Puffer Finance UniFi

September 2024

Made with ❤ by the following Creed authors: Shayan Eskandari and Dominik Muhs

Table of Contents

Executive Summary

This report presents the results of our engagement with Puffer Finance to review the first milestone of the UniFi smart contracts, which is the operator and validator registration. The review was conducted over one week, from September 10, 2024 to September 15, 2024 by Shayan Eskandari and Dominik Muhs. A total of 10 person-days were spent. Additionally, two person-days were spent reviewing the fixes and updating the report.

Most notably, we have identified one critical issue regarding the deregistration process and three medium-severity issues. All the findings were confirmed to have been fixed.

The codebase is still under active development, and as additional functionality is introduced, we recommend regular follow-up reviews aligned with implementation milestones.

Scope and Objectives

Our review focused on the commit hash `e4444a2934cc7c554287cb0a3aceaa3469ce39b1`.

The focus of this audit is the registration phase of the Puffer UniFi AVS, with the following files in the scope:

- `l1-contracts/src/UniFiAVSManager.sol`
- `l1-contracts/src/UniFiAVSManagerStorage.sol`

Together with the Puffer Finance team, we identified the following priorities for our review:

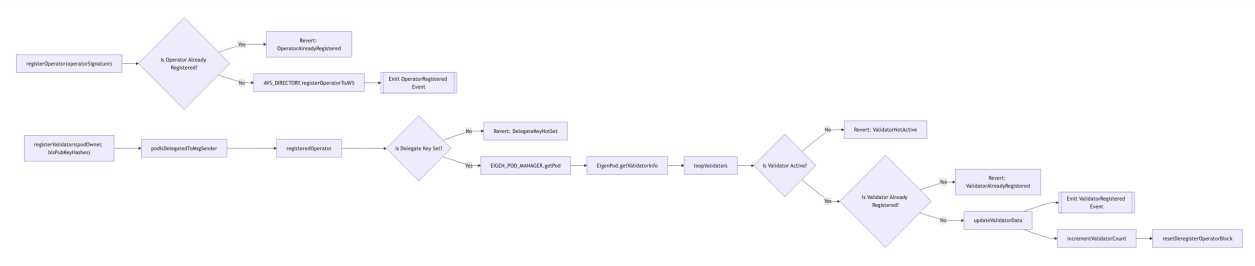
- Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
- Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Security Field Guide, and the ones outlined in the EEA EthTrust Security Levels Specification.

Additionally, the reviewed fixes are contained in pull-request #25 at the following commit hash: `edbc19a856d63a44e614a728d96e95298b6f68ef`.

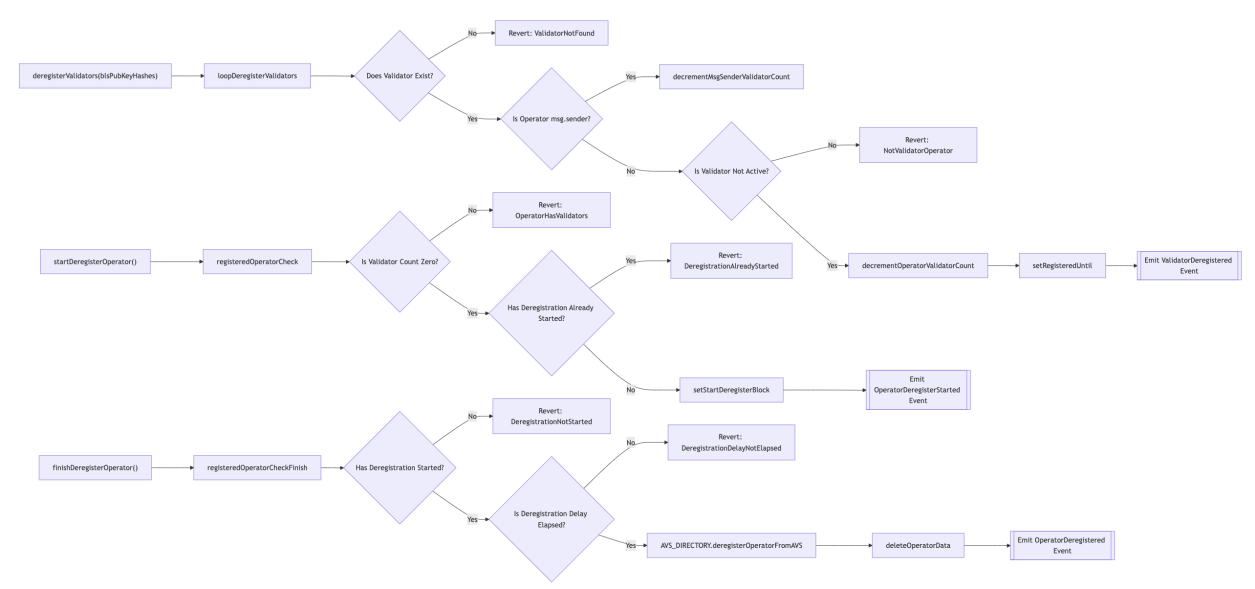
Audit Artifacts

The target repository's documentation already contained many flow charts and sequence diagrams of the system in great detail. Here are some of the diagrams that were created during the audit:

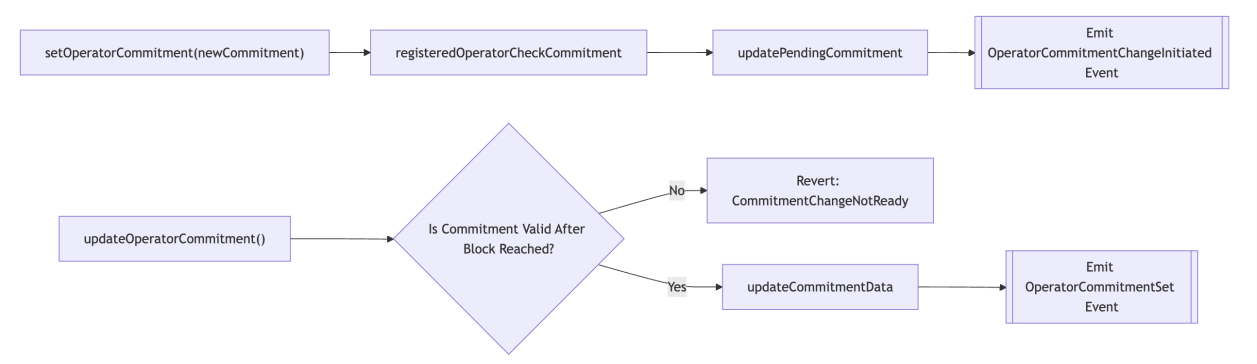
Registration Flow



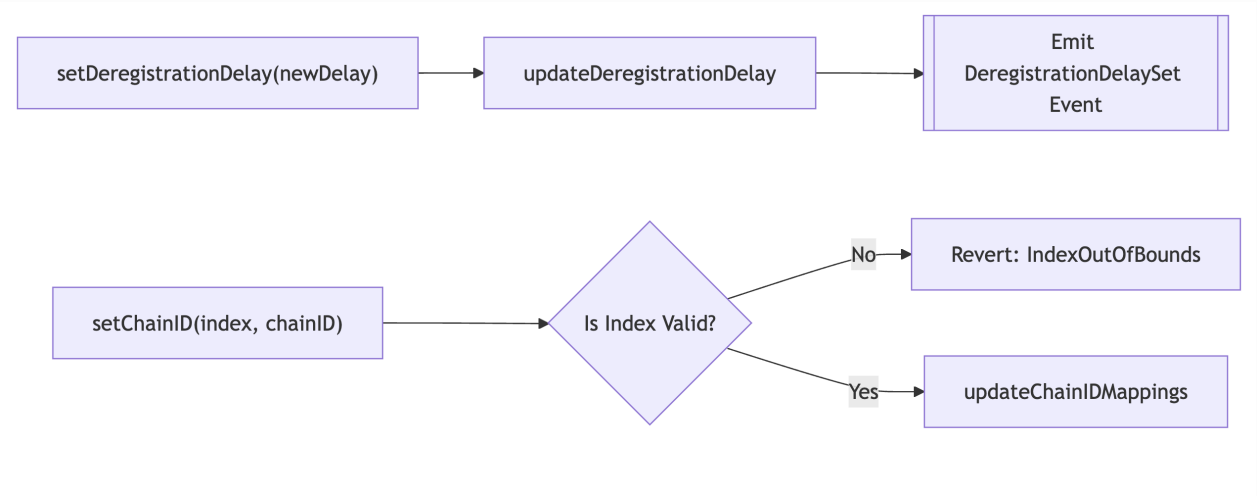
Deregistration Flow



Setting Commitment Flow



Setters Flow



Findings

Critical Prevent deregistering validators and also make deregistering operators with active validators possible

Fixed

> Fixed by only allowing the operator to deregister their validators and checking `registeredUntil` to prevent state changes on duplicate pubkeys in https://github.com/PufferFinance/UniFi/pull/25

The function `deregisterValidators` is not gated and anyone can call to deregister `!=VALIDATOR_STATUS.ACTIVE` validators of any operators.

Attack Scenario

1. Operator has 2 validators [pubkey1, pubkey2] (this attack can work for 2+ validators as well)
2. if at least one of the validators becomes Inactive or Withdrawan, then anyone can call `deregisterValidators`
3. A malicious actor (or a mistake by an operator) calls `deregisterValidators([pubkey1, pubkey1]` (repetition of the pubkey of the inactive validator)
4. `deregisterValidators` will iterate through the `blsPubKeyHashes` which are the same pubkey, and increase `msgSenderValidatorCount` to the length of the array (in the case of operator == msg.sender, then this happens inside the loop (`$.operators[operator].validatorCount -= 1;`)
5. upon finishing the iteration, the value of `operators[msg.sender].validatorCount` would be 0

Effect of the attack

- The operator can no longer deregister their registered validator (as the following line will underflow and revert)

l1-contracts/src/UniFiAVSManager.sol

```
200     $.operators[msg.sender].validatorCount -= uint128(msgSenderValidatorCount);
201 }
```

- The Operator can now call `startDeregisterOperator` and deregister (as it passes the validator count check) even though they have active validators

l1-contracts/src/UniFiAVSManager.sol

```
212 if (operator.validatorCount > 0) {
213     revert OperatorHasValidators();
214 }
```

Recommendation

Use a set for the pubkeys or prevent duplicate pubkeys to be iterated through. Overall recommendation is to revisit the deregistration flow and possibly add more check to `deregisterValidators`

Medium Off-by-one error in `bitmapToChainIDs`

Fixed

> Fixed in https://github.com/PufferFinance/UniFi/pull/25

The function `bitmapToChainIDs` takes a bitmap of type uint256 as input and returns an array of uint256 chain IDs corresponding to the set bits in the bitmap. The loop runs from i = 1 to i < 255. This loop effectively iterates from 1 to 254, missing the bit at index 255. If the 255th bit is set in the bitmap, it will not be checked or processed, likely an unintended behavior.

l1-contracts/src/UniFiAVSManager.sol

```
366 for (uint8 i = 1; i < 255; i++) {
367     if ((bitmap & (1 << i)) != 0) {
368         result[count] = $.bitmapIndexToChainId[i];
369         count++;
370     }
371 }
```

Recommendation

The loop should iterate up to i <= 255 to cover all bits of a uint256 integer.

Medium Initial deregistration delay allows immediate exits

Acknowledged

> The Puffer team added the `deregistrationDelay` to the constructor, but it is initially set to 0. Here's their comment on this decision: At launch, the team will not use the deregistration delay, as the plan involves only operator registration and excludes active commitments and validations.

The `deregistrationDelay` is not set in the constructor, as seen here:

l1-contracts/src/UniFiAVSManager.sol

```
69 constructor(
70     IEigenPodManager eigenPodManager,
71     IDelegationManager eigenDelegationManager,
72     IAVSDirectory avsDirectory
73 ) {
74     EIGEN_POD_MANAGER = eigenPodManager;
75     EIGEN_DELEGATION_MANAGER = eigenDelegationManager;
76     AVS_DIRECTORY = IAVSDirectoryExtended(address(avsDirectory));
77     _disableInitializers();
78 }
```

As a result, it remains at its default value of zero until the DAO explicitly updates it:

l1-contracts/src/UniFiAVSManager.sol

```
291 /**
292  * @inheritdoc IUniFiAVSManager
293  * @dev Restricted to the DAO
294  */
295 function setDeregistrationDelay(uint64 newDelay) external restricted {
296     UniFiAVSStorage storage $ = _getUniFiAVSManagerStorage();
297     uint64 oldDelay = $.deregistrationDelay;
298     $.deregistrationDelay = newDelay;
299     emit DeregistrationDelaySet(oldDelay, newDelay);
300 }
```

This default value of zero permits immediate validator exits, which could compromise the security of the validator commitment process:

l1-contracts/src/UniFiAVSManager.sol

```
195 validator.registeredUntil = uint64(block.number) + uint64(deregistrationDelay);
```

Recommendation

We recommend setting the `deregistrationDelay` in the constructor with a reasonable initial value.

Medium Inconsistency in handling matured `pendingCommitment` & `activeCommitment`

Fixed

> Fixed in https://github.com/PufferFinance/UniFi/pull/25. All functions now return the matured commitment (even though the update function has not been called explicitly)

Any operator can start updating their commitment by calling `setOperatorCommitment` first, waiting for `deregistrationDelay`, and then calling `updateOperatorCommitment` to update their commitment on the chain. The current implementation has some getter functions to get the operator or validators that behave differently if the operator has a matured pending commitment but has not updated the commitment on the chain yet.

Examples

`isValidatorInChainId` does not consider any pending commitment information and does its checks on the "old" commitment:

l1-contracts/src/UniFiAVSManager.sol

```
409 OperatorCommitment memory activeCommitment = operator.commitment;
```

However, `_getOperator`, will return the pending commitment if the maturity is reached but the update function has not been called yet.

l1-contracts/src/UniFiAVSManager.sol

```
425 OperatorCommitment memory activeCommitment = operatorData.commitment;
426 if (operatorData.commitmentValidAfter != 0 && block.number >= operatorData.commi…
427     activeCommitment = operatorData.pendingCommitment;
428 }
```

`_getValidator` behaves similarly to `_getOperator`.

Recommendation

Use the same behavior for updated commitment. Either enforce the call to `updateOperatorCommitment` on maturity or use the matured pending commitment on all getter functions. The fix depends on the off-chain use of these getters and any possible race condition that might result from these implicit updates in the returned values.

Minor Sanity check on setting chainID

Fixed

> Mitigated by emitting an event on the chainID bitmap update. `emit ChainIDSet(index, chainID);`

Given that the commitments directly use the chainID bitmap (`isValidatorInChainId`), it is advised to ensure the chainIDs that are already set won't be silently overwritten.

l1-contracts/src/UniFiAVSManager.sol

```
306 function setChainID(uint8 index, uint256 chainID) external restricted {
307     if (index == 0) revert IndexOutOfBounds();
308
309     UniFiAVSStorage storage $ = _getUniFiAVSManagerStorage();
310     $.bitmapIndexToChainId[index] = chainID;
311     $.chainIdToBitmapIndex[chainID] = index;
312 }
```

Recommendation

Add checks to prevent chainIDs from being overwritten by mistake. Ideally, there should be more sanity checks on the critical variable setters. Possibly emit an event for any new chainIDs (or changes). In that case, the operators can be notified to update their commitment on the chain.

Minor Unnecessary and redundant uint casting

Fixed

Throughout the code base, many uint casts are not necessary and can be removed.

`deregistrationDelay` is defined as uint64

```
l1-contracts/src/UniFiAVSManagerStorage.sol

17    uint64 deregistrationDelay;
```

In different parts of the code, it's been cast to different uint sizes:

uint256:

```
l1-contracts/src/UniFiAVSManager.sol

170 uint256 deregistrationDelay = $.deregistrationDelay;
```

uint64 (redundant):

```
l1-contracts/src/UniFiAVSManager.sol

195 validator.registeredUntil = uint64(block.number) + uint64(deregistrationDelay);
```

uint128 (the addition operator will result in uint64 (uint + uint64):

```
l1-contracts/src/UniFiAVSManager.sol

262 operator.commitmentValidAfter = uint128(block.number + $.deregistrationDelay);
```

uint256 newValidatorCount

newValidatorCount is defined as uint256 and later casted to uint128, you can define it as uint128 to begin with.

```
l1-contracts/src/UniFiAVSManager.sol

126 uint256 newValidatorCount = blsPubKeyHashes.length;
127 for (uint256 i = 0; i < newValidatorCount; i++) {
```

Similar to the above example:

```
l1-contracts/src/UniFiAVSManager.sol

169 uint256 msgSenderValidatorCount;
170 uint256 deregistrationDelay = $.deregistrationDelay;
```

msgSenderValidatorCount is defined as uint256 and then cast to uint128 in the same function.

l1-contracts/src/UniFiAVSManager.sol

```
200     $.operators[msg.sender].validatorCount -= uint128(msgSenderValidatorCount);
201 }
```

Recommendation

Revisit the variable definitions and their use to clean up redundant castings.

None Registration flow can be improved

Fixed

> This finding has been fixed by introducing `registerOperatorWithCommitment`. Note that this change requires updates to the docs regarding the registration and update commitment flow.

The current Operator/validator registration flow is multi-stepped and can be simplified.

As also mentioned in the docs, the registration flow involved:

1. registerOperator
2. setOperatorCommitment
3. wait for the deregistrationDelay
4. updateOperatorCommitment
5. registerValidators

This is not a security issue. However, an overall goal is to simplify the code flow.

Recommendation

There are different approaches to solving this issue. One approach is to add extra input for the current function (0x0 when commitment is not set in the first call). Another approach can be a second register entry point, such as `registerOperatorWithCommitment`, to prevent the extra operator/validator registration transactions, enhancing the user experience.

1. registerOperatorWithCommitment
2. (updateOperatorCommitment) (this step can be optional in this new code flow)
3. registerValidators

None Redundant check for operator registration

Fixed

> Fixed by removing the explicit duplicate check in https://github.com/PufferFinance/UniFi/pull/25/

When registering operators, `registerOperator` checks if the operator is registered and then calls the AVS directory. `AVS_DIRECTORY.registerOperatorToAVS` also does the same check.

l1-contracts/src/UniFiAVSManager.sol

```
 94 if (
 95     AVS_DIRECTORY.avsOperatorStatus(address(this), msg.sender)
 96         == IAVSDirectory.OperatorAVSRegistrationStatus.REGISTERED
 97 ) {
 98     revert OperatorAlreadyRegistered();
 99 }
100
101 AVS_DIRECTORY.registerOperatorToAVS(msg.sender, operatorSignature);  // @audit-i…
```

`AVS_DIRECTORY.registerOperatorToAVS`:

https://github.com/Layr-Labs/eigenlayer-contracts/blob/decf99caab298592d157a454c225286bd4491093/src/contracts/core/AVSDirectory.sol#L72-L75

Recommendation

The explicit check can be removed.

File Hashes

- ./l1-contracts/src/UniFiAVSManagerStorage.sol
  - fba4480a655ef3b1fc56af0c53eb291344df352a9163839392a344b1205bd736
- ./l1-contracts/src/UniFiAVSManager.sol
  - 5020cdd5e7c2ac5f7af10f870b8bb74a3402ba441b5b1e3b07a6d7cc166591c1

Disclaimer

Creed (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via CD publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report 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. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that CD are not responsible for the content or operation of such Web sites, and that CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. CD assumes no responsibility for the use of third party software on

the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by CD.