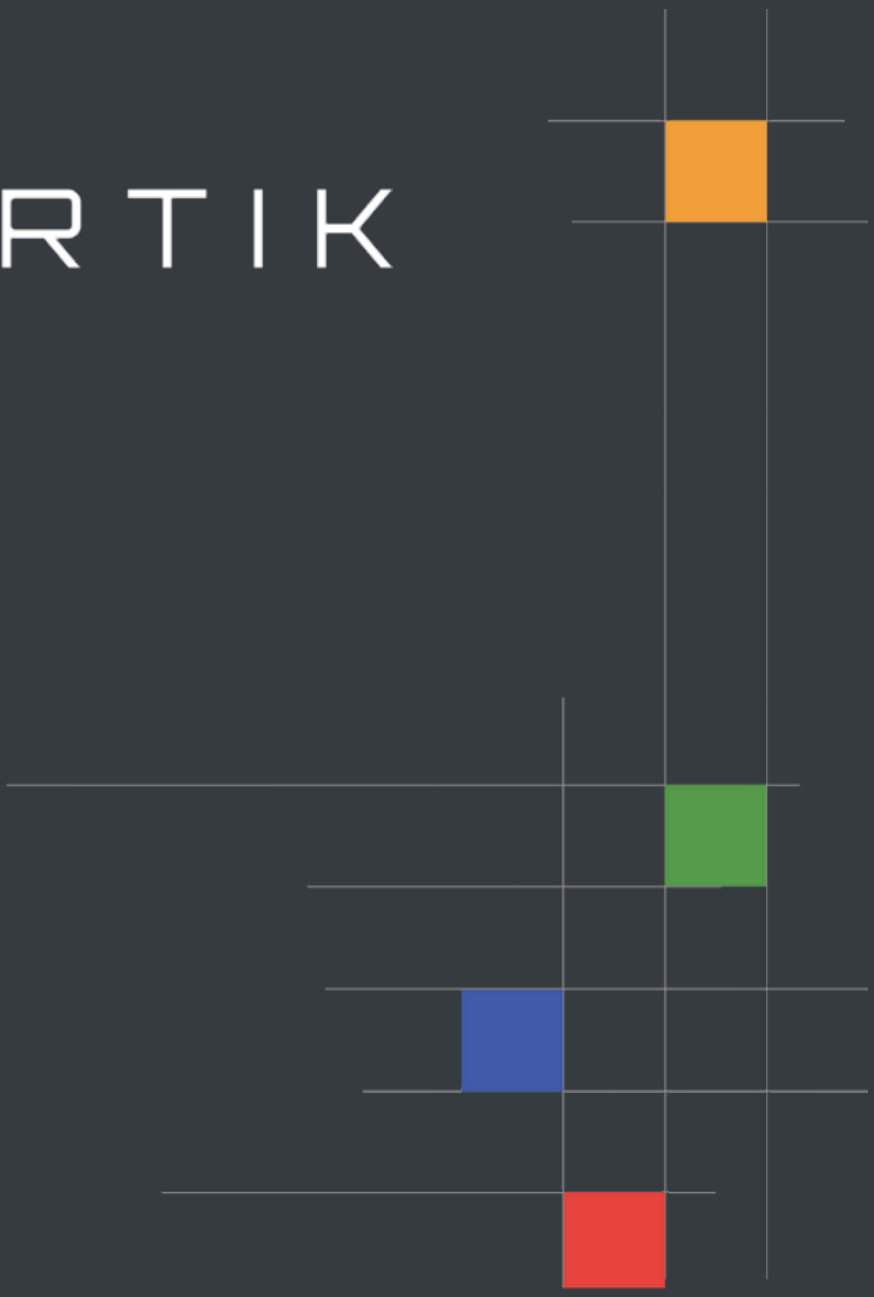
Hedget

Foundation

Security Assessment

May 26th, 2021



Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Hedget-Foundation
Description	Decentralized Application for Options
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. b62086a1cf294a6b6b571ffacbcdda5c7ba41801 2. 7301600eb254384a70f4556279b26670bfd65814 3. 66fa7fa02f87ab9d7f37ebaa044fec875cb16c83

Audit Summary

Delivery Date	May 26th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	April 19th, 2021 - April 28th, 2021

Vulnerability Summary

Total Issues	18
Total Critical	1
Total Major	2
Total Medium	2
Total Minor	2
Total Informational	11

Executive Summary

The report represents the results of our engagement with Hedget on their Hedget DApp version 2. The initial review was conducted for six days: April. 20, 2020 - April. 27 2020 by Adrian Hetman and Alex Papageorgiou.

A critical issue that we have found is the code that leads to vulnerabilities and potential exploitation from the owner side. In the EO_Owner.sol there is `EmergencyStop` function that enables an owner to transfer any tokens from the contract to himself. We strongly advise removing this function as they do not add any needed functionality to the codebase, and in case of lost access to the contract, malicious actors could steal user funds. As an emergency exit for the user's funds, we recommend implementing Withdrawal for users that only the owners of the options could withdraw the funds only when a contract is paused. For more information, please check the finding.

Most functions in EO_Impl.sol doesn't follow checks-effects-pattern, making them susceptible to re-entrancy attacks, especially when any ERC20 compliant token can be added to the Option. It's worth remembering that some implementations of ERC20 and ERC777 tokens like imBTC can inform the recipient of token transfer with callback call thus leading to re-entrancy possibility, which we see in these cases. Such attacks can lead to the loss of funds of the users.

Unfortunately, the project lacks proper documentation explaining the protocol processes, detailing the workflow and intended behavior. We have received a very basic explained which helped in our initial understanding of the code. Still, for the end product and the protocol user, we recommend the team create full documentation similar to the one found for Uniswap. Also, we have discovered leftover code for debugging that could be maliciously used by the owner or by the hacker with stolen owner's keys.

The project is using an upgradability pattern called Diamond Standard which is loosely based on this implementation from the official diamond standard repository. Hedget's version of diamond standard includes diamond storage with faucet support but is missing `diamondCut()` and standard `loupe` functions. It is a basic version of the Diamond to only allow for upgrading the contracts (faucets). With this approach to the upgradability and still developing standard, we must point out some potential issues with it.

1. No contract existence check for the contract's code. When the proxy found in EOpt.sol delegates to an incorrect address or the implementation that has been destructed, the call to the implementation will return success even though no code was executed.
2. Storage pointer risks. Despite the claim that collisions are impossible if the base pointers are

different, a malicious contract can collide with a variable from another implementation.

All the current issues and discussion on the standard can be found here.

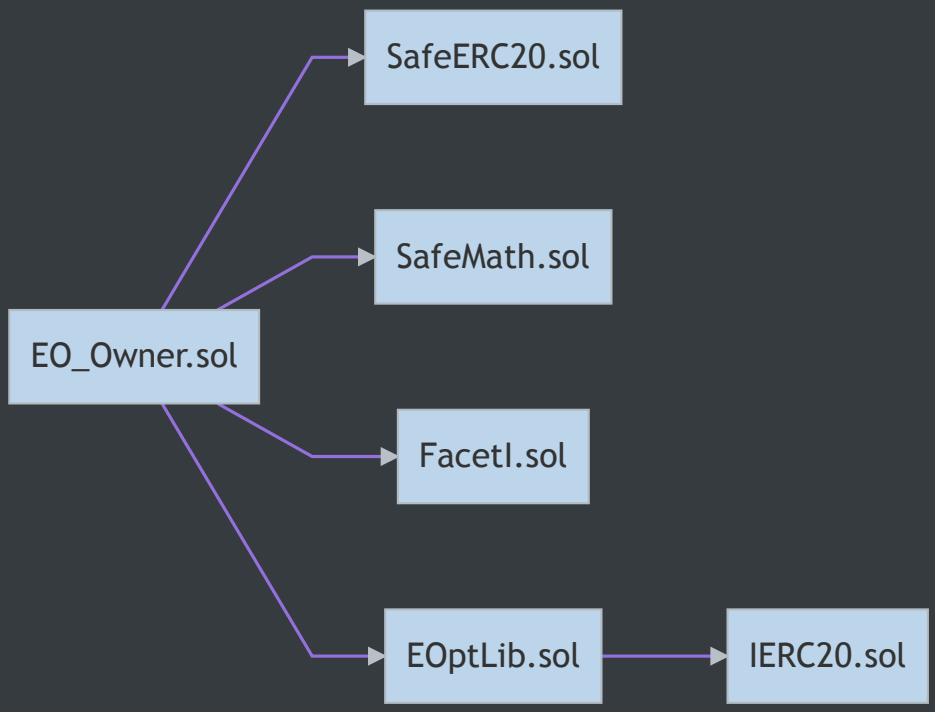
System Analysis

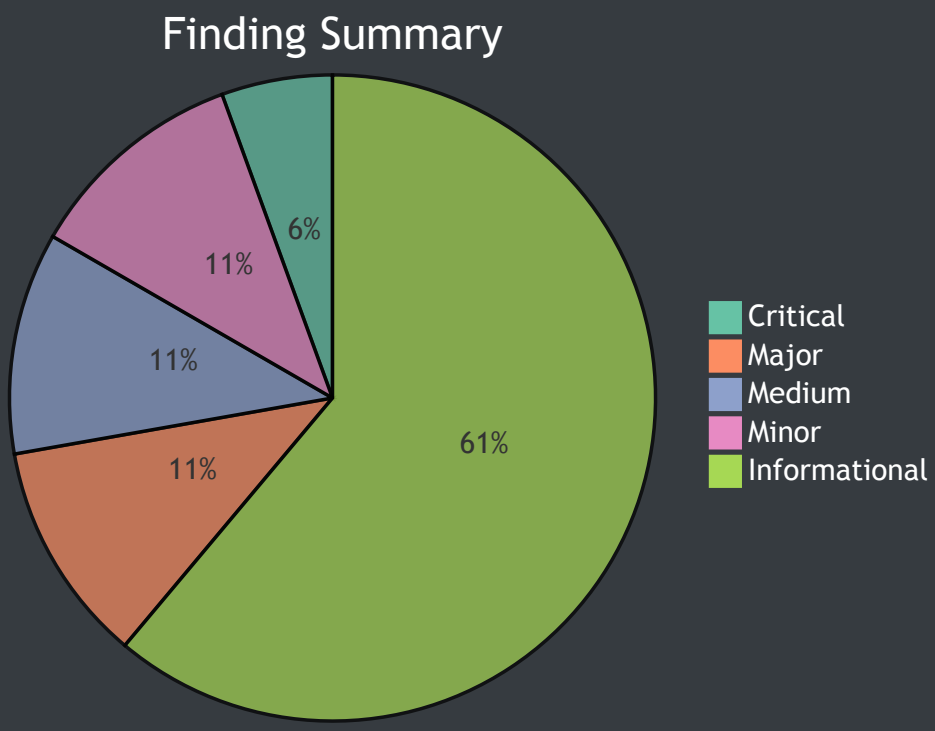
Analysing the contracts we have centralization concerns regarding the project. `onlyOwner` is used for few functions responsible for protocol's parameters and is also used for `EmergencyStop` which we raised our concerns above.
Looking also at the upgradability pattern only the owner could add new implementations. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that and replace key parameters. We advise that a governance system or multi-signature wallet is utilized instead of a single account in this case.

Files In Scope

ID	Contract	Location
EOI	EO_Impl.sol	Hedget2/EO_Impl.sol
EOO	EO_Owner.sol	Hedget2/EO_Owner.sol
EOV	EO_View.sol	Hedget2/EO_View.sol
EOT	EOpt.sol	Hedget2/EOpt.sol
EOL	EOptLib.sol	Hedget2/EOptLib.sol
HED	FacetI.sol	Hedget2/FacetI.sol

File Dependency Graph




Finding Summary
6%
11%
11%
11%
61%
Critical
Major
Medium
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
EOI-01	Possibility of re-entrancy attack	Volatile Code	Major	✓
EOI-02	Lack of input validation	Volatile Code	Minor	✓
EOI-03	Unlocked Compiler Version	Language Specific	Informational	✓
EOI-04	now_time should return block.timestamp	Volatile Code	Informational	✓
EOI-05	Redundant Statements	Dead Code	Informational	✓
EOO-01	Possible drain of all funds by the owner.	Volatile Code	Critical	✓
EOO-02	Owner can set an arbitrary time that is used within the protocol	Volatile Code	Major	✓
EOO-03	Centralization concern	Volatile Code	Medium	✓
EOO-04	Lack of input validation	Volatile Code	Minor	✓
EOO-05	Unlocked Compiler Version	Language Specific	Informational	✓
EOO-06	Redundant Statements	Dead Code	Informational	✓
EOV-01	Unlocked Compiler Version	Language Specific	Informational	✓
EOV-02	Redundant Statements	Dead Code	Informational	✓
EOT-01	Lack of bytecode check before adding a new implementation	Volatile Code	Medium	✓
EOT-02	Unlocked Compiler Version	Language Specific	Informational	✓
EOT-03	Redundant Statements	Dead Code	Informational	✓

EOL-01	Unlocked Compiler Version	Language Specific	Informational	✓
EOL-02	Redundant Statements	Dead Code	Informational	✓

EOI-01: Possibility of re-entrancy attack

Type	Severity	Location
Volatile Code	Major	EO_Impl.sol L326, L356, L387, L414, L469

Description:

Linked functions make token transfer calls before state variables are updated making it susceptible to re-rentrancy attack.
Please also remember that some implementations of ERC20 and ERC777 tokens like imBTC can inform the recipient of token transfer with callback call thus leading to re-entrancy possibility.

Recommendation:

It is recommended to follow checks-effects-interactions pattern for cases like this.
It shields public functions from re-entrancy attacks. It's always a good practie to follow this pattern. `checks-effects-interaction` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.
We would also advise to utilize `nonReantrant` modifier from OpenZeppelin's ReentrancyGuard.sol contract.

Alleviation:

Issue has been resolved. Checks-Effects-Pattern was used alongside `nonReantrant` modifier.

EOI-02: Lack of input validation

Type	Severity	Location
Volatile Code	Minor	EO_Impl.sol General

Description:

Most of the functions that are dealing with addresses, IERC20 or any parameters that should have limits, are missing basic verifiaction of correctness of the passed arguments.

Recommendation:

Add verification of passed arguments e.g. passed address cannot be 0x0 or new limit is between safe limit amount.

Alleviation:

Issue has been resoled.

Client's comment: "In SetoptManager the address of zero means that there is no manager, so the address of zero is valid and serves a function"

EOI-03: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EO_Impl.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue has been resolved. Project is now using 0.6.12 solidity version

EOI-04: now_time should return block.timestamp

Type	Severity	Location
Volatile Code	Informational	EO_Impl.sol L726-L731

Description:

now_time() function should return block.time instead of relying on custom, owner defined variable and its value.

Recommendation:

now_time() should return block.timestamp or remove completely `now_time()` and replace it with block.timestamp.

Alleviation:

Issue has been resolved. not_time() returns now block.timestamp.

EOI-05: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EO_Impl.sol L480-L483, L499, L607, L648, L650

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Issue has been resolved.

EOO-01: Possible drain of all funds by the owner.

Type	Severity	Location
Volatile Code	Critical	EO_Owner.sol L81-L92

Description:

Owner of the contract can call `EmergencyStop` to drain all the tokens in the contract.

Recommendation:

This function needs to be removed as it open a critical vulnerability in the contract. A malicious owner or in a case of loss access to the owner account, a malicious actor could drain all of the funds.
In case of needed to do an emergency withdrawal, we would recommend creating special function just for the user to withdraw their funds.

Alleviation:

Issue has been resolved. Team opted in for removal of the linked function.

EOO-02: Owner can set an arbitrary time that is used within the protocol

Type	Severity	Location
Volatile Code	Major	EO_Owner.sol L95

Description:

Owner have power to change the time in EOLib which is used in EO_Impl.sol for critical parts of the option handling in the system.

Recommendation:

We would recommend to remove this function alltogether and rely on block.timestamp.

Alleviation:

Issue has been resolved. `set_time()` function has been removed.

EOO-03: Centralization concern

Type	Severity	Location
Volatile Code	Medium	EO_Owner.sol L39-L95

Description:

Owner has too much power over most important addresses used in the contract. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that and exploit the protocol.

Recommendation:

Mentioned functions should be called by governance or be handled by multi-sig wallet.

Alleviation:

The Hedget-Foundation development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

EOO-04: Lack of input validation

Type	Severity	Location
Volatile Code	Minor	EO_Owner.sol General

Description:

Most of the functions that are dealing with addresses, IERC20 or any parameters that should have limits, are missing basic verifiaction of correctness of the passed arguments.

Recommendation:

Add verification of passed arguments e.g. passed address cannot be 0x0 or new limit is between safe limit amount.

Alleviation:

Issue has been resolved.

EOO-05: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EO_Owner.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue has been resolved.

EOO-06: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EO_Owner.sol L27

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Issue has been resolved.

EOV-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EO_View.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue has been resolved.

EOV-02: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EO_View.sol L71, L119, L130

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Issue has been resolved.

EOT-01: Lack of bytecode check before adding a new implementation

Type	Severity	Location
Volatile Code	Medium	EOpt.sol L55-L61

Description:

`AddImpl()` is lacking check for contract code that is present in diamond-1 implementation.

Recommendation:

We would recommend to add function `enforceHasContractCode` like in the diamond-1 implementation to prevent adding new implementation address without code in it.

Alleviation:

Issue has been resolved. Bytecode check has been added.

EOT-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EOpt.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue has been resolved.

EOT-03: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EOpt.sol L27-L29, L37, L43-L45

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Issue has been resolved.

EOL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	EOptLib.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue has been resolved.

EOL-02: Redundant Statements

Type	Severity	Location
Dead Code	Informational	EOptLib.sol L90-L91

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Issue has been resolved.

Appendix

Finding Categories

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.