Eloin

Smart Contract Security Audit

Prepared by ShellBoxes

Jan 25th, 2024 - Jan 26th, 2024

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Eloin
Version	1.0
Classification	Public

Scope

Contract Name	Contract Address
Eloin	0xdaaa1cdb729eeedda33d03ce89f11fdab307fc43

Re-Audit

Contract Name	Contract Address
Eloin	0x9d4282b4B6Dc457F44F15e39Ebffd4621C6dF246

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Eloin engaged ShellBoxes to conduct a security assessment on the Eloin beginning on Jan 25th, 2024 and ending Jan 26th, 2024. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Eloin

Eloin is a part of booming meme coin era with a vision of Utility as well as Community. Therefore, Eloin is both Utility and Community-driven token. The duo of these two would act as a catalyst for the overall ecosystem and token growth.

Issuer	Eloin
Website	`https://www.eloin.tech`
Type	Solidity Smart Contracts
Whitepaper	LION PAPER
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Eloin implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 2 critical-severity, 1 high-severity, 3 medium-severity, 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Inconsistencies in Total Supply and Tax Fee Structure	CRITICAL	Fixed
SHB.2. Inadequate Handling of Fees in _update Function	CRITICAL	Fixed
SHB.3. Loss of Precision Can Lead To Fee Bypass	HIGH	Fixed
SHB.4. Owner Can Renounce Ownership	MEDIUM	Fixed
SHB.5. Centralized Ownership Control	MEDIUM	Acknowledged
SHB.6. Front-Run Attack	MEDIUM	Acknowledged
SHB.7. Floating Pragma	LOW	Fixed
SHB.8. Missing Value Verification	LOW	Fixed
SHB.9. Missing Address Verification	LOW	Fixed

3 Finding Details

SHB.1 Inconsistencies in Total Supply and Tax Fee Structure

- Severity : CRITICAL
- Likelihood : 3
- Status : Fixed
- Impact : 3

Description:

There's a significant disparity in the token total supply, with both the website and contract implementation asserting 1 trillion tokens, while the whitepaper specifies 100 trillion. Additionally, the tax fee percentage is inconsistently communicated across project documents, further complicating the understanding of the project's financial structure.

Files Affected:

SHB.1.1: Eloin.sol

```
346     uint256 public c = 6
347     uint256 public burnFee = 1;
```

SHB.1.2: Eloin.sol

```
383         _mint(
384             owner(),
385             1000000000000 * 10 ** decimals()
386         );
```

Recommendation:

To address these concerns and bolster community trust, it's crucial to align and maintain uniformity in token total supply, and tax fee percentages across all project documentation and the contract implementation. Consider conducting a thorough review and adjusting the

Eloin contract to adhere to the intended total supply and tax fee percentages outlined in project documentation.

Updates

The Eloin team has addressed the issue by adjusting the values according to the project pitch deck (EloinPaper). The total supply is now set to 1 trillion, and the fee structure has been updated as follows: sellFee = 90, buyFee = 60, burnFee = 10, with division = 1000.

SHB.1.3: Eloin.sol

```
338     uint256 public sellFee = 90;
339     uint256 public buyFee = 60;
340
341     uint256 public burnFee = 10;
342
343     uint16 immutable DIVISOR = 1000;
```

SHB.1.4: Eloin.sol

```
377         _mint(owner(), 1000000000000 * 10 ** decimals());
```

SHB.2 Inadequate Handling of Fees in _update Function

- Severity : CRITICAL
- Status : Fixed
- Likelihood : 3
- Impact : 3

Description:

In the _update function of the Eloin contract, the fee logic poses a critical issue where the taxFee is not transferred to the designated taxWallet account, and the burnAmount is not burned as intended. This implementation allows fees to accumulate in the contract, creating a vulnerability. Moreover, the contract exposes the withdrawToken function that permits the owner to withdraw all token balances, including the accumulated fees, which deviates from the intended business logic.

Files Affected:

SHB.2.1: Eloin.sol

```
412         if(takeFee){
413             uint256 burnAmount = value * burnFee / 100;
414             if(_isSell(from , to)  _isBuy(from) ){
415                 uint256 tax = value * taxFee / 100;
416                 value = value - tax;
417             }
418             value = value - burnAmount;
419         }
```

SHB.2.2: Eloin.sol

```
523     function withdrawToken(IERC20 token) external onlyOwner {
524         token.safeTransfer(owner(), token.balanceOf(address(this)));
525     }
```

Recommendation:

To address this issue, it is imperative to enhance the fee-handling logic within the _update function. Ensure that the taxFee is appropriately transferred to the designated taxWallet account, and the burnAmount is effectively burned. Additionally, reconsider the design of the withdrawToken function to align with the intended business logic, preventing the owner from withdrawing accumulated fees.

Updates

The Eloin team has fixed the issue by transferring the buy and sell fees to the taxWallet and burning the burnAmount using the super._update function.

SHB.2.3: Eloin.sol

```
416             if (tax > 0) {
417                 value = value - tax;
418                 super._update(from, taxWallet, tax);
419             }
420 
421             value = value - burnAmount;
422             super._update(from, address(0), burnAmount);
423         }
424         super._update(from, to, value);
```

SHB.3 Loss of Precision Can Lead To Fee Bypass

- Severity : HIGH
- Likelihood : 2
- Status : Fixed
- Impact : 3

Description:

The _update function calculates two values: the burn amount and the tax amount, using the burnFee and taxFee respectively. The calculation involves multiplying the transfer amount by these fee values and then dividing by 100. However, when processing a small transfer amount, such as 10, with a tax fee of 6%, the result is 60/100, which Solidity rounds down to 0. Consequently, no amount is deducted in such cases.

Files Affected:

SHB.3.1: Eloin.sol

```
412         if(takeFee){
413             uint256 burnAmount = value * burnFee / 100;
414             if(_isSell(from , to)  _isBuy(from) ){
415                 uint256 tax = value * taxFee / 100;
```

```
416                 value = value - tax;
417             }
418             value = value - burnAmount;
419         }
420         super._update(from, to, value);
421     }
```

Recommendation:

It is recommended to either implement a minimum transfer value requirement of 100 or to enhance the precision of the tax and burn fee calculations to accommodate smaller transfer amounts.

Updates

The Eloin team has fixed the issue by increasing the precision of the fees. They changed the divisor from 100 to 1000 and added one extra point of precision in fees.

SHB.4 Owner Can Renounce Ownership

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 1
- Impact : 3

Description:

The contract inherits from the Ownable OpenZeppelin contract, enabling the owner to renounce ownership. Renouncing ownership leads to the contract being left without an owner, effectively disabling any functionality exclusively available to the owner.

Files Affected:

SHB.4.1: Eloin.sol

```
6 import "@openzeppelin/contracts/access/Ownable.sol";
```

SHB.4.2: Eloin.sol

```
343 contract Eloin is ERC20, ERC20Burnable, Ownable {
```

Recommendation:

It is recommended to prevent the owner from invoking the renounceOwnership function and to disable its functionality by overriding it.

Updates

The Eloin team resolved the issue by disabling the renounceOwnership functionality.

SHB.5 Centralized Ownership Control

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 2
- Impact : 2

Description:

The Eloin contract introduces a centralization risk since several critical functions utilize the onlyOwner modifier, consolidating exclusive control under the contract owner. This centralization extends to functions related to fee management, blacklisting, limit updates, withdrawal of Ether and tokens, and other key functionalities. Relying solely on the owner's authority for these operations may compromise the decentralized nature of the contract.

Files Affected:

SHB.5.1: Eloin.sol

```
479     function excludeFromFee(address account) public onlyOwner {
480         isExcludedFromFee[account] = true;
481     }
482
483     function includeInFee(address account) public onlyOwner {
484         isExcludedFromFee[account] = false;
485     }
486
487     function addToBlacklist(address account) public onlyOwner {
488         isBlacklisted[account] = true;
489     }
490
491     function removeFromBlacklist(address account) public onlyOwner {
492         isBlacklisted[account] = false;
493     }
494
495     function setBurnFeePercent(uint256 fee) external onlyOwner {
496         burnFee = fee;
497     }
498
499     function setTaxFeePercent(uint256 fee) external onlyOwner {
500         taxFee = fee;
501     }
502
503     function changeTaxWallet(address account) public onlyOwner {
504         taxWallet = account;
505     }
506
507     function updateBuyLimit(uint256 limit) external onlyOwner {
508         buyLimit = limit;
509     }
510
511     function updateSellLimit(uint256 limit) external onlyOwner {
```

```
512         sellLimit = limit;
513     }
514
515     function updateTimelimit(bool status) external onlyOwner {
516         timeLimit = status;
517     }
518
519     function withdrawEth() external onlyOwner {
520         payable(owner()).transfer(address(this).balance);
521     }
522
523     function withdrawToken(IERC20 token) external onlyOwner {
524         token.safeTransfer(owner(), token.balanceOf(address(this)));
525     }
```

Recommendation:

To address the centralization risk, consider implementing a multi-signature or governance mechanism that involves multiple authorized parties in decision-making processes. This approach distributes control and reduces reliance on a single owner.

Updates

The Eloin team has acknowledged the issue, stating that they will work with a multi-signature wallet for contract ownership control.

SHB.6 Front-Run Attack

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The Eloin contract exhibits a vulnerability where the owner can front-run user transfer transactions, potentially manipulating fees in their favor. This vulnerability allows the owner to preemptively adjust fees during user transactions, compromising the fairness and transparency of the fee structure.

Files Affected:

SHB.6.1: Eloin.sol

```
495     function setBurnFeePercent(uint256 fee) external onlyOwner {
496         burnFee = fee;
497     }
498
499     function setTaxFeePercent(uint256 fee) external onlyOwner {
500         taxFee = fee;
501     }
```

Recommendation:

To mitigate this risk, it is crucial to implement measures that prevent the owner from adjusting fees opportunistically during user transfer transactions. Consider incorporating safeguards such as time-lock functionalities, access control mechanisms, or utilizing multi-signature verification.

Updates

The Eloin team has acknowledged the issue, stating that they will work with a multi-signature wallet for the owner, which will reduce the front-run risk.

SHB.7 Floating Pragma

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The Eloin contract uses a floating Solidity pragma of 0.8.22, indicating compatibility with any compiler version from 0.8.19 (inclusive) up to, but not including, version 0.9.0. This flexibility could potentially introduce unexpected behavior if the contracts are compiled with a newer compiler version that includes breaking changes.

Files Affected:

SHB.7.1: Eloin.sol

```
2 pragma solidity ^0.8.22;
```

Recommendation:

It is generally recommended to lock the pragma statement to a specific Solidity compiler version to ensure consistent behavior across different compiler versions. To achieve this, consider removing the caret (^) from the pragma statement and specifying a fixed version, such as pragma solidity 0.8.22.

Updates

The Eloin team has resolved this issue by fixing the pragma version and locking it to 0.8.22.

SHB.8 Missing Value Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Certain functions in the Eloin contract lack necessary value safety checks on their arguments. Therefore, the contract fails to ensure that all values provided are greater than 0. Additionally, there is no validation in place to verify that fee values for both tax and burn fall within the valid percentage range of 0 to 100. This absence of value verification poses a potential risk to the contract's integrity and security.

Files Affected:

SHB.8.1: Eloin.sol

```
495     function setBurnFeePercent(uint256 fee) external onlyOwner {
496         burnFee = fee;
497     }
498
499     function setTaxFeePercent(uint256 fee) external onlyOwner {
500         taxFee = fee;
501     }
```

SHB.8.2: Eloin.sol

```
507     function updateBuyLimit(uint256 limit) external onlyOwner {
508         buyLimit = limit;
509     }
510
511     function updateSellLimit(uint256 limit) external onlyOwner {
512         sellLimit = limit;
513     }
```

Recommendation:

To address this issue, we recommend implementing thorough value verification checks for the arguments in the affected functions. Use require statements to ensure that all input values are greater than 0, and validate that fee values for tax and burn are within the acceptable percentage range of 0 to 100.

Updates

The Eloin team has fixed the issue by adding value checks for all setter functions, using require statements to ensure proper validation.

SHB.9 Missing Address Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

The changeTaxWallet function in the Eloin contract lacks a critical address verification check. Currently, it allows the taxWallet to be set to any address, including address(0). This absence of address validation poses a potential risk, as setting the taxWallet to address(0) may result in unintended consequences or vulnerabilities. Additionally, it's important to note that the contract constructor initializes the contract with an initialOwner address. This initialization process could introduce similar vulnerabilities if the provided initial owner address is address(0).

Files Affected:

SHB.9.1: Eloin.sol

```
364     constructor(
365         address initialOwner
366     ) ERC20("Eloin", "ELOIN") Ownable(initialOwner) {
```

SHB.9.2: Eloin.sol

```
503     function changeTaxWallet(address account) public onlyOwner {
504         taxWallet = account;
505     }
```

Recommendation:

To mitigate this issue, it is essential to incorporate a check in the changeTaxWallet function to ensure that the provided account address is not equal to address(0). Similarly, consider adding a check in the constructor to validate that the initialOwner address is not address(0).

Updates

The Eloin team resolved the issue by implementing our recommendation and incorporating zero address checks

4 Best Practices

BP.1 Remove Dead Code

Description:

Enhance code cleanliness and maintainability in the Eloin contract by conducting a thorough review and removal of any dead or commented code snippets. This practice promotes improved readability, reduces confusion, and ensures a streamlined and more maintainable smart contract.

Files Affected:

BP.1.1: Eloin.sol

```
9 // pragma solidity >=0.5.0;
```

BP.1.2: Eloin.sol

```
42 // pragma solidity >=0.5.0;
```

BP.1.3: Eloin.sol

```
143 // pragma solidity >=0.6.2;
```

BP.1.4: Eloin.sol

```
294 // pragma solidity >=0.6.2;
```

BP.1.5: Eloin.sol

```
367         // IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(0
                ↪ x10ED43C718714eb63d5aA57B78B54704E256024E); // mainnet
```

Status - Fixed

BP.2 Optimize Fee-Related Validation Logic

Description:

Streamline the codebase in the Eloin contract by optimizing the fee-related validation logic. The private functions _validateTransfer and _validateTime are currently conditioned on the takeFee variable. To improve efficiency, consider moving this conditional check into the _update function. By doing so, you can eliminate redundant checks in the individual functions, reducing overhead and enhancing overall contract performance.

Files Affected:

BP.2.1: Eloin.sol

```
400         bool takeFee = true;
401
402         //if any account belongs to isExcludedFromFee account then remove
                ↪  the fee
403         if (isExcludedFromFee[from] || isExcludedFromFee[to]) {
404             takeFee = false;
405         }
406
407         _validateTransfer(from, to, value, takeFee);
408         if (timeLimit) {
409             _validateTime(from, to, takeFee);
410         }
```

BP.2.2: Eloin.sol

```
446     function _validateTransfer(
447         address sender,
448         address recipient,
449         uint256 amount,
450         bool takeFee
451     ) private view {
```

```
452         // Excluded addresses don't have limits
453         if (takeFee) {
454             if (_isBuy(sender) && buyLimit != 0) {
```

BP.2.3: Eloin.sol

```
423     function _validateTime(
424         address sender,
425         address recipient,
426         bool takeFee
427     ) private {
428         // Excluded addresses don't have time limits
429         if (takeFee) {
430             if (_isBuy(sender)) {
```

Status - Fixed

BP.3 Remove Unnecessary ETH Handling Functions

Description:

Enhance the simplicity and clarity of the Eloin contract by removing unnecessary Ethereum (ETH) handling functions. The contract does not anticipate receiving funds, so the receive function and the withdrawEth function, which facilitates fund withdrawal, are redundant. Eliminating these functions streamlines the contract, reducing unnecessary complexity and ensuring that the codebase aligns more closely with the intended functionality of the Eloin project.

Files Affected:

BP.3.1: Eloin.sol

```
477     receive() external payable {}
```

BP.3.2: Eloin.sol

```
519     function withdrawEth() external onlyOwner {
520         payable(owner()).transfer(address(this).balance);
521     }
```

Status - Fixed

BP.4 Optimization In Code

Description:

- In the contract's constructor, the uniswapV2 router object is initially assigned to the _uniswapV2Router variable, followed by a redundant assignment to the uniswapV2Router variable. It would be more efficient to directly assign the value of the uniswapV2 router to the uniswapV2Router variable from the outset, thereby eliminating the need for the initial assignment to _uniswapV2Router.
- The excludeFromFee and includeInFee functions have the potential to be consolidated into a single function. This unified function would accept a boolean state (true or false) and directly apply this state to the relevant variable. Similarly, the addToBlacklist and removeFromBlacklist functions could be merged in the same manner, streamlining the process of managing blacklist statuses.

Files Affected:

BP.4.1: Eloin.sol

```
368         IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
369             0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3
370         ); // testnet
```

```
371
372         // Create a uniswap pair for this new token
373         uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
374             .createPair(address(this), _uniswapV2Router.WETH());
375
376         // set the rest of the contract variables
377         uniswapV2Router = _uniswapV2Router;
```

BP.4.2: Eloin.sol

```
479     function excludeFromFee(address account) public onlyOwner {
480         isExcludedFromFee[account] = true;
481     }
482
483     function includeInFee(address account) public onlyOwner {
484         isExcludedFromFee[account] = false;
485     }
```

BP.4.3: Eloin.sol

```
487     function addToBlacklist(address account) public onlyOwner {
488         isBlacklisted[account] = true;
489     }
490
491     function removeFromBlacklist(address account) public onlyOwner {
492         isBlacklisted[account] = false;
493     }
```

Status - Partially Fixed

5 Conclusion

In this audit, we examined the design and implementation of Eloin contract and discovered several issues of varying severity. Eloin team addressed 7 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Eloin Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

6 Scope Files

6.1 Audit

Files	MD5 Hash
Eloin.sol	b79c23ae84dc883bc655f5af6ddb29cd

6.2 Re-Audit

Files	MD5 Hash
Eloin.sol	bc8d78ce98c946c8c515e92863bca267

7 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com