mStable Public Report

PROJECT: mstable/mStable-contracts
April 2020

Prepared For:
James Simpson | Stability Labs Pty Ltd
james@stabilitylabs.co

Prepared By:
Jonathan Haas | Bramah Systems, LLC.
jonathan@bramah.systems
# Table of Contents

**Executive Summary**  
Scope of Engagement  
Timeline  
Engagement Goals  
Protocol Specification  
Overall Assessment  
Timeliness of Content  

**General Recommendations**  
Usage of Experimental Solidity Version  
Usage of Block.timestamp  
Integration and Third Party Code Risk  
Highly Privileged Governor Accounts  
Variable Naming  
Outdated NPM Module Usage  
Emitting Events on Success of External Operation  
Inclusion of Unused External Dependency  
ERC20 Race Condition  

**Specific Recommendations**  
Unique to the mStable Protocol  
Deployment Cost Considerations  
Time Passage does not Account for Leap Years and Seconds  
Excess Gas Consumption and Costly Loops in Nexus.sol  
Code Duplication in Module.sol & InitializableModule.sol  
Completion of TODO’s & Incomplete Functionality  
Adherence to Specification  
Concerns regarding De-Pegging  
Concerns regarding Inflation  
StableMath.sol could be simplified  
Missing Return Statements  
Variable Shadowing  

**Toolset Warnings**  
Overview  
Test Coverage  
Static Analysis Coverage
Fuzzing Coverage

Directory Structure

Appendix
  Mythril Detection Capabilities
  Oyente Detection Capabilities
  Slither Detection Capabilities
mStable Protocol Assessment

Executive Summary

Scope of Engagement
Bramah Systems, LLC was engaged in March of 2020 to perform a comprehensive security review of the Stability Labs Pty Ltd (hereafter: “mStable”) protocol smart contract repository. An initial review was conducted over a period of one week by a member of the Bramah Systems, LLC. executive staff. During this period, all Solidity smart contract code (*.sol) as of commit 9c43066ec9cec78234d239a6107d9b3571b6606a was included within scope, along with TypeScript files (*.ts) relevant to testing. TypeScript files were not assessed for their overall security. Bramah Systems completed the assessment using manual, static and dynamic analysis techniques. Following initial completion of the audit, mStable and Bramah Systems mutually expanded the scope of the project to include 4c758e7c4e589cff6d1ef27b02862f1395ef25e5 which featured additional functionality in the ForgeValidator and Masset code segments. This assessment lasted over a period of an additional week.

Timeline
Audit Commencement: April 14, 2020 | May 11th, 2020
Report Delivery: April 17, 2020 | May 15, 2020
Re-Audit Audit Scope Close: May 15, 2020

Engagement Goals
The primary scope of the engagement was to evaluate and establish the overall security of the mStable system, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Does the Solidity code match the specification as provided?
- Is there a way to interfere with the balancing mechanisms?
- Are the arithmetic calculations trustworthy?

Protocol Specification
A substantial specification document was supplied to the Bramah audit team. This document
detailed the interactions between numerous aspects of the code, provided relevant materials containing supporting documentation on aspects of governance and management, and supplied additional information regarding the static analysis performed by the team. The team intends to make certain aspects of this documentation (where not already available) provided to the general public at large.

Overall Assessment

Bramah Systems was engaged to evaluate and identify multiple security concerns in the codebase of the mStable protocol architecture. During the course of our engagement, Bramah Systems noted numerous instances wherein the protocol deviated from established best practices and procedures of secure software development. **With limited exceptions (as described below), these instances were a result of structural limitations of Solidity and not due to inactions on behalf of the development team.**

Overall, the code reviewed is of excellent quality, written with clear awareness of current smart contract development best practices, common security pitfalls, and overall readability. Its interfaces are well designed and its use of patterns display strong code maturity.

In particular, Bramah Systems notes that the code is well commented, particularly in sections where understanding the developer's intent is essential. Additionally, the overall contract organization is consistent throughout (within contracts themselves and their overarching interactions with others).

While during the course of the review Bramah Systems discovered areas worthy of attention by the mStable team, these issues have since been addressed and no significant security concerns remain. We applaud the mStable team for their immense dedication in following security best practices throughout the course of development of their protocol.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.’s knowledge of security patterns as they relate to the mStable Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within “Scope of Engagement” and contained within “Directory Structure”. The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report. The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the mStable
This report is not and should not be relied upon by mStable or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided “as is” without warranty, representation, or guarantee of any kind, including the accuracy of the information provided.

Bramah Systems, LLC. makes no warranties, representations, or guarantees about the mStable Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

**Timeliness of Content**

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.
General Recommendations
Best Practices & Solidity Development Guidelines

Usage of Experimental Solidity Version

A majority of the contracts associated with the protocol make usage of an experimental Solidity version (`pragma experimental ABIEncoderV2`) which enables usage of the new ABI encoder. `ABIEncoderV2` allows for the usage of structs and arbitrarily nested arrays (such as `string[]` and `uint256[][[]]`) in function arguments and return values.

As no present non experimental version for these constructs exists, one must acknowledge the associated risk in utilizing non release-candidate ("RC") software. It is understood that software in the beta phase will generally have more bugs than completed software as well as speed/performance issues and may cause crashes or data loss.

Resolution: Team has accepted the risk, noting no viable alternative.

Usage of Block.timestamp

Miners can affect `block.timestamp` for their benefits. Thus, one should not rely on the exact value of `block.timestamp`. As a result of such, `block.timestamp` and `now` should traditionally only be used within inequalities (note: the protocol does not follow this strategy).

This is particularly important in the Governance and integration areas in which the presumption that `block.timestamp` operates in seconds (per documentation via code comment within `DelayedClaimableGovernor.sol`) presents great risk if ownership exchange of the governor address is particularly time sensitive. While this risk is relatively minimal as a deviance of more than roughly 12 seconds from NTP will not allow an individual to connect to the Ethereum network, a time sensitive change (such as an agreed upon exchange of power at a certain time and date) could prove troublesome.

This noted, no particular test in the testing files provided (specifically, within the `DelayedClaimableGovernor.behaviour.ts` file) by mStable suggests particularly highly time sensitive features, and confirmation with the team ensured the general risk behind block timestamps is known.

Block numbers and average block time can be used to estimate time, but this is not future proof as block times may change (such as the changes expected during Casper). Substantial change to the representation of time unfortunately would lead to deviance from intended ideals, but future solutions are expected to make note of this (due to the sensitive nature of time...
throughout the general corpus of published smart contracts).

**Resolution: Team has accepted the risk, noting no viable alternative.**

### Integration and Third Party Code Risk

Third party integrations weigh a significant risk if untrusted parties are to be involved. While the general security stature of organisations mStable has integrated with (and resultantly, built protocol integrations for) is quite high, this report (and present security analysis) cannot say for certain these integrations will be without flaw. It is notable that all integrations have seen some form of security scrutiny (be it a bug bounty, security audit, or security focused testing via the development team). That said, the scope of this audit does not cover the security of these integrations beyond the protocol integrations themselves.

Notably, substantial testing exists for each integration and verification exists for each step of the integration process (primarily through usage of revert) to mitigate the bulk of these concerns.

**Resolution: Team has accepted the risk.**

### Highly Privileged Governor Accounts

Much of the power of the smart contract is centralized to the governor, an address granted special privileges to make certain modifications to the smart contract operation. Understandably, this poses a fairly unique challenge of ensuring this wallet (regardless of how it is managed) and the associated keys are secured. This centralization of power should be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

As the team notes, ineffective or malicious governance (as a result of these highly permissive accounts) can cause serious concern, including:

- Augmentation of core protocol functionality (namely **BasketManager**)
- Calling **addBasset** with some generic **ERC20** token, setting the basket weight to 100%, then redeeming everything else in the basket
- Pausing the **BasketManager** before implementing a delayed module upgrade and performing the above attack

This noted, the team has included the delayed change of governance (allowing for cancellation) which does mitigate the overall impact of such privileges.

**Resolution: Team has accepted the risk, noting future modifications in the roadmap and the ability for a DAO to operate as Governor.**
Variable Naming

Some of the variable naming could potentially be made more clear. For instance, `basketIsHealthy()` could potentially be renamed `basketIsFailed`, as this is the check that is directly performed (on variable `failed`).

Resolution: Team has renamed relevant variables.

Outdated NPM Module Usage

Throughout the project, NPM modules are utilized in order to import various functionality (notably, OpenZeppelin contracts). While this practice enables relatively minimal modifications to be made in order to invoke certain functions securely (such as with SafeMath), these libraries must be continuously updated in order to ensure they are used securely.

Virtually every non-blockchain application has these issues because most development teams do not focus on ensuring their components/libraries are up to date. In the case of blockchain codebases, however, knowing all outside components utilized is critical.

It is suggested the following steps are followed (as noted by the OWASP project):

1. Identify all components and the versions you are using, including all dependencies. (NPM package lock can help determine these).
2. Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up to date.
3. Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.
4. Where appropriate, consider adding security wrappers around components to disable unused functionality and/or secure weak or vulnerable aspects of the component.

Resolution: Team has updated relevant components where applicable.

Emitting Events on Success of External Operation

Throughout the project, emitted events are used to signify completion or failure of actions internal to the contract (for instance, an event is emitted on successful minting of a token). However, the contracts make numerous external calls which influence core functionality of the protocol which should additionally emit events on their respective successes or failures.

Resolution: The team has added additional emit events to clarify successes and failures.

Inclusion of Unused External Dependency

While the project makes use of numerous external libraries, one such library, MiniMe, is not included despite being referenced within the code. While the usage of this library would
minimize concerns relating to the ERC20 race condition mentioned below, until properly implemented, these protections are not offered. MiniMe does not stop it from the race condition from happening but it at least gives the user a chance to see if the second party withdrew tokens from their account or not, while still maintaining the ERC20 standard.

Resolution: While the governance token will be a MiniMe token, this token is not involved in the scope of this audit.

ERC20 Race Condition
Throughout the project, transfer and transferFrom are utilised heavily. Both of these functions are vulnerable to an attack pattern as follows:

1. Alice allows Bob to transfer $N$ of Alice's tokens ($N > 0$) by calling approve method on Token smart contract passing Bob's address and $N$ as method arguments
2. After some time, Alice decides to change from $N$ to $M$ ($M > 0$) the number of Alice's tokens Bob is allowed to transfer, so she calls approve method again, this time passing Bob's address and $M$ as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls transferFrom method to transfer $N$ Alice's tokens somewhere
4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer $N$ Alice's tokens and will gain an ability to transfer another $M$ tokens
5. Before Alice noticed that something went wrong, Bob calls transferFrom method again, this time to transfer $M$ Alice's tokens.

Alice's attempt to change Bob's allowance from $N$ to $M$ ($N > 0$ and $M > 0$) made it possible for Bob to transfer $N + M$ of Alice's tokens, while Alice never wanted to allow so many of her tokens to be transferred by Bob. The attack described above is possible because the approve method overrides current allowance regardless of whether spender already used it or not, so there is no way to increase or decrease allowance by certain value atomically, unless the token owner is a smart contract, not an account.

While there do appear to be multiple mitigations in place (for instance, using safeApprove from the OpenZeppelin library), no usage of safelIncreaseAllowance or safeDecreaseAllowance exists, both of which are used to prevent successful execution of this vulnerability.

Resolution: The team has introduced mitigations to eliminate concerns regarding the allowance double spend.
Specific Recommendations

Unique to the mStable Protocol

Deployment Cost Considerations

Multiple decisions are made throughout the application that increase the relative deployment cost while bolstering the security of the application. ReentrancyGuard is one such example, with the design specification specifically denoting that design decisions were made to maximize chance of refund which, over the lifetime of the contract, would ideally eclipse the deployment cost.

Resolution: Team has accepted the risk.

Time Passage does not Account for Leap Years and Seconds

Multiple variables are set relying upon the premise of time being roughly equivalent to one day, one week, and so on. However, because not every year equals 365 days and not even every day has 24 hours because of leap seconds, this one day/week/year period is inexact. Due to the fact that leap seconds cannot be predicted, an exact calendar library would require updating by an external oracle.

Note, the direct comparison of these variables within their respective functions poses additional concern, as discussed in “Usage of block.timestamp” above (namely, a proper comparison may not be set). It is worth noting that this has downstream implications on calculations utilising this passage of time (such as interest rates and APY calculations).

Resolution: Team has accepted the risk.

Excess Gas Consumption and Costly Loops in Nexus.sol

If the state variables .balance or .length are used several times, holding its value in a local variable is more gas efficient (as the variable does not need to be accessed every loop iteration).

Moreover, as Ethereum miners impose a limit on the total number of gas consumed in a block, if array.length is large enough, the function will exceed the block gas limit, and transactions calling it will never be confirmed. As a result, if an external entity is to influence array.length, this could pose an issue (such as an individual adding too many Modules). Where possible, avoiding loops with a large number of iterations (or an unknown number of iterations) is advised.
Most notably, the various Module processing code within Nexus.sol falls victim to this attack pattern, although this attack would be incredibly cost prohibitive for the attacker (requiring the addition and subsequent approval of a vast number of modules).

Resolution: Team has accepted the risk.

Code Duplication in Module.sol & InitializableModule.sol

Multiple modifiers are duplicated within the two primary Solidity files concerning module code, Module.sol and InitializableModule.sol. In particular, modifiers pertaining to role-based access control granting certain levels of access to the manager, governor, and ProxyAdmin all exist in duplicated code. While this is not an inherent security issue, this code duplication will increase deployment costs.

Resolution: The usage of the initializableModuleKeys prevents this from being generic. In terms of downsides, the code duplication is mitigated through sharing of test behaviours.

Completion of TODO’s & Incomplete Functionality

Throughout the project, there are multiple instances in which TODO is referenced. In each, establish whether or not the goal of the file has been established (e.g. in contracts/upgradability/DelayedProxyAdmin.sol it appears the contract is feature complete but the TODO exists to denote code that should be removed).

Resolution: Team has completed incomplete functionality.

Adherence to Specification

The smart contracts generally adhere to the provided specification, with some small changes noted, particularly as a result of typographical errors in the code comments. These deviances have been addressed by the team.

Resolution: Team has expanded the specification document to include all code elements.

Concerns regarding De-Pegging

The mStable team noted a unique concern regarding potential de-pegging of bAsset given potential price deviances. In both scenarios posed by the team, the existence of the Auto-Redistribution event should occur, which ideally will handle potential deviances.

However, we do suggest that further exploration be performed into deeper actions that may be able to be taken by governance (especially given the nature of governor accounts in the first iteration of the protocol). For example, removal of offending assets from baskets (those which
despite having the same general peg seem to vary wildly), the ability to freeze exchange of these assets and any assets tied to them (potentially through a global freeze function, but also simply a freeze on the basket itself).

While not inherently a technical control, a vetting process of which assets can be added on the platform would likely assuage most fears of potential depegging, as all relevant stablecoins are understandably designed to be "stable", and frequent or recent instability within the stablecoins history could be indicative of potential problems to come.

**Resolution:** The team has introduced functionality (primarily based within the introduction of fees) that mitigate these concerns.

### Concerns regarding Inflation

The team denotes a particular concern regarding hyperinflation surrounding improper validation during the execution of the `checkBalance` function. In our testing, `checkBalance` performed as anticipated, and we did not encounter issues, even when presenting the function with improper data. This noted, we suggest research into external verification of the price of the `bAsset`, potentially through the use of a third-party verification service (assuaging potential fears related to overly permissive governor accounts).

**Resolution:** The team has introduced functionality (primarily based within the introduction of fees) that mitigate these concerns.

### StableMath.sol could be simplified

StableMath.sol performs a number of mathematical computations needed during successful execution of the smart contracts. However, many of the precise arithmetic and ratio functions have identical code, merely swapping out the secondary variable being impacted (e.g. `mulTruncateCeil` and `mulRatioTruncateCeil` operating essentially the same, only the scale and second variable supplied to the function differ).

**Resolution:** Bramah and mStable agree that the readability benefit to the code outweighs the gas savings.

### Missing Return Statements

`Masset.swap(address,address,uint256,address)` within `masset/Masset.sol#288-333` has missing return statements which can lead to `Masset.swap(address,address,uint256,address).output` within `masset/Masset.sol#295` returning an uninitialized value. Initializing the relevant return variable would remove these concerns.
Variable Shadowing

Variable shadowing occurs when a variable declared within a certain scope (decision block, method, or inner class) has the same name as a variable declared in an outer scope.

Masset.initialize(string,string,address,address,address).symbol (masset/Masset.sol#65) shadows InitializableERC20Detailed.symbol (shared/InitializableToken.sol#12), a state variable.

Masset.initialize(string,string,address,address,address).name (masset/Masset.sol#64) shadows InitializableERC20Detailed.name (shared/InitializableToken.sol#11), a state variable.

InitializableToken.initialize(string,string).symbol (shared/InitializableToken.sol#69) shadows InitializableERC20Detailed._symbol (shared/InitializableToken.sol#12), a state variable.

InitializableToken.initialize(string,string).name (shared/InitializableToken.sol#69) shadows InitializableERC20Detailed._name (shared/InitializableToken.sol#11), a state variable.

InitializableERC20Detailed._initialize(string,string,uint8).decimals (shared/InitializableToken.sol#20) shadows InitializableERC20Detailed.decimals() (shared/InitializableToken.sol#53-55), a function.

InitializableERC20Detailed._initialize(string,string,uint8).symbol (shared/InitializableToken.sol#20) shadows InitializableERC20Detailed.symbol() (shared/InitializableToken.sol#37-39), a function.

InitializableERC20Detailed._initialize(string,string,uint8).name (shared/InitializableToken.sol#20) shadows InitializableERC20Detailed.name() (shared/InitializableToken.sol#29-31), a function.

Resolution: Relevant variables where applicable have been renamed.
Toolset Warnings
Unique to the mStable Protocol

Overview
In addition to our manual review, our process involves utilizing concolic analysis and dynamic testing in order to perform additional verification of the presence security vulnerabilities. An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist. The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable, in addition to findings generated through manual inspection.

Test Coverage
The contract repository heavily benefits from substantial unit test coverage throughout. This testing provides a variety of unit tests which encompass the various operational stages of the contract. The mStable protocol (and its relevant components and their respective subcomponents) possesses numerous tests validating functionality and ensuring that certain behaviors (those relating to erroneous or overflow-prone input) do not see successful execution.

In particular, specific focus within the testing suite was placed upon validating that various actions (especially with respect to governance and basket management) cannot occur after a state change or as the result of bad input (such as an invalid address).

The mStable team constructed tests in both TypeScript and native Solidity, allowing for a fairly robust test-suite.

Static Analysis Coverage
The contract repository underwent heavy scrutiny with multiple static analysis agents, including:

- Securify
- MAIAN
- Mythril
- Oyente
- Slither

In each case, the team had mitigated relevant concerns raised by each of these tools. In particular, many tools pointed to potential areas of reentrancy, in which multiple state variables
are written following external calls. For each of these individual calls, Bramah confirmed the existence of a mitigating factor (namely, the usage of `ReentrancyGuard`). In areas in which `ReentrancyGuard` is not used, such as within `DelayedProxyAdmin`, specific efforts by the development team are made to avoid potential for reentrancy (seen within lines 96-97).

Fuzzing Coverage

The contract repository underwent heavy scrutiny with fuzzing utility `Echidna`, running custom rulesets as well as those within the `Crytic.io` platform.

Rules included checking for limitations around the ability to swap (such as when BasketManager is paused, bAssets are undergoing recollateralisation, or if the basket has failed).
Directory Structure

At time of initial review, the directory structure of the mStable contract (/contracts) repository was as follows:

```
- Migrations.sol
- governance
  - ClaimableGovernor.sol
  - DelayedClaimableGovernor.sol
  - Governable.sol
  - InitializableGovernableWhitelist.sol
- interfaces
  - IBasketManager.sol
  - IMasset.sol
  - INexus.sol
  - IPlatformIntegration.sol
  - ISavingsContract.sol
  - ISavingsManager.sol
- masset
  - BasketManager.sol
  - Masset.sol
  - MassetToken.sol
- forge-validator
  - ForgeValidator.sol
  - IForgeValidator.sol
- mUSD.sol
- platform-integrations
  - AaveIntegration.sol
  - CompoundIntegration.sol
  - IAave.sol
  - ICompound.sol
  - InitializableAbstractIntegration.sol
```
Following the request for expansion of audit scope, the directory structure of the mStable contract (.contracts) repository was as follows:

- Migrations.sol
- governance
  - ClaimableGovernor.sol
  - DelayedClaimableGovernor.sol
  - Governable.sol
  - InitializableGovernableWhitelist.sol
- interfaces
mStable Protocol

├── MockModule.sol
├── MockPausableModule.sol

├── MockProxy.sol
├── PublicStableMath.sol
└── upgradability
    └── MockImplementation.sol

18 directories, 57 files
## Appendix

### Mythril Detection Capabilities

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
<th>Mythril Detection Module(s)</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unprotected functions</td>
<td>Critical functions such as sends with non-zero value or suicide() calls are callable by anyone, or msg.sender is compared against an address in storage that can be written to. E.g. Parity wallet bugs.</td>
<td><code>Unchecked_suicide, Ether_send unchecked_retrval</code></td>
<td></td>
</tr>
<tr>
<td>Missing check on CALL return value</td>
<td></td>
<td><code>unchecked_retrval</code></td>
<td>Handle errors in external calls</td>
</tr>
<tr>
<td>Re-entrancy</td>
<td>Contract state should never be relied on if untrusted contracts are called. State changes after external calls should be avoided.</td>
<td><code>external calls to untrusted contracts</code></td>
<td>Call external functions last Avoid state changes after external calls</td>
</tr>
<tr>
<td>Multiple sends in a single transaction</td>
<td>External calls can fail accidentally or deliberately. Avoid combining multiple send() calls in a single transaction.</td>
<td></td>
<td>Favor pull over push for external calls</td>
</tr>
<tr>
<td>Issue</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>External call to untrusted contract</td>
<td><strong>external calls to untrusted contracts</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delegatecall or callcode to untrusted contract</td>
<td><strong>delegatecall_forward</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer overflow/underflow</td>
<td><strong>integer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp dependence</td>
<td><strong>Dependence on predictable variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payable transaction does not revert in case of failure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of tx.origin</td>
<td><strong>tx_origin</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type confusion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predictable RNG</td>
<td><strong>Dependence on predictable variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction order dependence</td>
<td><strong>Transaction order dependence</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information exposure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complex fallback function (uses more than 2,300 gas)</td>
<td>A too complex fallback function will cause send() and transfer() from other contracts to fail. To implement this we first need to fully implement gas simulation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use require()instead of assert()</td>
<td>Use assert() only to check against states which should be completely unreachable.</td>
<td>Exceptions</td>
<td>Solidity docs</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Use of deprecated functions</td>
<td>Use revert() instead of throw(), selfdestruct() instead of suicide(), keccak256() instead of sha3()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detect tautologies</td>
<td>Detect comparisons that always evaluate to 'true', see also #54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call depth attack</td>
<td>Deprecated</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Oyente Detection Capabilities

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-entrancy</td>
<td>Contract state should never be relied on if untrusted contracts are called. State changes after external calls should be avoided.</td>
</tr>
<tr>
<td>Timestamp Dependence</td>
<td>The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. Block numbers and average block time can be used to estimate time, but this is not future proof as block times may change (such as the changes expected during Casper).</td>
</tr>
<tr>
<td>Assertion Failure</td>
<td>An assertion is a boolean expression at a specific point in a program which will be true unless there is a bug in the program.</td>
</tr>
</tbody>
</table>
Assertion failures as such denote critical instances in which assumptions made by the developer no longer hold to be true.

<table>
<thead>
<tr>
<th>Callstack Depth Attack</th>
<th>Deprecated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Order Dependence (TOD)</td>
<td>Since a transaction is in the mempool for a short while, one can know what actions will occur, before it is included in a block. This can be troublesome for things like decentralized markets, where a transaction to buy some tokens can be seen, and a market order implemented before the other transaction gets included.</td>
</tr>
<tr>
<td>Parity Multisig Bug 2</td>
<td>Unchecked kill/selfdestruct functions, such as those within the Parity Multisig Bug 2 can lead to destruction of the contract, sending funds to the given address provided.</td>
</tr>
</tbody>
</table>

### Slither Detection Capabilities

<table>
<thead>
<tr>
<th>Detector</th>
<th>What it detects</th>
<th>Impact</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>name-reused</td>
<td>Contract's name reused</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>rtlo</td>
<td>Right-To-Left-Override control character is used</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>shadowing-state</td>
<td>State variables shadowing</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>suicidal</td>
<td>Functions allowing anyone to destruct the contract</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>uninitialized-state</td>
<td>Uninitialized state variables</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>uninitialized-storage</td>
<td>Uninitialized storage</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Severity</td>
<td>Risk</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>arbitrary-send</td>
<td>Functions that send ether to arbitrary destinations</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>controlled-delegatecall</td>
<td>Controlled delegatecall destination</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>reentrancy-eth</td>
<td>Reentrancy vulnerabilities (theft of ethers)</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>erc20-interface</td>
<td>Incorrect ERC20 interfaces</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>erc721-interface</td>
<td>Incorrect ERC721 interfaces</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>incorrect-equality</td>
<td>Dangerous strict equalities</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>locked-ether</td>
<td>Contracts that lock ether</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>shadowing-abstract</td>
<td>State variables shadowing from abstract contracts</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>tautology</td>
<td>Tautology or contradiction</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>boolean-cst</td>
<td>Misuse of Boolean constant</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>constant-function-asm</td>
<td>Constant functions using assembly code</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>constant-function-state</td>
<td>Constant functions changing the state</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>divide-before-multiply</td>
<td>Imprecise arithmetic operations order</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>reentrancy-no-eth</td>
<td>Reentrancy vulnerabilities (no theft of ethers)</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>tx-origin</td>
<td>Dangerous usage of tx.origin</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Category</td>
<td>Issue Description</td>
<td>Severity</td>
<td>Recommendation</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------------</td>
</tr>
<tr>
<td>unchecked-lowlevel calls</td>
<td>Unchecked low-level calls</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>unchecked-send</td>
<td>Unchecked send</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>uninitialized-local</td>
<td>Uninitialized local variables</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>unused-return</td>
<td>Unused return values</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>shadowing-builtin</td>
<td>Built-in symbol shadowing</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>shadowing-local</td>
<td>Local variables shadowing</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>void-cst</td>
<td>Constructor called not implemented</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>calls-loop</td>
<td>Multiple calls in a loop</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>reentrancy-benign</td>
<td>Benign reentrancy vulnerabilities</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>reentrancy-events</td>
<td>Reentrancy vulnerabilities leading to out-of-order Events</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>timestamp</td>
<td>Dangerous usage of block.timestamp</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>assembly</td>
<td>Assembly usage</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>boolean-equal</td>
<td>Comparison to boolean constant</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>deprecated-standards</td>
<td>Deprecated Solidity Standards</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>erc20-indexed</td>
<td>Un-indexed ERC20 event parameters</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>low-level-calls</td>
<td>Low level calls</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>naming-convention</td>
<td>Conformance to Solidity naming conventions</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>pragma</td>
<td>If different pragma directives are used</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>solc-version</td>
<td>Incorrect Solidity version</td>
<td></td>
<td>Informational</td>
</tr>
<tr>
<td>unused-state</td>
<td>Unused state variables</td>
<td></td>
<td>Informational</td>
</tr>
</tbody>
</table>

28
<table>
<thead>
<tr>
<th>Category</th>
<th>Issue Description</th>
<th>Severity</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>reentrancy-unlimited-gas</td>
<td>Reentrancy vulnerabilities through send and transfer</td>
<td>Informational</td>
<td>Medium</td>
</tr>
<tr>
<td>too-many-digits</td>
<td>Conformance to numeric notation best practices</td>
<td>Informational</td>
<td>Medium</td>
</tr>
<tr>
<td>constable-states</td>
<td>State variables that could be declared constant</td>
<td>Optimization</td>
<td>High</td>
</tr>
<tr>
<td>external-function</td>
<td>Public function that could be declared as external</td>
<td>Optimization</td>
<td>High</td>
</tr>
</tbody>
</table>