CERTIK

ParaSwap

ParaSwap

Security Assessment

May 6th, 2021

Disclaimer

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	ParaSwap-ParaSwap
Description	ParaSwap aggregates decentralized exchanges and other DeFi services in one comprehensive interface to streamline and facilitate users' interactions with Ethereum's decentralized finance.
Platform	Ethereum; Solidity, Yul
Codebase	GitLab Repository
Commits	1. e5cbed367619eae60d174a5c60770f2bf305a42e

Audit Summary

Delivery Date	May 6th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	March 25th, 2021 - April 7th, 2021

Vulnerability Summary

Total Issues	17
🔴 **Total Critical**	0
🟠 **Total Major**	0
🟡 **Total Medium**	3
🔵 **Total Minor**	3
🟢 **Total Informational**	11

Executive Summary

We were tasked with auditing the ParaSwap contracts for AugustusSwapper. Contract allows for multi-path orders across different exchanges. We also were tasked with checking the UniswapV2.sol implementation and ParaSwap’s Gas Token, ReduxToken.
We haven’t found any critical or major issues with the contracts. Code itself is very well written and optimized for gas savings. Functions are well documented, and more essential contracts, like AugustusSwapper.sol have it’s own md file with documentation.
The team also took its time to rewrite UniswapV2 Router and UniswapV2Lib, to a more optimized version. We haven’t found any issues with those implementations, but few functions are missing some basic checks present in the original implementation.

AugustusSwapper has many publicly available functions that can be re-enter and cause misbehavior of the contract's logic due to events emitted out of order and amounts returned out of order.

Another potential issue is the amount of arbitrary external calls happening within the contract that can cause many misbehavior, i.e. airdrops being claimed by other users on behalf of the contract. We would recommend, in this regard having a whitelist of addresses that can be called externally.

When team got back to us with remediations, the contracts were already deployed onto the mainnet and not many issues were fixed. Contract can be found under this address 0x1bd435f3c054b6e901b7b108a0ab7617c808677b.

System Analysis

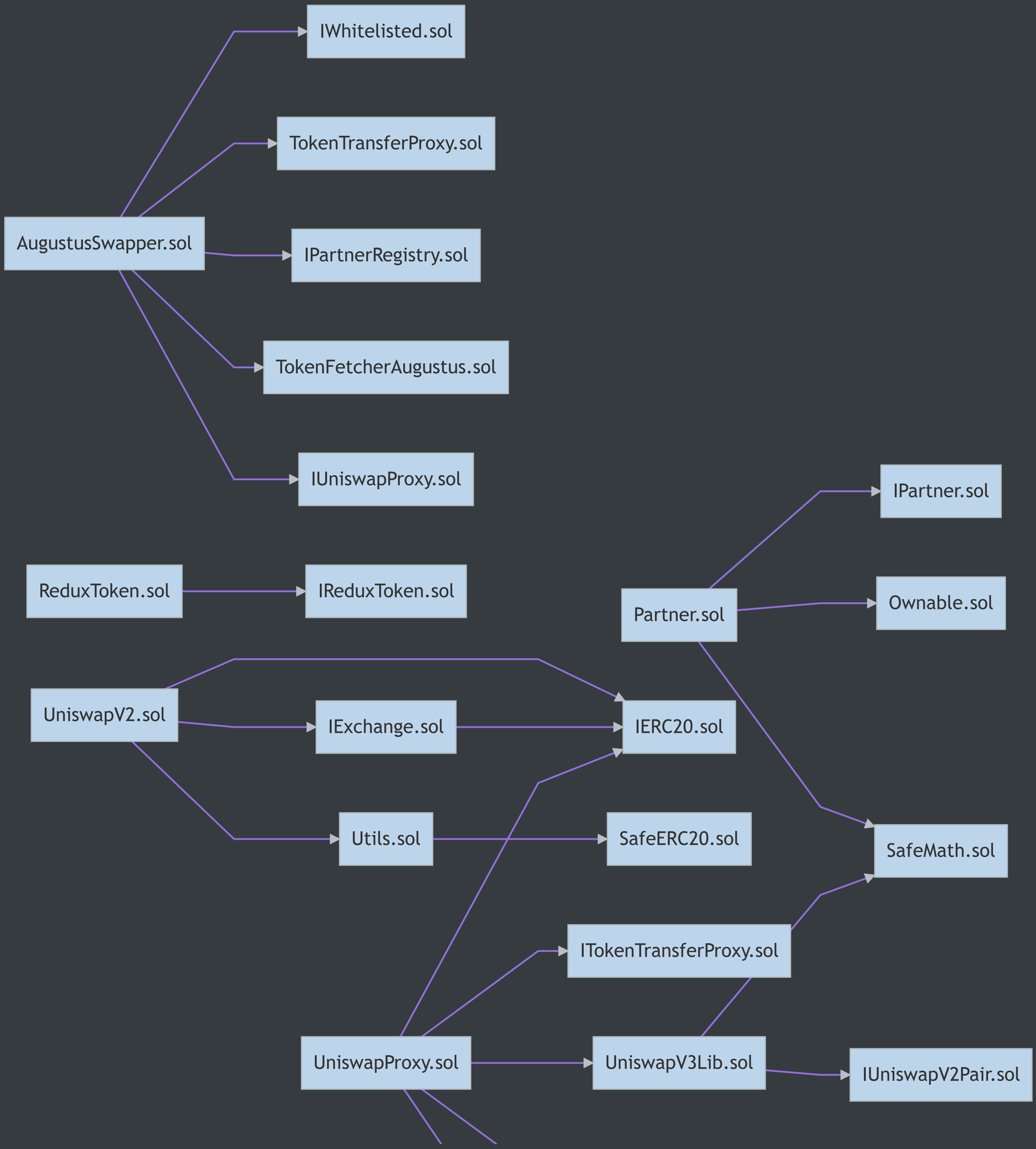
We have found many usages of onlyOwner modifier usage in the AugustusSwapper. Many contract parameters can be changed by the owner at will. The `initializeAdapter` function can be quite dangerous if a malicious actor gets access to the owner's keys.

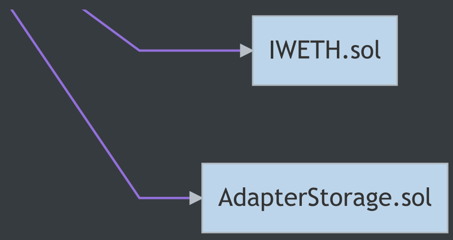
In case of lost access to an account's private key or mishandling security of private keys, an attacker could benefit from that and replace key parameters. We advise that a governance system or multi-signature wallet is utilized instead of a single account in this case.

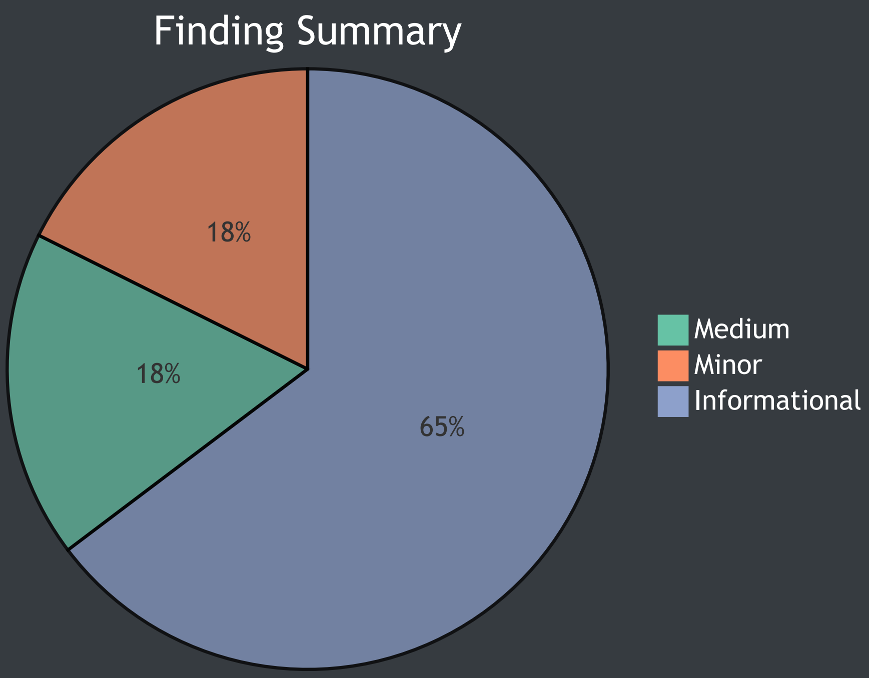
Files In Scope

ID	Contract	Location
ASE	AdapterStorage.sol	original_contracts/AdapterStorage.sol
ASR	AugustusSwapper.sol	original_contracts/AugustusSwapper.sol
IAS	IAugustusSwapper.sol	original_contracts/IAugustusSwapper.sol
IPR	IPartner.sol	original_contracts/IPartner.sol
ORI	IPartnerRegistry.sol	original_contracts/IPartnerRegistry.sol
IRT	IReduxToken.sol	original_contracts/IReduxToken.sol
ITT	ITokenTransferProxy.sol	original_contracts/ITokenTransferProxy.sol
IUP	IUniswapProxy.sol	original_contracts/IUniswapProxy.sol
PAR	Partner.sol	original_contracts/Partner.sol
TTP	TokenTransferProxy.sol	original_contracts/TokenTransferProxy.sol
UPY	UniswapProxy.sol	original_contracts/UniswapProxy.sol
IEE	IExchange.sol	original_contracts/lib/IExchange.sol
RTN	ReduxToken.sol	original_contracts/lib/ReduxToken.sol
TFA	TokenFetcherAugustus.sol	original_contracts/lib/TokenFetcherAugustus.sol
UVL	UniswapV3Lib.sol	original_contracts/lib/UniswapV3Lib.sol
UTI	Utils.sol	original_contracts/lib/Utils.sol
UV2	UniswapV2.sol	original_contracts/lib/uniswapv2/UniswapV2.sol

File Dependency Graph




IWETH.sol
AdapterStorage.sol


Finding Summary
18%
18%
65%
Medium
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
ASR-01	Front-running on \`withdrawAllWETH()\`	Volatile Code	🟡 Medium	ⓘ
ASR-02	Artificially inflating gas refund.	Volatile Code	🟡 Medium	✓
ASR-03	Centralization concern	Control Flow	🟡 Medium	ⓘ
ASR-04	Possibility of Re-entrancy attack	Volatile Code	🔵 Minor	ⓘ
ASR-05	Typo in a function name	Coding Style	🟢 Informational	ⓘ
PAR-01	Immutable variables	Control Flow	🟢 Informational	ⓘ
UPY-01	Packing of local variables	Gas Optimization	🟢 Informational	ⓘ
UPY-02	Unlocked Compiler Version	Language Specific	🟢 Informational	✓
UPY-03	Unnecessary type casting	Gas Optimization	🟢 Informational	ⓘ
RTN-01	Hardcoded address is different then the one in the comments	Inconsistency	🟢 Informational	✓
RTN-02	Pre-compute hash for gas saving.	Gas Optimization	🟢 Informational	✓
UVL-01	Unlocked Compiler Version	Language Specific	🟢 Informational	ⓘ
UTI-01	Should only approve passed amount	Volatile Code	🔵 Minor	ⓘ
UTI-02	Variable tigth-packing	Gas Optimization	🟢 Informational	✓
UV2-01	Lack of token verification	Volatile Code	🔵 Minor	ⓘ
UV2-02	Packing of local variables	Gas Optimization	🟢 Informational	ⓘ
UV2-03	Unnecessary type casting	Gas Optimization	🟢 Informational	ⓘ

ASR-01: Front-running on `withdrawAllWETH()`

Type	Severity	Location
Volatile Code	● Medium	AugustusSwapper.sol L438-L441

Description:

As `withdrawAllWETH` function can be callable by anyone, a front-running attack can occur.

Recommendation:

We would advise to always send remaining amount after a swap/buy/sell to the user thus eliminating need for this function.

Alleviation:

The ParaSwap development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

Client's comment:
"this is not an issue, also `withdrawAllWET` was made onlySelf"

Side effect of making this decision is ETH being locked in the contract without a way of withdrawing it as ETH will be sent to contract itself.

ASR-02: Artificially inflating gas refund.

Type	Severity	Location
Volatile Code	● Medium	AugustusSwapper.sol L419-L429, L499-L509, L586-L596, L647-L662, L678

Description:

Gas refund can be inflated artificially. This can have an impact in case user is making a trade for another beneficiary and that beneficiary is a contract. Malicious beneficiary can run a arbitrary operations ramping up the gas usage and deplete redux token as the user is taking up the gas cost.

Recommendation:

Calculation within the refund of redux tokens uses gasleft() plus the gasleft() calculated at the beginning of the function call. The delta between the two can artificially be inflated by the beneficiary.
We would recommend checking if benefiiciary is one of whitelisted addresses or is not a contract. Another solution would be to ask up front user how much redux tokens he is willing to spend maximally during function execution and use that value if delta between initial gasLeft() and deltaLeft() after all operations is higher then maximum amount user is willing to spent.

Alleviation:

Issue partially resolved. The team added a gas limit of 4000 on ETH transfers.

ASR-03: Centralization concern

Type	Severity	Location
Control Flow	● Medium	AugustusSwapper.sol L88, L119-L137

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 ParaSwap users.

Recommendation:

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

Alleviation:

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

ASR-04: Possibility of Re-entrancy attack

Type	Severity	Location
Volatile Code	● Minor	AugustusSwapper.sol L236, L300, L447, L529, L616

Description:

All linked functions have calls inside them that can send ether to an arbitrary address or make an external call using `externalCall`. This could lead to re-entrancy and cause events being emitted out of order and amount returned out of order as well.
This would cause issues to contracts potentially based on the swap implementation.

Recommendation:

We would recommend using `nonReentrant` modifier from OpenZeppelin.

Alleviation:

The ParaSwap development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

Client's comment:
"we don’t want to use reentrancy flag in storage because of gas, I think this issue stands though"

ASR-05: Typo in a function name

Type	Severity	Location
Coding Style	● Informational	AugustusSwapper.sol L236

Description:

Function `simplBuy` has a typo in it's name.

Recommendation:

Change function name from `simplBuy` to `simpleBuy` and all instances of the `simplBuy` to `simpleBuy`.

Alleviation:

The client won't fix as the contract is already deployed.

PAR-01: Immutable variables

Type	Severity	Location
Control Flow	● Informational	Partner.sol L38, L40

Description:

Linked variables as they are only assigned once during constructor call of the contract should be defined as immutable.

Recommendation:

We would advise to make linked variables immutable.

Alleviation:

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

UPY-01: Packing of local variables

Type	Severity	Location
Gas Optimization	● Informational	UniswapProxy.sol L126, L199, L211

Description:

Linked `for` loops are packing local variables which is inefficient and costs more gas than using uint256.

Recommendation:

We would advise to use uint256 for `i` variable inside for loop.

Alleviation:

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

UPY-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	UniswapProxy.sol L1

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 fixed as of commit 956d4ab0e30e03f0083704d6ede9299aab82b48d on Github repository

UPY-03: Unnecessary type casting

Type	Severity	Location
Gas Optimization	● Informational	UniswapProxy.sol L170, L250

Description:

`uint256(0)` is unneded. Literal 0 can be used directly.

Recommendation:

We would advise to directly use literal 0 for gas saving.

Alleviation:

Issue isn't fixed. Client's comment: "I doubt this makes any difference at all on gas"

RTN-01: Hardcoded address is different than the one in the comments

Type	Severity	Location
Inconsistency	● Informational	ReduxToken.sol L41, L67

Description:

Encoded address inside `mstore` is different than the one used in the pseudocode.

Recommendation:

We recommend changing the comment or the encoded address to be consistent throughtout the example <> code.

Alleviation:

Issue no longer valid. The value on mstore is the same address but XOR'ed with 0x12 (the address of the JUMPDEST).

RTN-02: Pre-compute hash for gas saving.

Type	Severity	Location
Gas Optimization	● Informational	ReduxToken.sol L26

Description:

`PERMIT_TYPEHASH = keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");` can be pre-computed to save gas.

Recommendation:

We would recommend to pre-compute the PERMIT-TYPEHASH and have a comment above explaining what is being hashed.

Alleviation:

Issue no longer valid. Using literal will mean an extra conversion to byte32 which will in fact cost more gas than using keccak256.

UVL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	UniswapV3Lib.sol L1

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 is not fixed as of commit 956d4ab0e30e03f0083704d6ede9299aab82b48d on Github repository. UniswapV3Lib.sol is using `pragma solidity >=0.5.0;` .

UTI-01: Should only approve passed amount

Type	Severity	Location
Volatile Code	● Minor	Utils.sol L103-L118

Description:

`approve` should only approve required amount, not MAX_UINT as in case of exchange mulfunctioning it could lead to potential loss of funds for a user.

Recommendation:

We would advise to only approve required amount sent in function param.

Alleviation:

Issue not fixed.
Client's comment: "we do MAX_UINT approve so that you don’t need to approve in future (saves gas) so this is intentional"

UTI-02: Variable tigth-packing

Type	Severity	Location
Gas Optimization	● Informational	Utils.sol L35-L45, L47-L56, L58-L67, L69-L75, L82-L86, L88-L95

Description:

Variables in linked structs can be tight-packed.

Recommendation:

`bool` variable can be tightpacked with any `address` variable as `address` is 160bytes and `bool` is 8bytes so two of them can be put into the same EVM slot.

Alleviation:

Issue not revelant. Structs defined in Utils.sol are used strictly to memory and variable tight-packing is not applicable in this scenario.

UV2-01: Lack of token verification

Type	Severity	Location
Volatile Code	● Minor	UniswapV2.sol L37, L58

Description:

Functions `swap` and `buy` lack of validation for tokens in a path.

Recommendation:

Functions buy and swap should validate fromToken == data.path[0] and toToken == data.path[data.path.length - 1]. We advise to add such validation to the code.

Alleviation:

Issue not resolved. Client's comment "this is a fair criticism, but in general we can’t validate the payload to our adapters (and it would cost gas)"

UV2-02: Packing of local variables

Type	Severity	Location
Gas Optimization	● Informational	UniswapV2.sol L139

Description:

Linked `for` loops are packing local variables which is inefficient and costs more gas than using uint256.

Recommendation:

We would advise to use uint256 for `i` variable inside for loop.

Alleviation:

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

CERTIK

UV2-03: Unnecessary type casting

Type	Severity	Location
Gas Optimization	● Informational	UniswapV2.sol L187, L250

Description:

`uint256(0)` is unneded. Literal 0 can be used directly.

Recommendation:

We would advise to directly use literal 0 for gas saving.

Alleviation:

Issue not resolved. Client's comment: "I doubt this makes any difference at all on gas"

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

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`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.