BLOCKSEC

Security Audit

Report for Puffer Institutional Contracts

Date: March 19, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Puffer
Target	Puffer Institutional Contracts

Version History

Version	Date	Description
1.0	March 19, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-03-19 07:23:03+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of Puffer Institutional Contracts[1] of Puffer. Note that we only focus on the contracts in the src/ folder, and other contracts and source code files in the repository are out of scope for this audit. The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process.

Project	Version	Commit Hash
Puffer Institutional Contracts	Version 1	7ed9c69c82db0374418afb1f04e5402f45d23462
	Version 2	cceb9737839011b4f6c0e4d2c31bbd37413e6d83

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.

[1] https://github.com/PufferFinance/puffer-institutional

- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we have **three** recommendations and **six** notes.

- Recommendation: 3
- Note: 6

ID	Severity	Description	Category	Status
1	-	Check the `shareTokenName` and `shareTokenSymbol` variables	Recommendation	Confirmed
2	-	Unused immutable value in the factory contract	Recommendation	Fixed
3	-	Check the length of the array variables of the `completeQueuedWithdrawals()` function	Recommendation	Confirmed
4	-	Design of deposits, withdrawals, and transfers functionality	Note	-
5	-	Potential centralization risk	Note	-
6	-	Trusted parties that own the validator keys	Note	-
7	-	Access control is aligned with function annotations	Note	-
8	-	The `implementation` variable in `createVault()` function	Note	-
9	-	The update of variables `nonRestakedValidatorsETH` and `restakedValidatorsETH`	Note	-

The details are provided in the following sections.

2.1 Additional Recommendation

2.1.1 Check the `shareTokenName` and `shareTokenSymbol` variables

Status Confirmed

Introduced by `Version 1`

Description The `createVault()` function receives the variable `shareTokenName` and `shareTokenSymbol` to create different vaults. However, if the values of `shareTokenName` and `shareTokenSymbol` are the same each time of creating a new vault, different vaults will have the same tokenName and tokenSymbol, which may result in potential confusions to users.

```
52  function createVault(
53      address admin,
54      address implementation,
55      bytes32 salt,
56      string calldata shareTokenName,
57      string calldata shareTokenSymbol
58  ) external onlyPufferMultisig returns (address, address) {
59      require(admin != address(0), ZeroAddress());
```

```
60      require(implementation != address(0), ZeroAddress());
61
62      AccessManager accessManager = new AccessManager(admin);
63
64      address vault = address(
65          Create2.deploy({
66              amount: 0,
67              salt: salt,
68              bytecode: abi.encodePacked(
69                  type(ERC1967Proxy).creationCode,
70                  abi.encode(
71                      implementation,
72                      abi.encodeCall(
73                          InstitutionalVault.initialize, (address(accessManager), shareTokenName,
                                shareTokenSymbol)
74                      )
75                  )
76              )
77          })
78      );
79
80      vaults.push(vault);
81
82      emit VaultCreated(vault, address(accessManager), salt);
83
84      return (address(accessManager), vault);
85  }
```

Listing 2.1: InstitutionalFactory.sol

Suggestion Make sure each vault has a different tokenName and tokenSymbol.

Feedback from the Project This check would introduce unnecessary code complexity.

2.1.2 Unused immutable value in the factory contract

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The immutable value WETH is not used in the contract, which should be removed.

```
18  IERC20 public immutable WETH;
```

Listing 2.2: InstitutionalFactory.sol

Suggestion Remove the unused variable.

2.1.3 Check the length of the array variables of the `completeQueuedWithdrawals()` function

Status Confirmed

Introduced by `Version 1`

Description In the `completeQueuedWithdrawals()` function, the length of the tokens array is 1. If the lengths of the `withdrawals` and `receiveAsTokens` parameters are not equal to 1, the transaction will be reverted.

```
315   function completeQueuedWithdrawals(
316       IDelegationManagerTypes.Withdrawal[] calldata withdrawals,
317       bool[] calldata receiveAsTokens
318   ) external virtual restricted {
319       IERC20[][] memory tokens = new IERC20[][](1);
320       tokens[0] = new IERC20[](1);
321       tokens[0][0] = IERC20(address(0));
322
323       EIGEN_DELEGATION_MANAGER.completeQueuedWithdrawals({
324           withdrawals: withdrawals,
325           tokens: tokens,
326           receiveAsTokens: receiveAsTokens
327       });
328   }
```

Listing 2.3: InstitutionalVault.sol

Suggestion Check the length of the two arrays of parameters `withdrawals` and `receiveAsTokens`, and make sure the length is equal to 1.

2.2 Note

2.2.1 Design of deposits, withdrawals, and transfers functionality

Introduced by `Version 1`

Description Currently, all the deposits and withdrawals are permissioned and controlled by the institution. Meanwhile, the share tokens are non-transferable. Thus, the vulnerabilities (e.g., inflation attack, donation attack) in `ERC-4626` are not applicable in this contract.

2.2.2 Potential centralization risk

Introduced by `Version 1`

Description Several protocol roles could conduct privileged operations, which introduces potential centralization risks. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.2.3 Trusted parties that own the validator keys

Introduced by `Version 1`

Description There is a public known issue that a malicious party could front-run a staker's deposit call to the Beacon chain deposit contract by depositing 1 ETH into the Beacon chain deposit contract with a specified party-controlled withdrawal credentials.

In this case, Puffer Institutional Vault's deposit transaction would be successfully processed but the withdrawal credentials provided by the malicious party will not be overwritten.

The end state is a validator managing 1 ETH of the party's funds and 32 ETH of Puffer Institutional users' funds, fully controlled and withdrawable by the party. Thus, the parties that own the validator keys in the Puffer Institutional Vault must be trusted.

2.2.4 Access control is aligned with function annotations

Introduced by Version 1

Description The audited contracts, utilizes OpenZepplin's `AccessManagedUpgradeable` library to manage access controls. During this audit, we assume that the access control is aligned with function annotations.

2.2.5 The `implementation` variable in `createVault()` function

Introduced by Version 1

Description The `createVault()` function currently receives the `implementation` variable to create the vault. This `implementation` variable may not always be the audited contract (i.e., `InstitutionalVault`) in this report. In the future, the project may use a upgraded implementation logic contract to create new Institutional Vault.

```
52  function createVault(
53      address admin,
54      address implementation,
55      bytes32 salt,
56      string calldata shareTokenName,
57      string calldata shareTokenSymbol
58  ) external onlyPufferMultisig returns (address, address) {
59      require(admin != address(0), ZeroAddress());
60      require(implementation != address(0), ZeroAddress());
61
62      AccessManager accessManager = new AccessManager(admin);
63
64      address vault = address(
65          Create2.deploy({
66              amount: 0,
67              salt: salt,
68              bytecode: abi.encodePacked(
69                  type(ERC1967Proxy).creationCode,
70                  abi.encode(
71                      implementation,
72                      abi.encodeCall(
73                          InstitutionalVault.initialize, (address(accessManager), shareTokenName,
                                shareTokenSymbol)
74                      )
75                  )
76              )
77          })
78      );
79
80      vaults.push(vault);
81
```

```
82        emit VaultCreated(vault, address(accessManager), salt);
83
84        return (address(accessManager), vault);
85    }
```

Listing 2.4: InstitutionalFactory.sol

2.2.6 The update of variables `nonRestakedValidatorsETH` and `restakedValidatorsETH`

Introduced by `Version 1`

Description The two variables `_getVaultStorage().restakedValidatorsETH` and `_getVaultStorage().nonRestakedValidatorsETH` can influence the calculation of share values. These two variables are not updated within the `completeQueuedWithdrawals()` function and will be updated manually with function `setValidatorsETH()`. Thus, the share values may not be precise. The project promises to address the exchange rate accuracy (precision of the share values) when the logic and design of the protocol is updated.

```
235   function startRestakingValidators(
236       bytes[] calldata pubKeys,
237       bytes[] calldata signatures,
238       bytes32[] calldata depositDataRoots
239   ) external virtual restricted {
240       require(pubKeys.length == signatures.length && pubKeys.length == depositDataRoots.length,
              InvalidInput());
241
242       _unwrapWETH(32 ether * pubKeys.length);
243
244       for (uint256 i = 0; i < pubKeys.length; i++) {
245           EIGEN_POD_MANAGER.stake{ value: 32 ether }(pubKeys[i], signatures[i], depositDataRoots[
                i]);
246           emit StartedRestakingValidator(pubKeys[i], depositDataRoots[i]);
247       }
248
249       _getVaultStorage().restakedValidatorsETH += uint128(pubKeys.length * 32 ether);
250
251       emit RestakedValidatorsETHUpdated(_getVaultStorage().restakedValidatorsETH);
252   }
```

Listing 2.5: InstitutionalVault.sol

```
260   function startNonRestakingValidators(
261       bytes[] calldata pubKeys,
262       bytes[] calldata signatures,
263       bytes32[] calldata depositDataRoots
264   ) external virtual restricted {
265       require(pubKeys.length == signatures.length && pubKeys.length == depositDataRoots.length,
              InvalidInput());
266
267       bytes memory withdrawalCredentials = getWithdrawalCredentials();
268
```

```
269         _unwrapWETH(32 ether * pubKeys.length);
270
271         for (uint256 i = 0; i < pubKeys.length; i++) {
272             BEACON_DEPOSIT_CONTRACT.deposit{ value: 32 ether }(
273                 pubKeys[i], withdrawalCredentials, signatures[i], depositDataRoots[i]
274             );
275             emit StartedNonRestakingValidator(pubKeys[i], depositDataRoots[i]);
276         }
277
278         _getVaultStorage().nonRestakedValidatorsETH += uint128(pubKeys.length * 32 ether);
279
280         emit NonRestakedValidatorsETHUpdated(_getVaultStorage().nonRestakedValidatorsETH);
281     }
```

Listing 2.6: InstitutionalVault.sol

```
359     function setValidatorsETH(uint128 restakedValidatorsETH, uint128 nonRestakedValidatorsETH)
360         external
361         virtual
362         restricted
363     {
364         Storage storage $ = _getVaultStorage();
365         $.restakedValidatorsETH = restakedValidatorsETH;
366         $.nonRestakedValidatorsETH = nonRestakedValidatorsETH;
367         emit RestakedValidatorsETHUpdated(restakedValidatorsETH);
368         emit NonRestakedValidatorsETHUpdated(nonRestakedValidatorsETH);
369     }
```

Listing 2.7: InstitutionalVault.sol


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS