ELVT
Smart Contract Preliminary Report

Project Synopsis

Project Name	ELVT
Platform	Ethereum, Solidity
Github Repo	Not Provided
Deployed Contract	Not Deployed
Total Duration	2 Days

Contract Details

Total Contract(s)	1
Name of Contract(s)	ELVT.sol
Language	Solidity
Commit Hash	Null

ImmuneBytes-ELVT Preliminary Report

Finding

#	Findings	Risk	Status
1	Lack of input validation in constructor	**Low**	Pending
2	Lack of Event Emissions in Configuration Functions	**Low**	Pending
3	ELVT Contract could prefer Role Based Access considering multiple roles in the contract	**Informatory**	Pending
4	Non-standard Parameter Naming Convention	**Informatory**	Pending

ImmuneBytes -ELVT Preliminary Report

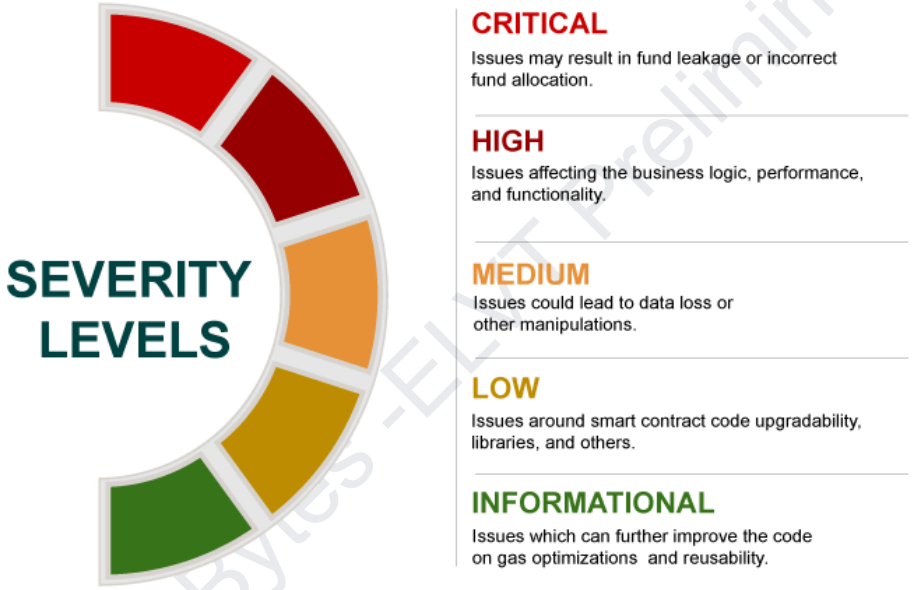
Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis, automated testing, manual code review etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

Security Level Reference

Every issues in this report were assigned a severity level from the following:



ImmuneBytes - ELIT Preliminary Report

Critical Severity Issues

No issues were found

High Severity Issues

No issues were found

Medium Severity Issues

No issues were found

ImmuneBytes -ELVT Preliminary Report

Low Severity Issues

1. **Lack of input validation in constructor**

   Description:

   The constructor of the ELVT contract takes three parameters: **_feeCalculator**, **_treasury**, and **_initialSupply**. While there is a check to ensure that **_feeCalculator** is not the zero address, there are no checks to validate the **_treasury** address and **_initialSupply** values. This oversight can lead to potential issues:

   1. Treasury Address Validation: The **_treasury** address is critical as it receives the initial supply of tokens. Without validation, there is a risk that tokens could be minted to an incorrect or zero address, leading to loss of control over these tokens.
   2. Initial Supply Validation: The **_initialSupply** parameter dictates the total initial tokens minted. Without checks, erroneous values (e.g., excessively high or zero) could be set, which might affect the tokenomics and security of the token.

   Recommendation:

   To mitigate these risks, it is recommended to implement the following validations:

   - Ensure **_treasury** is not the zero address to prevent tokens from being irretrievably locked or minted to a non-retrievable address.
   - Validate that **_initialSupply** is greater than zero to avoid initializing a token with no usable supply, which could render the contract useless.

   Implementing these recommendations will ensure that the contract is initialized with valid parameters, enhancing its security and operational reliability.

ImmuneBytes - ELVT Preliminary Report

2. **Lack of Event Emissions in Configuration Functions**

Description:

Event emissions in smart contracts are crucial for transparency and traceability, allowing off-chain applications and interfaces to react to changes within the contract.

The absence of events in these two functions means that changes to the fee calculator and payment splitter addresses are not easily traceable through external tools or blockchain explorers.

Additionally, this can complicate integration with front-end applications and hinder monitoring and verification of contract state changes by users and external services.

Recommendation:

To enhance transparency and ensure that changes to key contract parameters are trackable, it is recommended to add event emissions in both **setFeeCalculator** and **setPaymentSplitter** functions.

This would enable clients and external applications to monitor updates in real-time and maintain an audit trail of changes.

ImmuneBytes -ELVT Preliminary Report

Informational

1. **ELVT Contract could prefer Role Based Access considering multiple roles in the contract**

   Description:

   The ELVT smart contract incorporates several functionalities, including trading toggles, fee exemptions, blacklist management, and DEX pair definitions. Each of these functionalities significantly impacts the operation and security of the contract.

   Currently, all these administrative functions are governed by a single owner, defined through the **Ownable** modifier from OpenZeppelin.

   This design introduces a single point of failure in the contract's administration. If the owner's private key is compromised, the attacker gains full control over these critical functionalities, potentially leading to damaging actions such as:

   - Blacklisting legitimate users or whitelisting malicious ones.
   - Manipulating fee structures to their advantage.
   - Disabling trading at will to disrupt normal operations.

   Furthermore, such centralized control contradicts the decentralized ethos of blockchain technologies, where distributed control is often more desirable to prevent abuse of power and increase trust in the system's operations.

   Recommendations:

   To mitigate these risks and enhance the contract's security architecture, it is recommended to implement a role-based access control (RBAC) system. OpenZeppelin offers an **AccessControl** module that can be used to assign specific roles to different addresses, each with defined capabilities. Key recommendations include:

   - **Implement AccessControl:** Utilize OpenZeppelin's **AccessControl** to create different roles for various administrative functions. For example:
     - Fee Manager Role: Handles fee-related functionalities.
     - Trading Manager Role: Manages trading toggles and restrictions.
     - Blacklist Manager Role: Oversees the blacklist functionalities.
   - **Multi-Signature Requirements:** For critical functions, require that multiple authorized signatories approve changes, reducing the risk of unilateral detrimental actions.
   - **Decentralized Control:** Spread control among trusted parties within the organization or community to reduce the impact of a single compromised account.

ImmuneBytes - ELVT Preliminary Report

2. **Non-standard Parameter Naming Convention**

Description:

The ELVT smart contract's parameter naming convention does not adhere to the commonly accepted Solidity style guide.

Several function parameters across various methods in the contract utilize non-mixedCase naming, which diverges from the best practices recommended for Solidity coding standards.

Details

The following parameters have been identified as not following the mixedCase naming convention:

- **ELVT.setFeeCalculator(address)._feeCalculator** should ideally be **_feeCalculator**.
- **ELVT.setPaymentSplitter(address)._splitter** should ideally be **_splitter**.
- **ELVT.toggleExcludeFee(address[], bool[])._wallets** should ideally be **_wallets**.
- **ELVT.toggleExcludeTrading(address[], bool[])._wallets** should ideally be **_wallets**.
- **ELVT.toggleDexPairAddresses(address[], bool[])._pairs** should ideally be **_pairs**.
- **ELVT.toggleBlacklist(address[], bool[])._addresses** should ideally be **_addresses**.

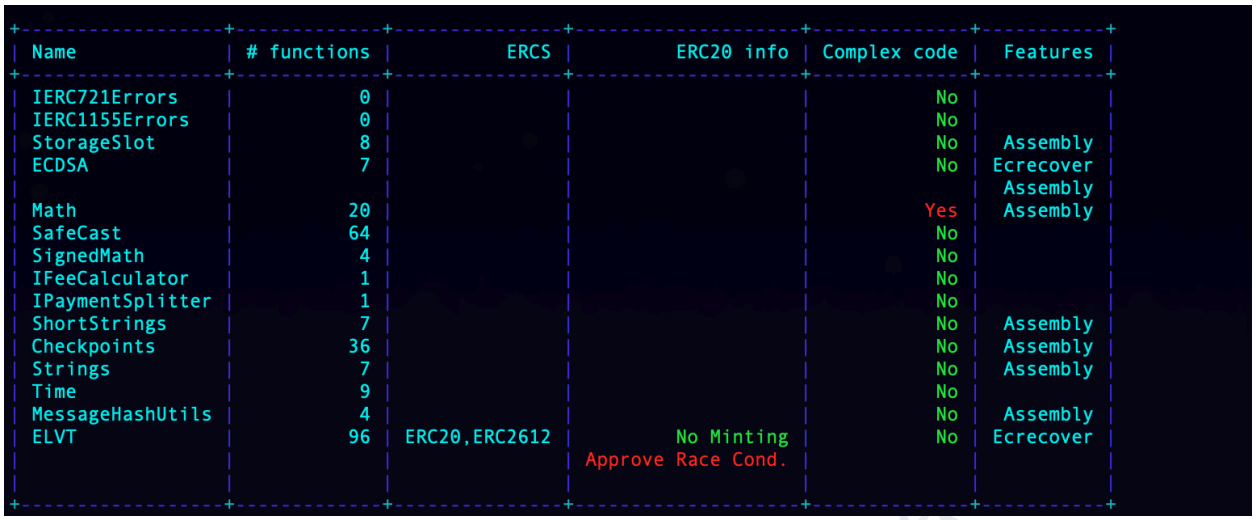
While this issue does not pose a direct security risk, adhering to a consistent and standard naming convention improves code readability and maintainability.

Recommendation:

It is recommended to refactor these parameter names to follow the mixedCase style. This change would make the codebase cleaner and more in line with typical Solidity practices, which emphasize readability and consistency.

ImmuneBytes - ELVT Preliminary Report

Automated Test Results



Name	# functions	ERCS	ERC20 info	Complex code	Features
IERC721Errors	0			No	
IERC1155Errors	0			No	
StorageSlot	8			No	Assembly
ECDSA	7			No	Ecrecover Assembly
Math	20			Yes	Assembly
SafeCast	64			No	
SignedMath	4			No	
IFeeCalculator	1			No	
IPaymentSplitter	1			No	
ShortStrings	7			No	Assembly
Checkpoints	36			No	Assembly
Strings	7			No	Assembly
Time	9			No	
MessageHashUtils	4			No	Assembly
ELVT	96	ERC20,ERC2612	No Minting Approve Race Cond.	No	Ecrecover

ImmuneBytes -ELVT Prelimina

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore, running a bug bounty program complementing this audit is strongly recommended.

Our team does not endorse the platform or its product, nor this audit is investment advice.

Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for code refactoring by the team on critical issues.

ImmuneBytes -ELVT Preliminary Report