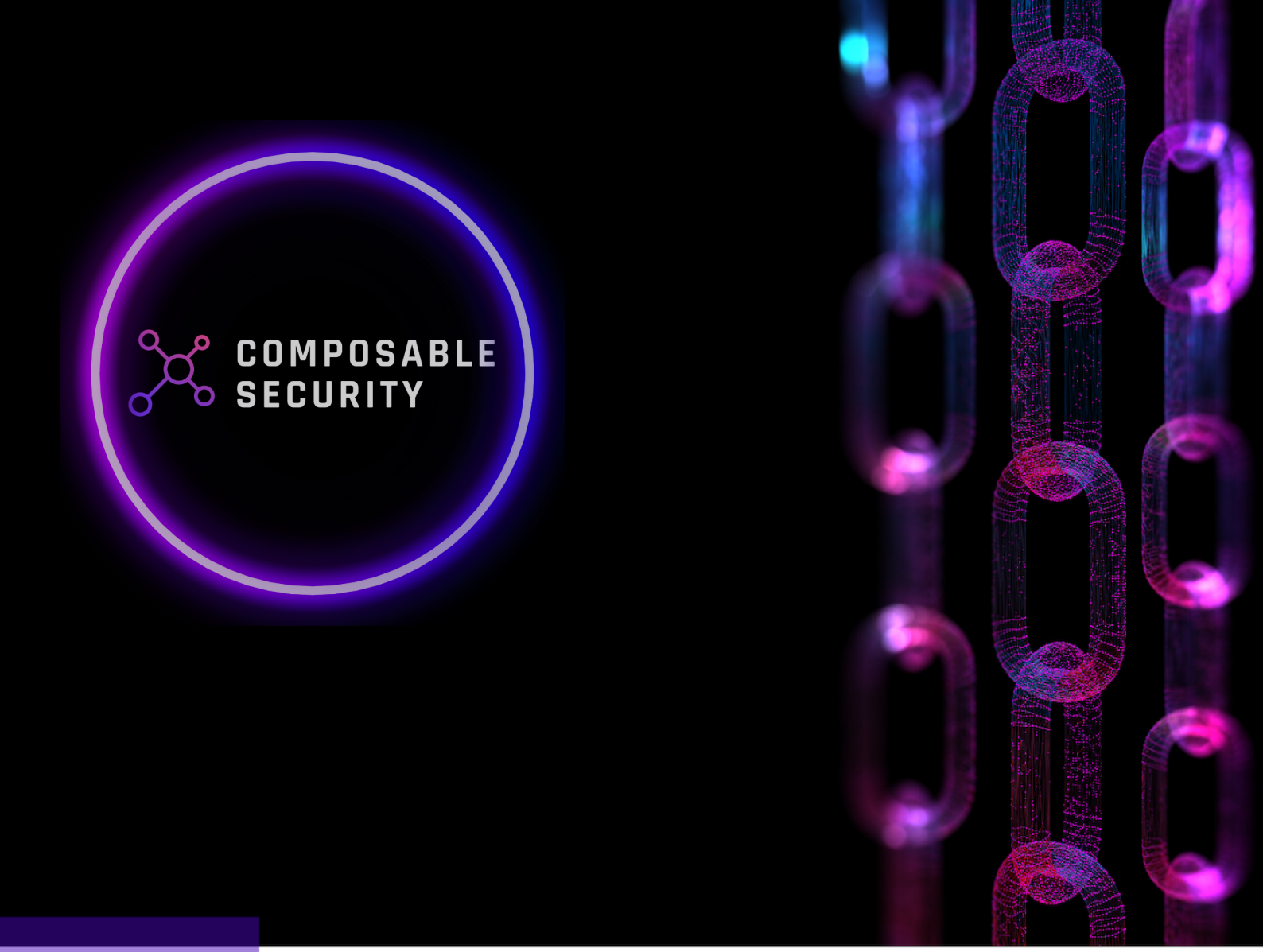


COMPOSABLE SECURITY

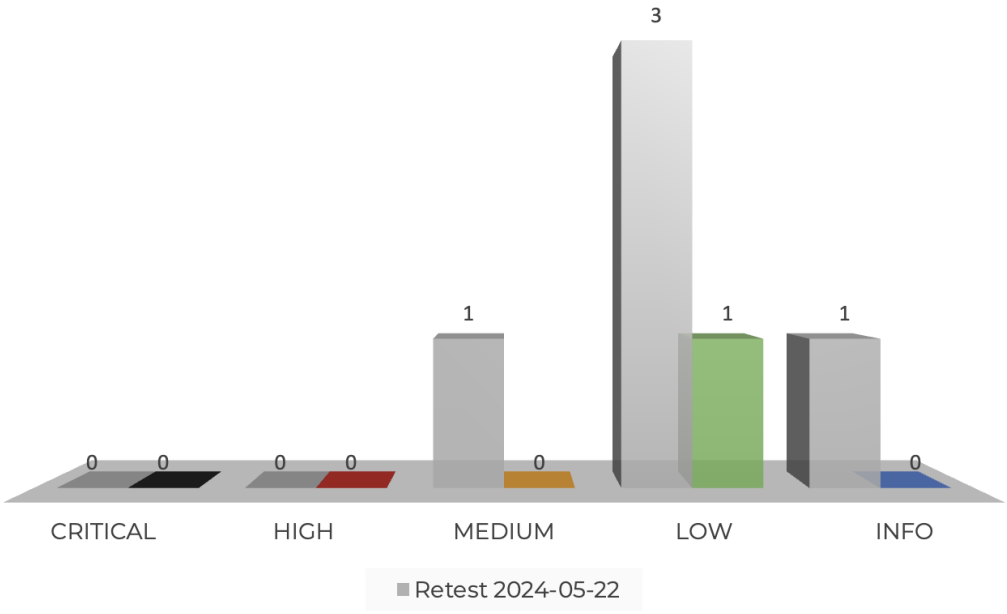
REPORT

Smart contract security review for Filecoin Incentive Design Labs

Prepared by: Composable Security
Report ID: FIDL-b7a8114
Test time period: 2024-05-14 - 2024-05-16
Retest time period: 2024-05-22 - 2024-05-22
Report date: 2024-05-17
Version: 1.0
Visit: composable-security.com

Contents

1. Retest summary (2024-05-22)



The description of the current status for each retested vulnerability and recommendation has been added in its section.

1.1. Results

The **Composable Security** team was involved in a one-time iteration of verification whether the vulnerabilities detected during the tests (between 2024-05-14 and 2024-05-16) were removed correctly and no longer appear in the code.

The current status of detected issues is as follows:

- The vulnerability with a **medium** impact on risk was fully fixed.
- 3 vulnerabilities with a **low** impact on risk were handled as follows:
  - 1 has been acknowledged,
  - 2 have been fixed.
- The security **recommendation** has been implemented.

1.2. Scope

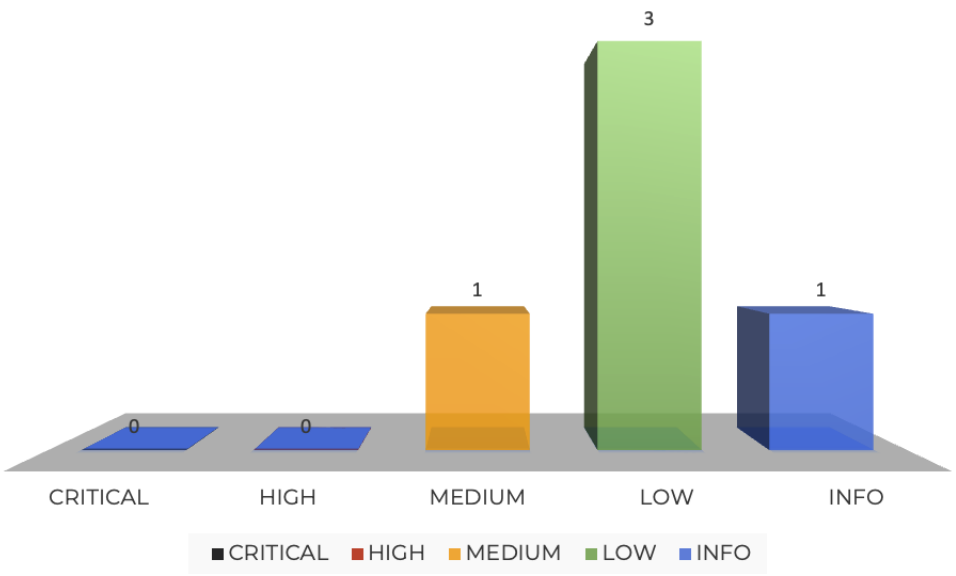
The retest scope included the same contracts, on a different commit in the same repository.

GitHub repository: https://github.com/fidlabs/contract-metaallocator
CommitID: f3af9f185ac702c9bf5b2da6ba1cddb3bd10452e

2. Current findings status

ID	Severity	Vulnerability	Status
FIDL-b7a8114-M01	MEDIUM	Unprotected deployment of allocators	FIXED
FIDL-b7a8114-L01	LOW	Lack of 2 step ownership transfer	FIXED
FIDL-b7a8114-L02	LOW	No allowance cap specified	ACKNOWLEDGED
FIDL-b7a8114-L03	LOW	Front-runnable allowance changes	FIXED
ID	**Severity**	**Recommendation**	**Status**
FIDL-b7a8114-R01	INFO	Consider using specific solidity version	IMPLEMENTED

3. Security review summary (2024-05-16)



3.1. Results

The **Filecoin Incentive Design Labs** engaged Composable Security to review security of **MetaAllocator**. Composable Security conducted this assessment over 2 person-days with 2 engineers.

The summary of findings is as follows:

- No vulnerabilities with **critical** and **high** impact on risk were identified.
- 1 vulnerability with a **medium** impact on risk was identified. It's potential consequence is taking over the meta allocator contract and its DataCap.
- 2 vulnerabilities with a **low** impact on risk were identified.
- 1 **recommendation** has been proposed that can improve overall security and help implement best practice.
- The issues detected concern the access control mechanism and the front-running attack vector.
- The development team was engaged and the communication was very good.

Composable Security recommends that **Filecoin Incentive Design Labs** complete the following:

- Address all reported issues.
- Extend unit tests with scenarios that cover detected vulnerabilities where possible.
- Consider whether the detected vulnerabilities may exist in other ongoing projects that have not been detected during engagement.

3.2. Scope

The scope of the tests included selected contracts from the following repository.

GitHub repository: https://github.com/fidlabs/contract-metaallocator
CommitID: b7a81140128911d7e32f46d911cfa2bf318250b4

The detailed scope of tests can be found in Agreed scope of tests section.

4. Project details

4.1. Projects goal

The Composable Security team focused during this audit on the following:

- Perform a tailored threat analysis.
- Ensure that smart contract code is written according to security best practices.
- Identify security issues and potential threats both for **Filecoin Incentive Design Labs** and their users.

4.2. Agreed scope of tests

The subjects of the test were selected contracts from the **Filecoin Incentive Design Labs** repository.

GitHub repository:
https://github.com/fidlabs/contract-metaallocator
CommitID: b7a81140128911d7e32f46d911cfa2bf318250b4

Files in scope:

```
.
├── ./Allocator.sol
├── ./Factory.sol
└── ./interfaces
    ├── ./interfaces/IAllocator.sol
    └── ./interfaces/IFactory.sol
```

Documentation: The documentation of the project was included within the README.md file.

4.3. Threat analysis

This section summarizes the potential threats that were identified during initial threat modeling performed before the audit. The tests were focused, but not limited to, finding security issues that could be exploited to achieve these threats.

Potential attackers goals:

- Obtaining more tokens by the client than allowed.
- Stealing tokens from other allocators.
- Taking over the meta allocator contract.
- Lock of DataCap due to Denial of Service.

Potential scenarios to achieve the indicated attacker's goals:

- Unauthorized call to functions that set allowances or add verified clients.
- Front-running allowance update to obtain tokens by the client twice.
- Obtaining more tokens by the client via re-entrancy.
- Invalid update of allowance on adding verified client.

4.4. Testing methodology

Smart contract security review was performed using the following methods:

- Q&A sessions with the **Filecoin Incentive Design Labs** development team to thoroughly understand intentions and assumptions of the project.
- Initial threat modeling to identify key areas and focus on covering the most relevant scenarios based on real threats.
- Automatic tests using slither and other tools.
- **Manual review of the code.**

4.5. Disclaimer

Smart contract security review **IS NOT A SECURITY WARRANTY**.

During the tests, the Composable Security team makes every effort to detect any occurring problems and help to address them. However, it is not allowed to treat the report as a security certificate and assume that the project does not contain any vulnerabilities. Securing smart contract platforms is a multi-stage process, starting from threat modeling, through development based on best practices, security reviews and formal verification, ending with constant monitoring and incident response.

Therefore, we encourage the implementation of security mechanisms at all stages of development and maintenance.

5. Vulnerabilities

[FIDL-b7a8114-M01] Unprotected deployment of allocators

MEDIUM FIXED

Retest (2024-05-22)

The team decided that this function cannot be protected with the `onlyOwner` modifier, because it needs to be excuted by the meta allocator's owner.
Instead, the proxy is deployed using `CREATE2` opcode where the salt depends on the `owner` address and its nonce.

Affected files

- Factory.sol#L43

Description

The `Factory` contract allows for the deployment of proxies to the `Allocator` contract, which represents the meta allocator. This is facilitated by the `deploy` function, which can be called by anyone.

In the event that the DataCap is assigned to the new verifier before it is deployed with the legitimate owner, an attacker can take control of the contract and manage the DataCap.

Attack scenario

The attackers might execute the following steps:

① Root Key Holders add a new verifier with the address of the meta allocator to be deployed and assigns some DataCap.
② Root Key Holders submit a transaction to deploy the meta allocator contract using the `deploy` function in the `Factory` contract.
③ The attacker front-runs the transaction and deploys their own meta allocator with a malicious owner address.
④ The attacker then controls the DataCap assigned to the meta allocator.

Result of the attack: The attacker takes over the meta allocator and gains control of the DataCap assigned to it.

Recommendation

Protect the `deploy` function with the `onlyOwner` modifier.

References

1. SCSVS G5: Access control

[FIDL-b7a8114-L01] Lack of 2 step ownership transfer

LOW FIXED

Retest (2024-05-22)

The vulnerability has been removed as recommended.

Affected files

- Factory.sol#L15
- Allocator.sol#L23

Description

The `Factory` contract inherits from `Ownable` contract and the `Allocator` contract inherits from similar, but upgradeable contract - `OwnebleUpgradeable`.

Both of them allow to instantly transfer ownership to any address except `address(0)`. In case of invalid address passed as the new owner (e.g. a contract that cannot make calls to protected functions and the `transferOwnership` function) there is no way to get the ownership back.

Result of the attack: Loss of the ownership for the contract.

Recommendation

- Use `Ownable2Step` contract instead of `Ownable` and `Ownable2StepUpgradeable` instead of `OwnebleUpgradeable`.
- Consider overriding the `renounceOwnership` function to make it always revert if you do not plan to do it

References

1. Ownable2Step.sol
2. SCSVS G5: Access control

[FIDL-b7a8114-L02] No allowance cap specified

LOW ACKNOWLEDGED

Retest (2024-05-22)

The team stated that the cap can change in time, can be assigned later due to the governance process and there is no easy way to read it within the EVM.
The front-end application, used by allocators, will warn them when they want to add a verified client and there is no available cap set yet.

Affected files

- Allocator.sol#L41

Description

The `Allocator` contract does not specify in the `initialize` function the parameter that reflects DataCap it can manage.

This can lead to a situation when the owner of the `Allocator` contract (its proxy) sets allowances to multiple allocators that excess the DataCap and some of allocators would not be able add the filecoin addresses as verified clients.

Vulnerable scenario

The vulnerable scenario can occur in the following way:

① The new meta allocator is deployed and has DataCap of 1000 set.
② The meta allocator's owner sets allowance of 500 to three allocators.
③ The first two allocators add new verified clients with the amount of 500 each.
④ The third allocator tries to add new verified client and the action is blocked because the meta allocator has no DataCap left.

Result of the attack: Inability to add new verified clients even though the allowance is set.

Recommendation

Add the cap of allowances to be set by the owner of meta allocator and set it in the `initialize` function to the value of DataCap assigned to the meta allocator.

References

1. SCSVS G4: Business Logic

[FIDL-b7a8114-L03] Front-runnable allowance changes

LOW FIXED

Retest (2024-05-22)

The vulnerability has been removed. The allowance has to be set to 0 first, if its value is greater than 0.

Affected files

- Allocator.sol#L85

Description

The `setAllowance` function in the `Allocator` contract is used to specify how much tokens an allocator can use to add new verified clients.

This function is vulnerable to a front-running attack in case the meta allocator's owner updates the allowance for an allocator. The allocator can spend both allowances, before and after the update.

Attack scenario

The attackers (malicious allocator) might take the following steps in turn:

① The meta allocator's owner has set the allowance to the allocator for 1000 tokens withing transaction number 1.
② The meta allocator's owner want to update the allowance to 500 and submits transaction number 2 that calls `setAllowance` function with the new amount.
③ The allocator who becomes malicious front-runs the second transaction and adds a verified client with 1000 tokens.
④ The transaction number 2 is executed and the allowance for the allocator, which is now decreased to 0, is set to 500.
⑤ The malicious allocator adds new verified client with 500 tokens.
⑥ In the end the malicious allocant has spent 1500 tokens which the meta allocator's owner wanted to allow them to spend only 500.

Result of the attack: An allocator can spend more tokens than allowed.

Recommendation

There are two possible approaches here:

- Remove the `setAllowance` function, use the `increaseAllowance` function and add `decreaseAllowance` that subtracts the amount from the current allowance value.
- Allow only to set the allowance to 0 from amount, and from 0 to amount within the `setAllowance` function.

References

1. SCSVS G4: Business Logic

6. Recommendations

[FIDL-b7a8114-R01] Consider using specific solidity version

INFO IMPLEMENTED

Retest (2024-05-22)

Implemented as recommended.

Description

In accordance with the best security practices, it is recommended to use the specific version of Solidity compiler so that all tests and compilations are performed with the same version.

The current implementation uses floating pragma:

```
pragma solidity ^0.8.25;
```

Recommendation

Define specific version:

```
pragma solidity 0.8.25;
```

References

1. SCSVS V1: Architecture, design and threat modeling
2. Floating pragma SWC-103

7. Impact on risk classification

Risk classification is based on the one developed by OWASP [1], however it has been adapted to the immutable and transparent code nature of smart contracts. The Web3 ecosystem forgives much less mistakes than in the case of traditional applications, the servers of which can be covered by many layers of security.

Therefore, the classification is more strict and indicates higher priorities for paying attention to security.

OVERALL RISK SEVERITY				
Impact on risk	HIGH	CRITICAL	HIGH	MEDIUM
	MEDIUM	MEDIUM	MEDIUM	LOW
	LOW	LOW	LOW	INFO
		HIGH	MEDIUM	LOW
		Likelihood		

[1] OWASP Risk Rating methodology

8. Long-term best practices

8.1. Use automated tools to scan your code regularly

It's a good idea to incorporate automated tools (e.g. slither) into the code writing process. This will allow basic security issues to be detected and addressed at a very early stage.

8.2. Perform threat modeling

Before implementing or introducing changes to smart contracts, perform threat modeling and think with your team about what can go wrong. Set potential targets of the attacker and possible ways to achieve them, keep it in mind during implementation to prevent bad design decisions.

8.3. Use Smart Contract Security Verification Standard

Use proven standards to maintain a high level of security for your contracts. Treat individual categories as checklists to verify the security of individual components. Expand your unit tests with selected checks from the list to be sure when introducing changes that they did not affect the security of the project.

8.4. Discuss audit reports and learn from them

The best guarantee of security is the constant development of team knowledge. To use the audit as effectively as possible, make sure that everyone in the team understands the mistakes made. Consider whether the detected vulnerabilities may exist in other places, audits always have a limited time and the developers know the code best.

8.5. Monitor your and similar contracts

Use the tools available on the market to monitor key contracts (e.g. the ones where user's tokens are kept). If you have used code from another project, monitor their contracts as well and introduce procedures to capture information about detected vulnerabilities in their code.

Damian Rusinek

Smart Contracts Auditor

@drdr_zz

damian.rusinek@composable-security.com

Paweł Kuryłowicz

Smart Contracts Auditor

@wh01s7

pawel.kurylowicz@composable-security.com