

The security audit conducted does not replace the full security audit of the overall Vega Investment Group's protocol. The scope is limited to the **CrownToken feature** and its related smart contracts.

The security audit covered the components at this specific state:

Item	Description
Components	▪ *CrownToken smart contract* ▪ *Imported associated smart contracts and libraries*
Git Repository	▪ *https://github.com/pellartech/vuca-blockchain-public*
Audit Commit	▪ *13fcd040cac4e00d4a2df2adfbd31aaaffa09ecd (branch: main)*
Certified Commit	▪ *5cc2e3fcb2a4268bd97e6e02395bac08b592a91d (branch: main)*
Audited Files	▪ *./contracts/CrownToken.sol* ▪ *Other imported associated Solidity files*

Excluded Files/Contracts	▪ *./contracts/VucaOwnable.sol* ▪ *./contracts/VucaStaking.sol* ▪ *./contracts/mock/CWT.sol* ▪ *./contracts/mock/USDT.sol*

Remark: Our security best practices strongly recommend that the Vega Investment Group team conduct a full security audit for both on-chain and off-chain components of its infrastructure and the interaction between them.

Auditors

Role	Staff List
Auditors	**Anak Mirasing** **Atitawat Pol-in** **Kritsada Dechawattana** **Parichaya Thanawuthikrai** **Phuwanai Thummavet**
Authors	**Anak Mirasing** **Atitawat Pol-in** **Kritsada Dechawattana** **Parichaya Thanawuthikrai** **Phuwanai Thummavet**
Reviewers	**Sumedt Jitpukdebodin**

Disclaimer

Our smart contract audit was conducted over a limited period and was performed on the smart contract at a single point in time. As such, the scope was limited to current known risks during the work period. The review does not indicate that the smart contract and blockchain software has no vulnerability exposure.

We reviewed the security of the smart contracts with our best effort, and we do not guarantee a hundred percent coverage of the underlying risk existing in the ecosystem. The audit was scoped only in the provided code repository. The on-chain code is not in the scope of auditing.

This audit report does not provide any warranty or guarantee, nor should it be considered an “approval” or “endorsement” of any particular project. This audit report should also not be used as investment advice nor provide any legal compliance.

Audit Result Summary

From the audit results and the remediation and response from the developer, Valix trusts that the **CrownToken feature** has sufficient security protections to be safe for use.

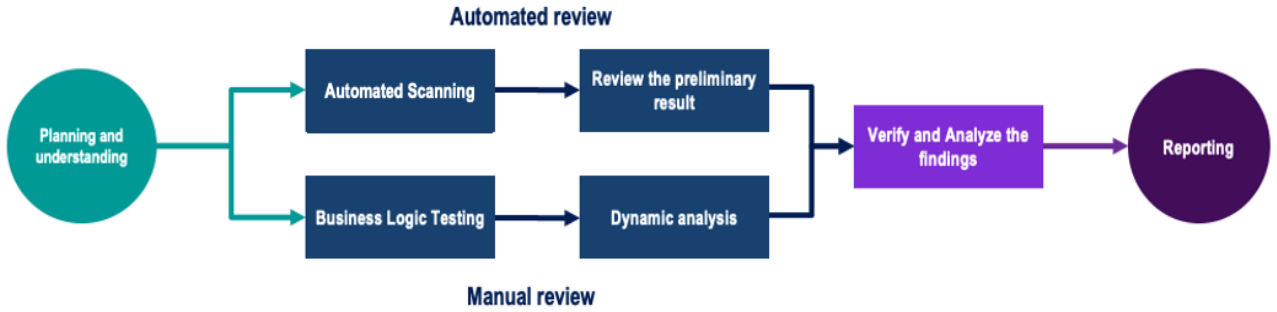
Initially, Valix was able to identify **4 issues** that were categorized from the "Critical" to "Informational" risk level in the given timeframe of the assessment. For the reassessment, **the *Vega Investment Group* team fixed all issues**. Below is the breakdown of the vulnerabilities found and their associated risk rating for each assessment conducted.

Target	Assessment Result					Reassessment Result				
	C	H	M	L	I	C	H	M	L	I
CrownToken	-	-	2	2	-	-	-	0	0	-

Note: *Risk Rating* **C** *Critical,* **H** *High,* **M** *Medium,* **L** *Low,* **I** *Informational*

Methodology

The smart contract security audit methodology is based on Smart Contract Weakness Classification and Test Cases (SWC Registry), CWE, well-known best practices, and smart contract hacking case studies. Manual and automated review approaches can be mixed and matched, including business logic analysis in terms of the malicious doer's perspective. Using automated scanning tools to navigate or find offending software patterns in the codebase along with a purely manual or semi-automated approach, where the analyst primarily relies on one's knowledge, is performed to eliminate the false-positive results.



Planning and Understanding

- Determine the scope of testing and understanding of the application's purposes and workflows.
- Identify key risk areas, including technical and business risks.
- Determine which sections to review within the resource constraints and review method – automated, manual or mixed.

Automated Review

- Adjust automated source code review tools to inspect the code for known unsafe coding patterns.
- Verify the tool's output to eliminate false-positive results, and adjust and re-run the code review tool if necessary.

Manual Review

- Analyzing the business logic flaws requires thinking in unconventional methods.
- Identify unsafe coding behavior via static code analysis.

Reporting

- Analyze the root cause of the flaws.
- Recommend improvements for secure source code.

Audit Items

We perform the audit according to the following categories and test names.

Category	ID	Test Name
Security Issue	SEC01	*Authorization Through tx.origin*
	SEC02	*Business Logic Flaw*
	SEC03	*Delegatecall to Untrusted Callee*
	SEC04	*DoS With Block Gas Limit*
	SEC05	*DoS with Failed Call*
	SEC06	*Function Default Visibility*
	SEC07	*Hash Collisions With Multiple Variable Length Arguments*
	SEC08	*Incorrect Constructor Name*
	SEC09	*Improper Access Control or Authorization*
	SEC10	*Improper Emergency Response Mechanism*
	SEC11	*Insufficient Validation of Address Length*
	SEC12	*Integer Overflow and Underflow*
	SEC13	*Outdated Compiler Version*
	SEC14	*Outdated Library Version*
	SEC15	*Private Data On-Chain*
	SEC16	*Reentrancy*
	SEC17	*Transaction Order Dependence*
	SEC18	*Unchecked Call Return Value*
	SEC19	*Unexpected Token Balance*
	SEC20	*Unprotected Assignment of Ownership*
	SEC21	*Unprotected SELFDESTRUCT Instruction*
	SEC22	*Unprotected Token Withdrawal*
	SEC23	*Unsafe Type Inference*
	SEC24	*Use of Deprecated Solidity Functions*
	SEC25	*Use of Untrusted Code or Libraries*
	SEC26	*Weak Sources of Randomness from Chain Attributes*
	SEC27	*Write to Arbitrary Storage Location*

Category	ID	Test Name
Functional Issue	FNC01	*Arithmetic Precision*
	FNC02	*Permanently Locked Fund*
	FNC03	*Redundant Fallback Function*
	FNC04	*Timestamp Dependence*
Operational Issue	OPT01	*Code With No Effects*
	OPT02	*Message Call with Hardcoded Gas Amount*
	OPT03	*The Implementation Contract Flow or Value and the Document is Mismatched*
	OPT04	*The Usage of Excessive Byte Array*
	OPT05	*Unenforced Timelock on An Upgradeable Proxy Contract*
Developmental Issue	DEV01	*Assert Violation*
	DEV02	*Other Compilation Warnings*
	DEV03	*Presence of Unused Variables*
	DEV04	*Shadowing State Variables*
	DEV05	*State Variable Default Visibility*
	DEV06	*Typographical Error*
	DEV07	*Uninitialized Storage Pointer*
	DEV08	*Violation of Solidity Coding Convention*
	DEV09	*Violation of Token (ERC20) Standard API*

Risk Rating

To prioritize the vulnerabilities, we have adopted the scheme of five distinct levels of risk: **Critical**, **High**, **Medium**, **Low**, and **Informational**, based on OWASP Risk Rating Methodology. The risk level definitions are presented in the table.

Risk Level	Definition
Critical	The code implementation does not match the specification, and it could disrupt the platform.
High	The code implementation does not match the specification, or it could result in losing funds for contract owners or users.
Medium	The code implementation does not match the specification under certain conditions, or it could affect the security standard by losing access control.
Low	The code implementation does not follow best practices or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.
Informational	Findings in this category are informational and may be further improved by following best practices and guidelines.

The **risk value** of each issue was calculated from the product of the **impact** and **likelihood values**, as illustrated in a two-dimensional matrix below.

- **Likelihood** represents how likely a particular vulnerability is exposed and exploited in the wild.
- **Impact** measures the technical loss and business damage of a successful attack.
- **Risk** demonstrates the overall criticality of the risk.

Impact \ Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Informational

The shading of the matrix visualizes the different risk levels. Based on the acceptance criteria, the risk levels "Critical" and "High" are unacceptable. Any issue obtaining the above levels must be resolved to lower the risk to an acceptable level.

Findings

Review Findings Summary

The table below shows the summary of our assessments.

No.	Issue	Risk	Status	Functionality is in use
1	Possibly Permanent Ownership Removal	Medium	Fixed	In use
2	Unsafe Ownership Transfer	Medium	Fixed	In use
3	Compiler Is Not Locked To Specific Version	Low	Fixed	In use
4	Compiler May Be Susceptible To Publicly Disclosed Bugs	Low	Fixed	In use

The statuses of the issues are defined as follows:

Fixed: The issue has been completely resolved and has no further complications.

Partially Fixed: The issue has been partially resolved.

Acknowledged: The issue's risk has been reported and acknowledged.

Detailed Result

This section provides all issues that we found in detail.

No. 1	Possibly Permanent Ownership Removal		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*@openzeppelin/contracts/access/Ownable.sol*		
Locations	*Ownable.sol L: 61 - 63*		

Detailed Issue

The *CrownToken* contract inherits from the *Ownable* abstract contract. The *Ownable* contract implements the *renounceOwnership* function (L61 - 63 in the code snippet below), which can remove the contract's ownership permanently.

If the contract owner mistakenly invokes the *renounceOwnership* function, they will immediately lose ownership of the contract, and this action cannot be undone.

Ownable.sol

```
61  function renounceOwnership() public virtual onlyOwner {
62      _transferOwnership(address(0));
63  }

    // (...SNIPPED...)

78  function _transferOwnership(address newOwner) internal virtual {
79      address oldOwner = _owner;
80      _owner = newOwner;
81      emit OwnershipTransferred(oldOwner, newOwner);
82  }
```

Listing 1.1 The *renounceOwnership* function
that can remove the ownership of the contract permanently

Recommendations

We consider the *renounceOwnership* function risky, and the contract owner should use this function with extra care.

If possible, we recommend removing or disabling this function from the contract. The code snippet below shows an example solution to disabling the associated *renounceOwnership* function.

CrownToken.sol

```
16 function renounceOwnership() external override onlyOwner {
17     revert("Ownable: renounceOwnership function is disabled");
18 }
```

Listing 1.2 The disabled *renounceOwnership* function

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Vega Investment Group* team decided to remove the *Ownable* contract from the *CrownToken*. Therefore, this issue was closed.

No. 2	Unsafe Ownership Transfer		
Risk	Medium	Likelihood	Low
		Impact	High
Functionality is in use	In use	Status	Fixed
Associated Files	*@openzeppelin/contracts/access/Ownable.sol*		
Locations	*Ownable.sol L: 69 - 72*		

Detailed Issue

The *CrownToken* contract inherits from the *Ownable* abstract contract. The *Ownable* contract implements the *transferOwnership* function (L69 - 72 in the code snippet below), which can transfer the ownership of the contract from the current owner to another owner.

Ownable.sol

```
69  function transferOwnership(address newOwner) public virtual onlyOwner {
70      require(newOwner != address(0), "Ownable: new owner is the zero address");
71      _transferOwnership(newOwner);
72  }

    // (...SNIPPED...)

78  function _transferOwnership(address newOwner) internal virtual {
79      address oldOwner = _owner;
80      _owner = newOwner;
81      emit OwnershipTransferred(oldOwner, newOwner);
82  }
```

Listing 2.1 The *transferOwnership* function that has the unsafe ownership transfer

From the code snippet above, the address variable *newOwner* (L69) may be incorrectly specified by the current owner by mistake; for example, an address that a new owner does not own was inputted. Consequently, the new owner loses ownership of the contract immediately, and this action is unrecoverable.

Recommendations

We recommend applying the two-step ownership transfer mechanism as shown in the code snippet below.

CrownToken.sol

```
16  function transferOwnership(address _candidateOwner) public override onlyOwner {
17      require(_candidateOwner != address(0), "Ownable: candidate owner is the zero
    address");
18      candidateOwner = _candidateOwner;
19      emit NewCandidateOwner(_candidateOwner);
20  }
21
22  function claimOwnership() external {
23      require(candidateOwner == _msgSender(), "Ownable: caller is not the
    candidate owner");
24      _transferOwnership(candidateOwner);
25      candidateOwner = address(0);
26  }
```

Listing 2.2 The recommended two-step ownership transfer mechanism

This mechanism works as follows.

1. The current owner invokes the *transferOwnership* function by specifying the candidate owner address *_candidateOwner* (L16).
2. The candidate owner proves access to his account and claims the ownership transfer by invoking the *claimOwnership* function (L22)

The recommended mechanism ensures that the ownership of the contract would be transferred to another owner who can access his account only.

The recommended code provides the concept of how to remediate this issue only. The code should be adjusted accordingly.

Reassessment

The *Vega Investment Group* team decided to remove the *Ownable* contract from the *CrownToken*. Therefore, this issue was closed.

No. 3	Compiler Is Not Locked To Specific Version		
Risk	Low	Likelihood	Low
		Impact	Medium
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/CrownToken.sol*		
Locations	*CrownToken.sol L: 2*		

Detailed Issue

The *CrownToken* smart contract should be deployed with the compiler version used in the development and testing process.

The compiler version that is not strictly locked via the *pragma* statement may make the contract incompatible against unforeseen circumstances.

The code that is not locked to a specific version (e.g., using `>=` or `^` directive) is shown below.

CrownToken.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.15;
```

Listing 3.1 The code that is not locked to a specific version

Recommendations

We recommend locking the *pragma* version like the example code snippet below.

```
pragma solidity 0.8.0;
// or
pragma solidity =0.8.0;

contract SemVerFloatingPragmaFixed {
}
```

Reference: *https://swcregistry.io/docs/SWC-103*

Reassessment

The *Vega Investment Group* team locked the pragma version to v0.8.17.

No. 4	Compiler May Be Susceptible To Publicly Disclosed Bugs		
Risk	Low	Likelihood	Low
		Impact	Medium
Functionality is in use	In use	Status	Fixed
Associated Files	*contracts/CrownToken.sol*		
Locations	*CrownToken.sol L: 2*		

Detailed Issue

The *CrownToken* smart contract uses an outdated Solidity compiler version (v0.8.15) which may be susceptible to publicly disclosed vulnerabilities. The latest compiler patch version is 0.8.17, which contains the list of known bugs as the following link:

https://docs.soliditylang.org/en/v0.8.17/bugs.html

The known bugs may not directly lead to the vulnerability, but it may increase an opportunity to trigger some attacks further.

The smart contract that does not use the latest patch version is shown below.

CrownToken.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.15;
```

Listing 4.1 The smart contract that does not use the latest patch version (v0.8.17)

Recommendations

We recommend using the latest patch version, v0.8.17, that fixes all known bugs.

Reassessment

The *Vega Investment Group* team fixed this issue by employing the patch version v0.8.17.

Appendix

About Us

Founded in 2020, Valix Consulting is a blockchain and smart contract security firm offering a wide range of cybersecurity consulting services such as blockchain and smart contract security consulting, smart contract security review, and smart contract security audit.

Our team members are passionate cybersecurity professionals and researchers in the areas of private and public blockchain technology, smart contract, and decentralized application (DApp).

We provide a service for assessing and certifying the security of smart contracts. Our service also includes recommendations on smart contracts' security and gas optimization to bring the most benefit to users and platform creators.

Contact Information


info@valix.io


https://www.facebook.com/ValixConsulting


https://twitter.com/ValixConsulting


https://medium.com/valixconsulting

References

Title	Link
OWASP Risk Rating Methodology	https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
Smart Contract Weakness Classification and Test Cases	https://swcregistry.io/