



Building the Futuristic **Blockchain Ecosystem**

# SECURITY AUDIT REPORT

**XTM**

# TOKEN OVERVIEW

## Risk Findings

Severity	Found
● High	5
● Medium	1
● Low	0
● Informational	0

## Centralization Risks

Owner Privileges	Description
● Can Owner Set Taxes >25% ?	Detected
● Owner needs to enable trading ?	Yes, owner needs to enable trades
● Can Owner Disable Trades ?	Not Detected
● Can Owner Mint ?	Not Detected
● Can Owner Blacklist ?	Detected
● Can Owner set Max Wallet amount ?	Not Detected
● Can Owner Set Max TX amount ?	Not Detected

# HIGH RISK FINDING

## Trades are disabled by default

Category: **Centralization**

Status: **Open**

Impact: **High**

### Overview:

The contract has been structured such that all trading is disabled by default, necessitating the contract owner's manual intervention to enable trading. This can lead to a situation where, if trades remain disabled, token holders won't be able to buy, sell, or trade their tokens, causing a severe impact on the token's usability and market liquidity.

```
function startTrading() external onlyOwner {  
    tradingAllowed = true;  
}
```

### Suggestion:

To mitigate this risk, it is recommended that trading be enabled before the token presale. This can be achieved by invoking the "startTrading" function or by transferring ownership of the contract to a third-party that has established trust with the community, such as a Certified SAFU developer. This reduces the concentration of power and the potential for malicious actions, thereby promoting a more decentralized and fair environment for all participants.

# HIGH RISK FINDING

## Excessive fees

Category: **Centralization**

Status: **Open**

Impact: **High**

### Overview:

The contract owner is able to set up to 100% tax on buy/sell/transfers

```
function setTransactionRequirements(
    uint256 _liquidity,
    uint256 _marketing,
    uint256 _burn,
    uint256 _development,
    uint256 _total,
    uint256 _sell,
    uint256 _trans
) external onlyOwner {
    liquidityFee = _liquidity;
    marketingFee = _marketing;
    burnFee = _burn;
    developmentFee = _development;
    totalFee = _total;
    sellFee = _sell;
    transferFee = _trans;
    require(
        totalFee <= denominator.div(1) &&
        sellFee <= denominator.div(1) &&
        transferFee <= denominator.div(1),
        "totalFee and sellFee cannot be more than 20%"
    );
}
```

# HIGH RISK FINDING

**Suggestion:**

Ensure that fees are within a reasonable range. Usually 0–10% is suggested as an optimal upper bound for fees.

**Alleviation:**

We need those functions as a countermeasure against bots and the contact is planned to be renounced after launch

# HIGH RISK FINDING

## Blacklisting

**Category:** Centralization

**Status:** Open

**Impact:** High

### Overview:

The contract owner is able to blacklist an arbitrary address. Blacklisted addresses have to pay 90% fee.

```
function setisBot(
  address[] calldata addresses,
  bool _enabled
) external onlyOwner {
  for (uint i = 0; i < addresses.length; i++) {
    isBot[addresses[i]] = _enabled;
  }
}
```

### Suggestion:

Implement a more decentralized method for blacklisting bad actors, such as using “dead blocks” after launch.

### Alleviation:

We need those functions as a countermeasure against bots and the contract is planned to be renounced after launch

# HIGH RISK FINDING

## setting swap threshold to zero

Category: **Logical**

Status: **Open**

Impact: **High**

### Overview:

Setting `_swapThreshold` to zero will disable sell transactions, this is because contract uses `swapThreshold` as the amount of tokens for performing internal swap

```
function setContractSwapSettings(
    uint256 _swapAmount,
    uint256 _swapThreshold,
    uint256 _minTokenAmount
) external onlyOwner {
    swapAmount = _swapAmount;
    swapThreshold = _totalSupply.mul(_swapThreshold).div(uint256(1000000));
    minTokenAmount = _totalSupply.mul(_minTokenAmount).div(uint256(1000000));
}
```

### Suggestion:

Set a lower bound for `swapAmount`, `swapThreshold`

```
function setContractSwapSettings(
    uint256 _swapAmount,
    uint256 _swapThreshold,
    uint256 _minTokenAmount
) external onlyOwner {
    swapAmount = _swapAmount;
    swapThreshold = _totalSupply.mul(_swapThreshold).div(uint256(1000000));
    minTokenAmount = _totalSupply.mul(_minTokenAmount).div(uint256(1000000));
    require(swapAmount >= 1, "Can't set swap amount to 0");
    require(swapThreshold >= 1, "Can't set swap amount to 0");
}
```

# HIGH RISK FINDING

## Burning tokens from the contract

Category: **Logical**

Status: **Open**

Impact: **High**

### Overview:

Contract tries to burn tokens from its own balance if total fee on buy/sell/transfer is more than burnFee.

```
if (burnFee > uint256(0) && getTotalFee(sender, recipient) > burnFee) {  
    _transfer(  
        address(this),  
        address(DEAD),  
        amount.div(denominator).mul(burnFee)  
    );  
}
```

There are two issues in this section of the code:

- 1- \_trnasfer is used for trasnferring tokens to DEAD wallet. This can cause recursive calls to takeFee function which might revert the whole transaction.
- 2- contract is using a poriton of "amount" as the burning amount eventhough there might not be enough tokens in the contract.

### Suggestion:

- use a portion of contract balance for burning tokens:

```
uint256 toBurn = balanceOf(address(this)) * burnFee / denominator;
```

- switch balances instead of using \_transfer:

```
balances[address(this)] -= toBurn;
```

```
balances[DEAD] -= toBurn;
```

```
totalSupply -= toBurn;
```

### Alleviation:

At the moment the contract is set to not burn and it won't bet set to burn, it's not in our plans (however, being the tokens burnt from the contract which is feeexempt, there should be no recursive call to takefee)



# TABLE OF CONTENTS

02	Token Overview	
03	High Risk	
09	Table of Contents	
10	Overview	
11	Contract Details	
12	Audit Methodology	
13	Vulnerabilities Checklist	
14	Risk Classification	
15	Inheritance Trees	
16	Function Details	
19	Testnet Version	
21	Manual Review	
22	High Risk Finding	
28	Medium Risk Finding	
29	About Expelee	
30	Disclaimer	

# OVERVIEW

The Expelee team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analysed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit:

<b>Audit Result</b>	<b>Passed with high risk</b>
<b>KYC Verification</b>	<b>-</b>
<b>Audit Date</b>	<b>2 August 2023</b>

# CONTRACT DETAILS

Token Name: XTIME

Symbol: XTM

Network: Ethereum mainnet

Language: Solidity

Contract Address:

0x159B438b5e2aaD937CbE6A7C7324D2E8f9462B8a

Total Supply: 1,000,000,000,000

Owner's Wallet:

0x8D03cb2DE860Adf5F0dA8C3292b451AF9AF472c6

Deployer's Wallet:

0x8D03cb2DE860Adf5F0dA8C3292b451AF9AF472c6

Testnet.

<https://testnet.bscscan.com/token/0xf532cd3fbe6Bc9Eab990d78c0f9abd25B29be88c>

# AUDIT METHODOLOGY

## Audit Details

Our comprehensive audit report provides a full overview of the audited system's architecture, smart contract codebase, and details on any vulnerabilities found within the system.

## Audit Goals

The audit goal is to ensure that the project is built to protect investors and users, preventing potentially catastrophic vulnerabilities after launch, that lead to scams and rugpulls.

## Code Quality

Our analysis includes both automatic tests and manual code analysis for the following aspects:

- Exploits
- Back-doors
- Vulnerability
- Accuracy
- Readability

## Tools

- DE
- Open Zeppelin
- Code Analyzer
- Solidity Code
- Compiler
- Hardhat

# VULNERABILITY CHECKS

Design Logic	Passed
Compiler warnings	Passed
Private user data leaks	Passed
Timestamps dependence	Passed
Integer overflow and underflow	Passed
Race conditions & reentrancy. Cross-function race conditions	Passed
Possible delays in data delivery	Passed
Oracle calls	Passed
Front Running	Passed
DoS with Revert	Passed
DoS with block gas limit	Passed
Methods execution permissions	Passed
Economy model	Passed
Impact of the exchange rate on the logic	Passed
Malicious event log	Passed
Scoping and declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Cross-function race conditions	Passed
Safe Zeppelin module	Passed

# RISK CLASSIFICATION

When performing smart contract audits, our specialists look for known vulnerabilities as well as logical and access control issues within the code. The exploitation of these issues by malicious actors may cause serious financial damage to projects that failed to get an audit in time. We categorize these vulnerabilities by the following levels:

## High Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

## Medium Risk

Issues on this level are critical to the smart contract's performance/functionality and should be fixed before moving to a live environment.

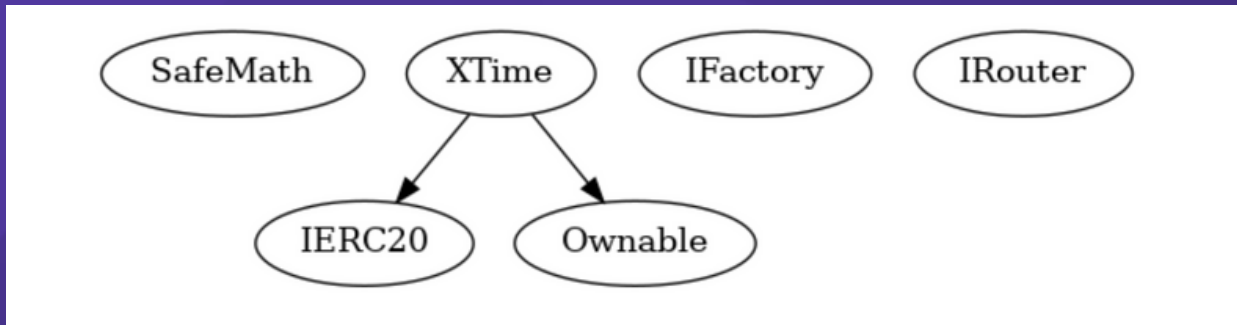
## Low Risk

Issues on this level are minor details and warnings that can remain unfixed.

## Informational

Issues on this level are minor details and warnings that can remain unfixed.

# INHERITANCE TREES
























# FUNCTION DETAILS

```



|  | approve | External ! | 🚫 | NO ! |
|  | transferFrom | External ! | 🚫 | NO ! |
|||||
| **Ownable** | Implementation | |||
|  | <Constructor> | Public ! | 🚫 | NO ! |
|  | isOwner | Public ! | | NO ! |
|  | transferOwnership | Public ! | 🚫 | onlyOwner |
|||||
| **IFactory** | Interface | |||
|  | createPair | External ! | 🚫 | NO ! |
|  | getPair | External ! | | NO ! |
|||||
| **IRouter** | Interface | |||
|  | factory | External ! | | NO ! |
|  | WETH | External ! | | NO ! |
|  | addLiquidityETH | External ! | 🏧 | NO ! |
|  | swapExactTokensForETHSupportingFeeOnTransferTokens | External ! | 🚫 | NO ! |
|||||
| **XTime** | Implementation | IERC20, Ownable |||
|  | <Constructor> | Public ! | 🚫 | Ownable |
|  | <Receive Ether> | External ! | 🏧 | NO ! |
|  | name | Public ! | | NO ! |
|  | symbol | Public ! | | NO ! |
|  | decimals | Public ! | | NO ! |
|  | startTrading | External ! | 🚫 | onlyOwner |
|  | getOwner | External ! | | NO ! |
|  | balanceOf | Public ! | | NO ! |
|  | transfer | Public ! | 🚫 | NO ! |
|  | allowance | Public ! | | NO ! |
|  | setisExempt | External ! | 🚫 | onlyOwner |
|  | approve | Public ! | 🚫 | NO ! |
|  | totalSupply | Public ! | | NO ! |
|  | shouldContractSwap | Internal 🗝️ | | |
|  | setContractSwapSettings | External ! | 🚫 | onlyOwner |
|  | setTransactionRequirements | External ! | 🚫 | onlyOwner |
|  | setTransactionLimits | External ! | 🚫 | onlyOwner |
|  | setInternalAddresses | External ! | 🚫 | onlyOwner |

```

# FUNCTION DETAILS

	└	setisBot		External	!				onlyOwner	
	└	manualSwap		External	!				onlyOwner	
	└	rescueERC20		External	!				onlyOwner	
	└	swapAndLiquify		Private					lockTheSwap	
	└	addLiquidity		Private						
	└	swapTokensForETH		Private						
	└	shouldTakeFee		Internal						
	└	getTotalFee		Internal						
	└	takeFee		Internal						
	└	_transfer		Private						
	└	transferFrom		Public	!				NO	!
	└	_approve		Private						

## ### Legend

	Symbol		Meaning	
	:-----:		-----	
			Function can modify state	
			Function is payable	

# TESTNET VERSION

Adding Liquidity ✓

Tx:

<https://testnet.bscscan.com/tx/0xaffdded465af4c86e10fa05bc40f55dcd114e3063a6c1d0e3750f18ac4dc3eeb>

=====

Buying when excluded from fees ✓

Tx (0% tax):

<https://testnet.bscscan.com/tx/0x594335e3a76d0d11fd89534a0b1ef48e7eebfb046434aa23da078e97c9aad9fc>

=====

Selling when excluded from fees ✓

Tx (0% tax):

<https://testnet.bscscan.com/tx/0x5ad9afa632911fb6acfd7f321eeca229aef95e7ef87f745f8aa6d54314987e1>

=====

Transferring when excluded from fees ✓

Tx (0% tax):

<https://testnet.bscscan.com/tx/0x803458bbc7092f3c98050d44a62d3377341c20556e273adc00d6c6c8c59cf719>

=====

Buying ✓

Tx (0-100% tax):

<https://testnet.bscscan.com/tx/0x5379917ef50e8fd73b638c6452ce16684ce82fa26f0900644552bb1349f9cf28>

# TESTNET VERSION

Selling ✓

Tx (0-100% tax):

<https://testnet.bscscan.com/tx/0x33953163cc10bfb4fa97b818449381bd09485afe57056a6c65e81dd0151d9126>

=====

Transferring ✓

Tx(0% tax):

<https://testnet.bscscan.com/tx/0xdab7d12fe82a29dba23c63bceccd43c0a954d667a5f130b9bc4672da62510a06>

=====

Internal swap (BNB to marketing wallet | reward token to dividend tracker | reward distribution) ✓

Tx:

<https://testnet.bscscan.com/tx/0x33953163cc10bfb4fa97b818449381bd09485afe57056a6c65e81dd0151d9126>

# MANUAL REVIEW

## Severity Criteria

Expelee assesses the severity of disclosed vulnerabilities according to methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories:

High

Medium

Low

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious input handling
- Escalation of privileges
- Arithmetic
- Gas use

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

# HIGH RISK FINDING

## Trades are disabled by default

Category: **Centralization**

Status: **Open**

Impact: **High**

### Overview:

The contract has been structured such that all trading is disabled by default, necessitating the contract owner's manual intervention to enable trading. This can lead to a situation where, if trades remain disabled, token holders won't be able to buy, sell, or trade their tokens, causing a severe impact on the token's usability and market liquidity.

```
function startTrading() external onlyOwner {  
    tradingAllowed = true;  
}
```

### Suggestion:

To mitigate this risk, it is recommended that trading be enabled before the token presale. This can be achieved by invoking the "startTrading" function or by transferring ownership of the contract to a third-party that has established trust with the community, such as a Certified SAFU developer. This reduces the concentration of power and the potential for malicious actions, thereby promoting a more decentralized and fair environment for all participants.

# HIGH RISK FINDING

## Excessive fees

Category: **Centralization**

Status: **Open**

Impact: **High**

### Overview:

The contract owner is able to set up to 100% tax on buy/sell/transfers

```
function setTransactionRequirements(
    uint256 _liquidity,
    uint256 _marketing,
    uint256 _burn,
    uint256 _development,
    uint256 _total,
    uint256 _sell,
    uint256 _trans
) external onlyOwner {
    liquidityFee = _liquidity;
    marketingFee = _marketing;
    burnFee = _burn;
    developmentFee = _development;
    totalFee = _total;
    sellFee = _sell;
    transferFee = _trans;
    require(
        totalFee <= denominator.div(1) &&
        sellFee <= denominator.div(1) &&
        transferFee <= denominator.div(1),
        "totalFee and sellFee cannot be more than 20%"
    );
}
```



# HIGH RISK FINDING

**Suggestion:**

Ensure that fees are within a reasonable range. Usually 0–10% is suggested as an optimal upper bound for fees.

**Alleviation:**

We need those functions as a countermeasure against bots and the contact is planned to be renounced after launch



# HIGH RISK FINDING

## Blacklisting

**Category:** Centralization

**Status:** Open

**Impact:** High

### Overview:

The contract owner is able to blacklist an arbitrary address. Blacklisted addresses have to pay 90% fee.

```
function setisBot(
  address[] calldata addresses,
  bool _enabled
) external onlyOwner {
  for (uint i = 0; i < addresses.length; i++) {
    isBot[addresses[i]] = _enabled;
  }
}
```

### Suggestion:

Implement a more decentralized method for blacklisting bad actors, such as using “dead blocks” after launch.

### Alleviation:

We need those functions as a countermeasure against bots and the contract is planned to be renounced after launch

# HIGH RISK FINDING

## setting swap threshold to zero

Category: **Logical**

Status: **Open**

Impact: **High**

### Overview:

Setting `_swapThreshold` to zero will disable sell transactions, this is because contract uses `swapThreshold` as the amount of tokens for performing internal swap

```
function setContractSwapSettings(
    uint256 _swapAmount,
    uint256 _swapThreshold,
    uint256 _minTokenAmount
) external onlyOwner {
    swapAmount = _swapAmount;
    swapThreshold = _totalSupply.mul(_swapThreshold).div(uint256(1000000));
    minTokenAmount = _totalSupply.mul(_minTokenAmount).div(uint256(1000000));
}
```

### Suggestion:

Set a lower bound for `swapAmount`, `swapThreshold`

```
function setContractSwapSettings(
    uint256 _swapAmount,
    uint256 _swapThreshold,
    uint256 _minTokenAmount
) external onlyOwner {
    swapAmount = _swapAmount;
    swapThreshold = _totalSupply.mul(_swapThreshold).div(uint256(1000000));
    minTokenAmount = _totalSupply.mul(_minTokenAmount).div(uint256(1000000));
    require(swapAmount >= 1, "Can't set swap amount to 0");
    require(swapThreshold >= 1, "Can't set swap amount to 0");
}
```

# HIGH RISK FINDING

## Burning tokens from the contract

Category: **Logical**

Status: **Open**

Impact: **High**

### Overview:

Contract tries to burn tokens from its own balance if total fee on buy/sell/transfer is more than burnFee.

```
if (burnFee > uint256(0) && getTotalFee(sender, recipient) > burnFee) {  
    _transfer(  
        address(this),  
        address(DEAD),  
        amount.div(denominator).mul(burnFee)  
    );  
}
```

There are two issues in this section of the code:

- 1- `_trnasfer` is used for trasnferring tokens to DEAD wallet. This can cause recursive calls to `takeFee` function which might revert the whole transaction.
- 2- contract is using a poriton of "amount" as the burning amount eventhough there might not be enough tokens in the contract.

### Suggestion:

- use a portion of contract balance for burning tokens:

```
uint256 toBurn = balanceOf(address(this)) * burnFee / denominator;
```

- switch balances instead of using `_transfer`:

```
balances[address(this)] -= toBurn;
```

```
balances[DEAD] -= toBurn;
```

```
totalSupply -= toBurn;
```

### Alleviation:

At the moment the contract is set to not burn and it won't bet set to burn, it's not in our plans (however, being the tokens burnt from the contract which is feeexempt, there should be no recursive call to `takefee`)

# MEDIUM RISK FINDING

## EOA receiving LP tokens

**Category:** Logical

**Status:** Open

**Impact:** Medium

### Overview:

an EOA is receiving LP tokens which are generated from auto-liquidity, this causes more centralization power over liquidity pool overtime.

```
function addLiquidity(uint256 tokenAmount, uint256 ETHAmount) private {  
    _approve(address(this), address(router), tokenAmount);  
    router.addLiquidityETH{ value: ETHAmount }(  
        address(this),  
        tokenAmount,  
        0,  
        0,  
        liquidity_receiver,  
        block.timestamp  
    );  
}
```

### Suggestion:

its suggested to burn or lock new LP tokens.

### Alleviation:

We will address this issue by setting the liquidityFee to zero and won't be changed again.

# ABOUT EXPELEE

Expelee is a product-based aspirational Web3 start-up. Coping up with numerous solutions for blockchain security and constructing a Web3 ecosystem from deal making platform to developer hosting open platform, while also developing our own commercial and sustainable blockchain.

 [www.expelee.com](http://www.expelee.com)



expeleeofficial



expelee



Expelee



expelee



expelee\_official



expelee-co

# expelee

Building the Futuristic **Blockchain Ecosystem**

# DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantess against the sale of team tokens or the removal of liquidity by the project audited in this document.

Always do your own research and project yourselves from being scammed. The Expelee team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools.

Under no circumstances did Expelee receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Alway do your own research and protect yourselves from scams.

This document should not be presented as a reason to buy or not buy any particular token. The Expelee team disclaims any liability for the resulting losses.

The logo for Expelee, featuring the word "expelee" in a stylized font. The "ex" is in white, and "pelee" is in orange. The letters are bold and modern.

Building the Futuristic **Blockchain Ecosystem**