BLOCKSTARS TECHNOLOGY

Smart Contract Security Audit

CLIENT:	**DLTX**
PROJECT:	BRICKTOPIANS NFT FORGE
ANALYSIS TYPE:	DEEP
VERSION:	1
START DATE:	24th APRIL 2023

REPORT CONTENTS

DECLARATION

This is a BLOCKSTARS TECHNOLOGY smart contract security audit report for our client DLTX and their project, BRICKTOPIANS NFT FORGE.

Results and documentation within this report contain confidential information regarding the clients' contracts.

Analysis results of the smart contracts are supplied containing lists of vulnerabilities and malicious code which could be used to malform the project. Upon the client receiving this report and until the issues are resolved or mitigated, vulnerability results that have been accumulated and listed within this report is kept private between both the BLOCKSTARS TECHNOLOGY auditing team and our respected client alone.

1. Introduction

This audit report contains confidential information of smart contract programs running in the Bricktopians NFT Forge project. It analyses security vulnerabilities, smart contract best practices, and possible attacks using standardised automated tests using static, dynamic, and symbolic analysis tools. We outlined our systematic approach to evaluate potential security issues in core smart contracts in the Bricktopians NFT Forge project and provide an audit summary with remedies for mitigating the vulnerability findings.

2. Project Context

This section explains the client's project in detail with scope.

ITEM	DESCRIPTION
Issuer	DLTX
Website	https://www.dltx.io/clients.html
Source	Smart contract programs
Language	Solidity
Blockchain	Ethereum (ETH)
Git Repository	https://github.com/dltxio/nft.bricktopunks
Audit type	DEEP
Analysis Methods	Static and Dynamic automation tests, Manual functional review, and Unit tests review.
Audit Team	Pura, Faizin, Marcus
Approved By	Nilanga
Timeline	**From:** *21st April 2023* **To:** *27th April 2023*
Change logs	V1

3. Audit Scope

Scope of this project is to identify smart contract vulnerabilities to improve the coding practice in the given smart contract (BrickToForge.sol) from Bricktopians NFT Forge project. The following steps were conducted to audit the given contract.

- Automated testing using analysis tools which includes static, dynamic, and Symbolic analysis methods.
- Manual functional reviews by our smart contract auditors.
- Unit test reviews.

BLOCKSTARS TECHNOLOGY audit method:	**DEEP**

ITEM	DESCRIPTION
Repository	https://github.com/dltxio/nft.bricktopunks
Commit Ids	a054d1954cc6f27a21ec2aa713d88dd4f83efae1
Branch	master
Technical Documentation	Business logics provided: **TRUE** Technical and functional Requirements.docx
Bricktoforge.sol	https://github.com/dltxio/nft.bricktopunks/blob/master/bricktopians/contracts/Bricktoforge.sol

4. Definitions

4.1. Security Severity

SEVERITY	DESCRIPTION
High	High level vulnerabilities may be difficult to exploit, however they also have significant impact on smart contract execution due to lack of secured access control. (Example: Public access to crucial functions and data)
Medium	Medium vulnerabilities do not lead to loss of assets or data, but it is important to fix those issues that may be used to exploit the project.
Low	Low level vulnerabilities are related to out-dated, unused code snippets, and they don't have a significant impact on contract execution.
Informational	Does not contain vulnerabilities, but requires best practices, code standards and documentary code.

5. Analytic Statics

Security Issues	#	Description	Type	Bricktoforge
Programming Issues	1	Unchecked return value from low-level external calls	SWC-104	P
	2	Floating pragma is set	SWC-103	P
	3	Error Handling and logging are implemented	Custom	P
	4	State variable should not be used without being initialized	Custom	P
	5	Is inheritance used properly	SWC-125	P
	6	External components used insecurely	Custom	P
	7	Functions that loop over unbounded data structures	Custom	F
	8	Msg.value should not be used in a loop	Custom	P
Code Specifications, Best Practices	10	Use of the "constant" state mutability modifier is deprecated.	SWC-111	P
	11	Use of the "throw" state mutability modifier is deprecated.	SWC-111	P
	12	Strict equalities should not render the function to be unusable	Custom	P
	13	Use of best Practices	Custom	P
	14	Business logic is implemented as per the documents provided	Custom	P
Gas Optimisation	15	Message call with hardcoded gas amount	SWC-134	P
	16	Check for gas usage and minimize gas consumption.	Custom	F
Risk to Attacks	17	Code contains suicidal, greedy, and prodigal instructions	SWC-106	P
	18	Contract is Haltable	Custom	P
	19	Adopt checks-effects-interactions patterns for any transactions of value	Custom	P
	20	Reduce and remove unnecessary code to reduce attack surface area.	Custom	P
	21	Timestamps should not be used to execute critical functions.	SWC-116	P
	22	Sensitive data in normal form should not be stored on-chain	Custom	P
	23	Vulnerable to Integer overflow and under-flow	SWC-101	P

Key: **F** = Fail **P** = Pass

6. Manual Audit process

6.1. Functions Overview of Bricktoforge.sol

#	Function	Type	Observation	Status
1	**constructor(address bricktopiansAddress)**	Constructor	• Initialises the contract and sets the _bricktopiansAddress variable to the provided address. • Initializees the _self address to BrickToForge contract address	Safe
2	**forge(uint256 burnTokenId, uint256 upgradeTokenId)**	External function	• Allows the owner of an upgrade token to use it to upgrade a burn token, by transferring the burn token to the contract and mapping it to the upgrade token. • The function also checks that the sender is the owner of the upgrade token, and the forge should be active.	Safe
3	**transfer(uint256 tokenId, address to)**	External onlyOwner function	• Allows the owner of the contract to transfer a token to another address.	Safe
4	**onERC721Received(address, address, uint256, bytes memory)**	Public virtual function returns (bytes4)	• A function that is called when the contract receives an ERC721 token. • It returns the function selector.	Safe
5	**tokensOfOwner(address owner)**	External view function returns (uint256[] memory)	• Returns an array of token IDs that are owned by the given address.	Safe
6	**burn(uint256 tokenId)**	External onlyOwner function	• Allows the owner of the contract to burn a specific token.	Safe
7	**burnAllTokens()**	External only owner function	• Allows the owner of the contract to burn all tokens owned by the contract. • It is not safe to have a for loop to burn all tokens at a time, since it will cause out of gas issue when it gets a big number of tokens to loop through and burn. • It is recommended to have a maxBurnCount variable and restrict using a modifier and allows admin/owner to set a safe value to the maxBurnCount considering the out of gas problem.	NOT Safe
8	**toggleIsActive()**	External onlyOwner function	• Allows the owner of the contract to toggle the isActive Boolean variable.	Safe

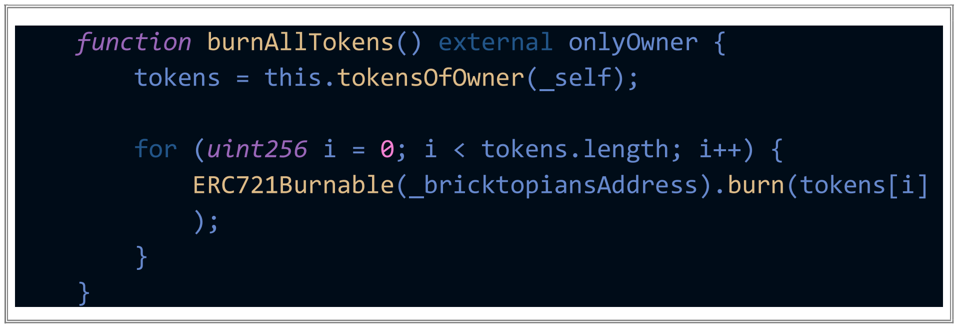
7. Audit Findings in Detail

7.1. BrickToForge.sol

7.1.1. HIGH

1. **Issue:** Using for loop through unbounded data structure

   Function name: burnAllTokens()



```
    function burnAllTokens() external onlyOwner {
        tokens = this.tokensOfOwner(_self);

        for (uint256 i = 0; i < tokens.length; i++) {
            ERC721Burnable(_bricktopiansAddress).burn(tokens[i]
            );
        }
    }
```

   Description:
   Having for loop through unbounded data structure or length, will cause the following problems.

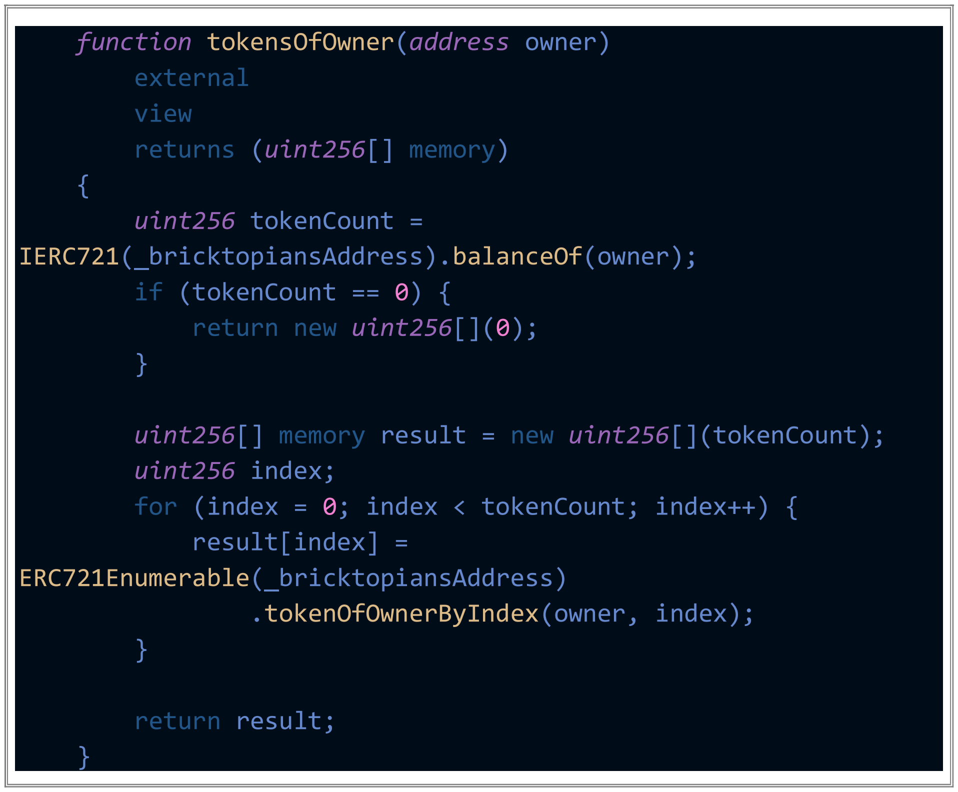
   - Out of gas issue in the middle if the owner does not have enough balance to execute the function.
   - It is likely to have Denial of service attacks.

   Resolution:
   It is recommended to have a setter function (setMaxBurnTokenCount) that can set a value for maximum number of burn tokens to loop over the tokens to burn. This setter function is accessible by onlyOwner, who can set or change the maxBurnTokenCount value, when he wants to burn.

7.1.2. **LOW**

1. **Issue:** Owner address != Zero address.

   Function name: tokensOfOwner



```
    function tokensOfOwner(address owner)
        external
        view
        returns (uint256[] memory)
    {
        uint256 tokenCount =
IERC721(_bricktopiansAddress).balanceOf(owner);
        if (tokenCount == 0) {
            return new uint256[](0);
        }

        uint256[] memory result = new uint256[](tokenCount);
        uint256 index;
        for (index = 0; index < tokenCount; index++) {
            result[index] =
ERC721Enumerable(_bricktopiansAddress)
                .tokenOfOwnerByIndex(owner, index);
        }

        return result;
    }
```

Description:
allowing zero address to be used as owner address can lead to the contract execution failing due to an invalid address.
Resolution:
Add a check to ensure that the owner address is not the zero address.

7.1.3. INFORMATIONAL

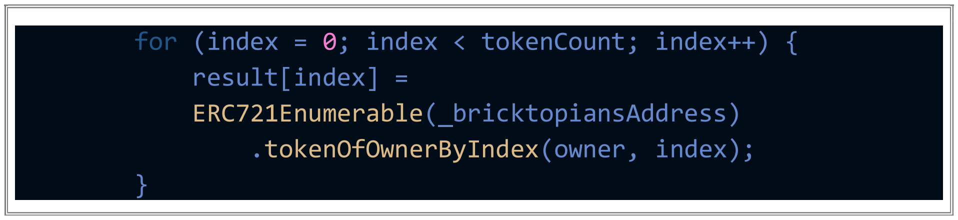
1. **Issue:** public virtual access level

   Function name: onERC721Received

   Description:
   The onERC721Received function is a callback function that is called by other contracts when a token is transferred to the contract address.
   Resolution:
   It is recommended to make the onERC721Received function external instead of public virtual since it is a callback function that is called by other contracts. By making the function external, Solidity can optimize the function call by avoiding copying arguments from memory to the stack, which can improve efficiency.

2. **Issue:** use IERC721Enumerable interface instead of ERC721Enumerable contract.

   Function name: burnAllTokens

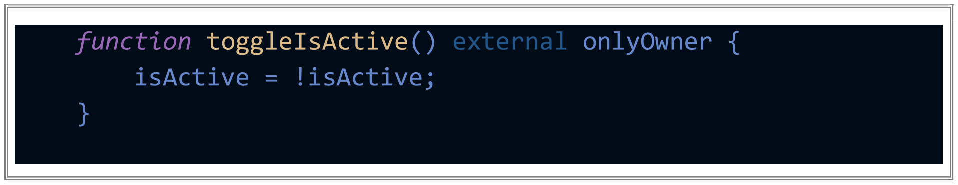


```
        for (index = 0; index < tokenCount; index++) {
            result[index] =
            ERC721Enumerable(_bricktopiansAddress)
                .tokenOfOwnerByIndex(owner, index);
        }
```

   Description:
   The burnAllTokens function is responsible for burning all the tokens owned by the caller.
   Resolution:
   It is recommended to use the IERC721Enumerable interface instead of the ERC721Enumerable contract in the burnAllTokens function. The IERC721Enumerable interface provides a standardized set of functions that can be used to interact with any contract that implements the ERC721 token standard, while the ERC721Enumerable contract is an implementation of the ERC721Enumerable interface.

3. **Issue:** not emitting event when isActive state variable toggled.

   Function name: toggleIsActive



```
    function toggleIsActive() external onlyOwner {
        isActive = !isActive;
    }
```

   Description:
   The toggleIsActive function is responsible for toggling the isActive state variable between true and false.
   Resolution:
   It is recommended to add an event to emit when the isActive state variable is toggled in the toggleIsActive function. As a state variable, the isActive variable represents the current state of the contract, and any changes to its value should be recorded and made visible to the users of the contract.

8. Test Case Review

#	Description	Status
1	It tests if the contract deploys successfully and if the minting function is working properly.	PASS
2	It tests if the contract can forge new tokens.	PASS
3	It tests if the contract prevents non-owners from forging new tokens.	PASS
4	It tests if the contract allows owners to burn tokens.	PASS
5	It tests if the contract prevents non-owners from burning tokens.	PASS
6	It tests if the contract allows owners to burn all tokens.	PASS
7	It tests if the contract prevents non-owners from burning all tokens.	PASS
8	It tests if the contract prevents transfers when the contract is not active.	PASS
9	It tests if the contract allows non-owners to transfer tokens to the owner.	PASS

9. Conclusion

The auditing team at BLOCKSTARS TECHNOLOGY was tasked with 1 smart contract from DLTX for a **DEEP** audit evaluation. The team has completed automation testing, manual review of codes and test cases review on the Bricktopians NFT Forge contract. The audit used static, dynamic, and symbolic analysis tools for reviewing each function within the project's contract/s.

The Bricktoforge contract does not contain suicidal instructions, hence it is not vulnerable. It does not contain Call/Suicide; hence it is not prodigal, nor does it have lock vulnerabilities found because the contract cannot receive Ether.

Based on this analysis, the Bricktoforge.sol contract has identified 2 vulnerabilities and 3 informational findings that need to be addressed:

Bricktoforge.sol	**1 HIGH**	**0 MEDIUM**	**1 LOW**	**3 INFORMATIONAL**

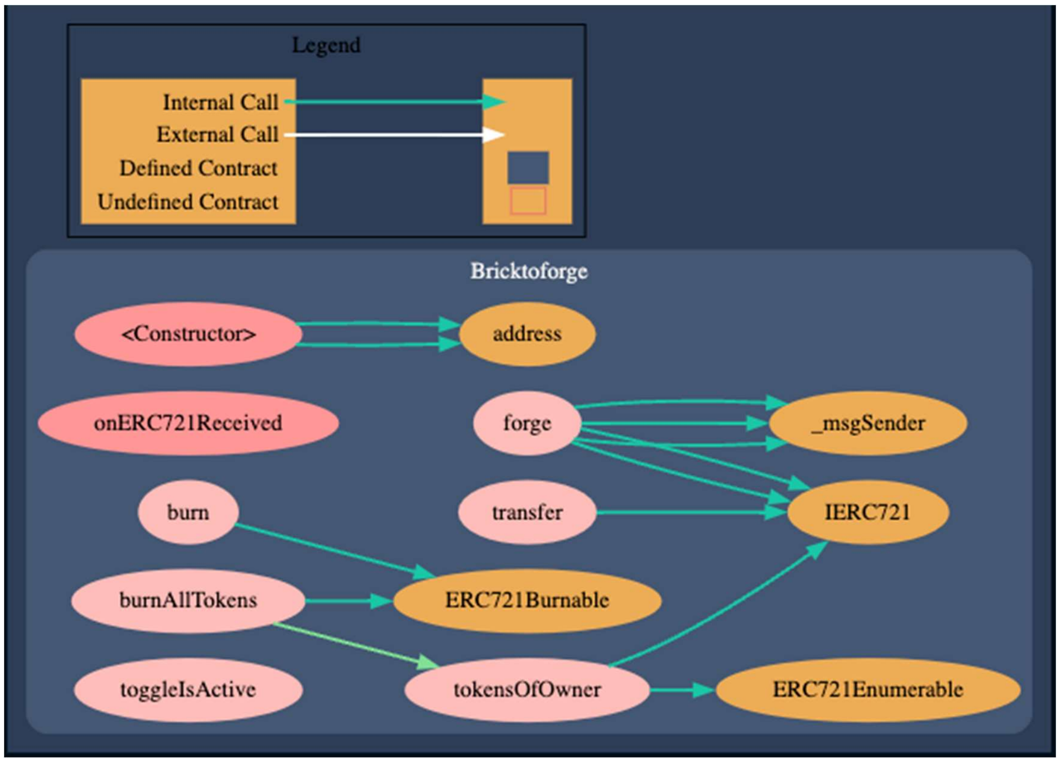
The Bricktoforge.sol contract in its current state is vulnerable to potential attacks, including re-entrancy. Apply the recommended resolutions for each vulnerable function before considering further deployment.

It is strongly advised not to underestimate the potential impact of low-severity vulnerabilities, as intentionally leaving them unaddressed could leave the Bricktopians NFT Forge project vulnerable to exploitation. In order to maintain the security of the project's contract/s, all identified vulnerabilities, regardless of their severity, should be promptly addressed and remediated. Neglecting to do so could result in significant harm to the project's integrity and reputation, and potentially even result in major loss. Therefore, it is imperative that regular security audits are conducted to identify and address any potential vulnerabilities from this point onward.

10. Appendix

10.1. Functional Flow Chart

10.1.1. Bricktoforge.sol


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Bricktoforge
<Constructor>
address
onERC721Received
forge
_msgSender
burn
transfer
IERC721
burnAllTokens
ERC721Burnable
toggleIsActive
tokensOfOwner
ERC721Enumerable