Three Sigma Labs

Code Audit

MAPLE

Maple V2 Token

Disclaimer

Code Audit

Maple V2 Token

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgement of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Maple V2 Token

Table of Contents

Disclaimer 3
Summary 8
Scope 10
Methodology 12
Project Dashboard 14
Code Maturity Evaluation 17
Findings 20
3S-MPLV2T-L01 20
3S-MPLV2T-L02 21
3S-MPLV2T-L03 22
3S-MPLV2T-L04 24
3S-MPLV2T-L05 26
3S-MPLV2T-N01 27
3S-MPLV2T-N02 28
3S-MPLV2T-N03 29
3S-MPLV2T-N04 30
3S-MPLV2T-N05 31
3S-MPLV2T-N06 32
3S-MPLV2T-N07 33
3S-MPLV2T-N08 34
3S-MPLV2T-N09 35
3S-MPLV2T-N10 36
3S-MPLV2T-N11 38
3S-MPLV2T-N12 40
3S-MPLV2T-N13 41
3S-MPLV2T-N14 42
3S-MPLV2T-N15 43
3S-MPLV2T-N16 44

Summary

Code Audit

Maple V2 Token

Summary

Three Sigma Labs audited Maple V2 Token in a 2 person week engagement. The audit was conducted from 24-07-2023 to 28-07-2023.

Protocol Description

Maple Finance is an institutional crypto-capital network built on Ethereum. Maple provides the infrastructure for credit experts to efficiently manage and scale crypto lending businesses and connect capital from institutional and individual lenders to innovative, blue-chip companies.

The Maple V2 token, used for Maple governance, is based on the audited and battle-tested ERC20 contract. The existing ERC20 contract used for xMPL and all existing pools will serve as the foundation for Maple V2. The token uses modules that are smart contracts with predefined rules that can mint or burn Maple tokens. Authorised modules have a significant impact on the token and the protocol and aim for simplicity to avoid unintended consequences.

Scope

Code Audit

Maple V2 Token

Scope

```
contracts/
├── EmergencyModule.sol
├── RecapitalizationModule.sol
├── MapleTokenInitializer.sol
├── MapleTokenProxy.sol
├── MapleToken.sol
└── interfaces
        ├── IEmergencyModule.sol
        ├── IRecapitalizationModule.sol
        ├── IMapleTokenInitializer.sol
        ├── IMapleTokenProxy.sol
        ├── IMapleToken.sol
        └── Interfaces.sol
```

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. The libraries used in the implementation of these contracts are internal contracts that have been previously audited and shown to be secure.

Methodology

Code Audit

Maple V2 Token

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarises the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Maple V2 Token

Project Dashboard

Application Summary

Name	Maple V2 Token
Commit	dc9dd28
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	24-07-2023 to 28-07-2023
№ of Auditors	2
Review Time	2 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	5	1	4
None	16	7	9

Category Breakdown

Suggestion	6

Documentation	0
Bug	4
Optimization	5
Good Code Practices	7

Code Maturity Evaluation

Code Audit

Maple V2 Token

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. The governor has significant privileges over the protocol.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Moderate. The smart contract implementations can be modified after deployment, albeit with proper timelocks and functional upgradeability patterns.
Function Composition	Moderate. Some constants and pieces of code in several files are similar and could be reused.
Front-Running	Satisfactory. While it's possible to front-run a migration in this code it is very unlikely.
Monitoring	Satisfactory. Events are correctly emitted.
Specification	Satisfactory. In-depth and well structured high-level specification as well as codebase documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Findings

Code Audit

Maple V2 Token

Findings

3S-MPLV2T-L01

The **`NonTransparentProxy`** is vulnerable to selector clashing

Id	3S-MPLV2T-L01
Classification	Low
Severity	Low
Likelihood	
Category	Bug

Description

The **`NonTrasparentProxy`** is vulnerable to a selector clashing vulnerability. Essentially, if the selector of the implementation matches one of the selectors of the proxy, the called function will always be the one in the proxy. This is dangerous because the governor might want to call a function in the implementation and end up calling a permissioned function in the proxy itself. The likelihood of this happening is very low.

Recommendation

There are several options to address this issue, namely

1. Check manually on every new implementation that there is no selector clash with the proxy functions (only **`setImplementation()`**).

2. Use Openzeppelin's approach. Deploy a separate proxy admin which is only able to call the functions of the proxy and the remaining users can only call the functions of the implementation.

3. Use a UUPS proxy.

Status

The team is aware of this condition and expects to find such issues in their testing suite, therefore no action is necessary.

3S-MPLV2T-L02

The **migrate()** function in the **Migrator** can be DoSed

Id	3S-MPLV2T-L02
Classification	Low
Severity	Low
Likelihood	
Category	Suggestion

Description

Anyone can DoS a migration that approves and calls **migrate()** non atomically by frontrunning the **migrate()** call, after the **approve()**, with a **migrate(address owner_, uint256 amount_)** call with an amount of 1. This would make the **transferFrom()** in the original **migrate()** call revert due to insufficient approval.

Note: it might be best not to fix this issue because this migrator code has been deployed in the past and smart contracts other than the xMPL might not be compatible with the function that only has the amount argument.

Recommendation

The correct fix would be modifying the visibility of the **migrate(address owner_, uint256 amount_)** function to internal/private. This way, only the other **migrate(uint256 amount_)** function is available, which is not vulnerable. The `xMPL` is compatible with the **migrate(uint256 amount_)** function.

But, as mentioned before, it might break compatibility with other smart contracts and it's probably best to let it as is.

Status

As stated in the description there can be compatibility issues and also the likelihood of this kind of grieving is very low, therefore the team decided to leave the code as is.

3S-MPLV2T-L03

`RecapitalizationModule`, `_claimable()` overflows if `issuanceRate*(windowEnd_ - from_)` is bigger than `type(uint208).max`

Id	3S-MPLV2T-L03
Classification	Low
Severity	Low
Likelihood	
Category	Bug

Description

In the `RecapitalizationModule`, `_claimable()`, the `claimableAmount_` is calculated as

```
claimableAmount_ += window_.issuanceRate * (windowEnd_ - from_);
```

`window_.issuanceRate` is type `uint208`, so it overflows if the result of the multiplication exceeds `type(uint208).max`, which could cause unexpected results if the `claimableAmount_` was expected to reach `uint256` in a given timeframe.

See the `test_fuzz_ThreeSigma_claimable()` for details.

Recommendation

This should not be a real issue because the `issuanceRate` is supposed to be a much more reasonable amount (the wiki mentions around `2^20` each year, which will not overflow).

The `window.issuanceRate` can be casted to `uint256` to have additional room for `claimableAmount_`, but even this could end up in too much `claimableAmount_` and revert when minting because it is bigger than `totalSupply`.

To be fully secure, at every iteration `claimableAmount_` should be bound to `type(uint256).max` and at the end of the function call checked against the current supply and bound to what's left to inflate.

This would increase gas costs and is probably not worth it unless Maple plans to reach a supply of **`type(uint256).max`**.

Status

As this is an unrealistic scenario with huge numbers the team decided not to implement a fix saying that if the situation ever arises they'll take the route of disabling the module and adding a new one to properly handle the claimable amount.

3S-MPLV2T-L04

When scheduling more than one window so that it replaces existing windows, events emitted become incorrect

Id	3S-MPLV2T-L04
Classification	Low
Severity	Low
Likelihood	
Category	Bug

Description

In **`RecapitalizationModule`** when scheduling at least two new windows that are going to completely replace at least one existing window the event `` `WindowScheduled()` `` will emit the incorrect **`previousWindowId`** starting from the second new window.

For example, imagine we have the following schedule:

```
|--------|----------|--------------->
 ---W1-------W2--------W3
```

and want to replace it with:

```
|--------|-------|----------|-------->
 ---W1----W2-----W4-----W5
```

The event for W4 will be **`WindowScheduled(2,4,...,...)`** which is correct.

The event for W5 will be **`WindowScheduled(3,5,...,...)`** which is incorrect it should be **`WindowScheduled(4,5,...,...)`**.

This happens because the **`previousWindowId`** is calculated using `` `insertionWindowId_` `` which only makes sense for the first new window.

This issue was not found during testing because the foundry command **`vm.expectEmit()`** checks if the events are being emitted correctly, however it only checks the next event and not all that follow.

Example:

```
vm.expectEmit();
emit WindowScheduled(1, 3, start + 50 days,  0.96e18);
emit WindowScheduled(3, 4, start + 120 days, 0.99e18);
```

This code only checks for the first event and not the second. To fix this you need to place an **`vm.expectEmit()`** before each **`emit`**.

Recommendation

Replace **`insertionWindowId_ + index_`** in the event with

`index_ == 0 ? insertionWindowId_ : newWindowId_ + index_ - 1`.

Status

Issue addressed here: maple-labs/mplv2#52

3S-MPLV2T-L05

Missing contract checks in several locations

Id	3S-MPLV2T-L05
Classification	Low
Severity	Low
Likelihood	
Category	Good Code Practices

Description

It's a good practice to check if there is code in an address when setting it, as it can prevent mistakes. The compiler automatically creates these checks since 0.8.10 when a function of that address is called. However, there are instances where the addresses are set without explicitly calling any function.

Recommendation

The following addresses are missing this check:

1. `setImplementation()` of the **`MapleTokenProxy`**.
2. **`implementation_`**, **`initializer_`** and **`tokenMigrator`** in the `constructor()` of **`MapleTokenProxy`**.
3. `addModule()` and `removeModule()` in **`MapleToken`**.
4. `globals` and `token` in the **`EmergencyModule`**.
5. In **`RecapitalizationModule`**, **`token`** in the `constructor()`.
6. In **`RecapitalizationModule`**, `.mapleTreasury()` an **`address(0)`** check.
7. In **`MapleTokenInitializer`**, the `treasury_` (at least an **`address(0)`** check).

Status

The team is not concerned with address checks and would rather keep the implementation as is.

3S-MPLV2T-N01

Memory variables not following underscore convention

Id	3S-MPLV2T-N01
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

In the **`MapleToken`** contract, functions `addModule()` and `removeModule()` have the parameter **`address module`** which doesn't follow maple's convention of adding an underscore at the end of local variable's names.

Recommendation

For consistency, add the underscore at the end of these variable's names.

Status

Issue addressed here: maple-labs/mplv2#45

3S-MPLV2T-N02

The **`MapleTokenProxy`**, **`setImplementation()`** is missing an optional argument to initialize a new implementation

Id	3S-MPLV2T-N02
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

Usually when a new implementation is set in a proxy, there is an optional argument with the arguments of an initialize function encoded as bytes. In the `MapleTokenProxy`, it is not available, which could make it more complicated if a new implementation that needs to be initialized is to be used.

Recommendation

Add an argument **`bytes memory initializationArgs_`** to **`setImplementation()`** and if its length is different than 0, call the function **`initialize()`** of the new implementation.

Status

For the Non-transparent-proxy the team would like to keep to the same pattern that they have with their MapleGlobals, they don't expect to be calling any initialization function in future implementations. Also in their pattern they tend to have a separate initializer contract and for that logic not to live on any implementation. Therefore they decided not to take action on this issue.

3S-MPLV2T-N03

The **ERC20Proxied** should leave a **gap** to allow future upgrades without worrying about storage positions

Id	3S-MPLV2T-N03
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

The **ERC20Proxied** contract is supposed to be inherited by the **MapleToken** contract. If in the future, with the code as is, if Maple wants to deploy a new token using a modified **ERC20Proxied**, with new state variables, these would have to be inserted at the top of the storage (after the variable **isModule** in the **MapleToken**).

Recommendation

Similarly to Openzeppelin's implementation, place the gap at end of the **ERC20Proxied** implementation:

```
abstract contract ERC20Proxied {
    ...
    uint256[43] private __gap;
}
```

Status

If the team would update anything for the deployed MapleTokenProxy they would most likely just add to the storage of the MapleToken and override the ERC20Proxied functions in the MapleToken. Therefore they decided to leave as is.

3S-MPLV2T-N04

In **`RecapitalizationManager`**, use cached variables to save on gas

Id	3S-MPLV2T-N04
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

In the **`schedule()`** function the new window id is fetched initially `` `uint16 newWindowId_ = lastScheduledWindowId + 1;` ``. Then at the end of the function, **`lastScheduledWindowId`** is updated `` `lastScheduledWindowId += newWindowCount_;` ``, which reads the same value from storage again.

Also in the function **`claimable()`** the **`lastClaimedTimestamp`** is cached in `` `lastClaimedTimestamp_` ``. Then when calling the `` `_claimable()` `` function it reads the variable from storage instead of using the cached one.

Recommendation

Update **`lastScheduleWindowId`** doing:

```
lastScheduledWindowId = newWindowId_ + newWindowCount_ - 1;
```

and update **`_claimable()`** doing:

```
( lastClaimableWindowId_, claimableAmount_ ) =
_claimable(lastClaimedWindowId, lastClaimedTimestamp_, to_);
```

Status

Issue addressed here: maple-labs/mplv2#47

3S-MPLV2T-N05

Remove memory variable that is written to but never used

Id	3S-MPLV2T-N05
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

In **`RecapitalizationModule`** the function **`claimable()`** creates the variable **`lastClaimableWindowId_`** and writes to it when calling `_claimable()`. Since this variable is never read it doesn't need to exist and you can call **`_claimable()`** as

```
(, claimableAmount_ ) = _claimable(lastClaimedWindowId,
lastClaimedTimestamp, to_);
```

Status

Issue addressed here: maple-labs/mplv2#48

3S-MPLV2T-N06

In **`MapleToken`**, the modules could be split into **`mint`** and **`burn`** to reduce attack surface

Id	3S-MPLV2T-N06
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

The **`MapleToken`** allows a module to mint and burn tokens. Some modules, such as the **`RecapitalizationModule`**, only do one of these 2 actions, such that minting and burning permissions could be split into 2 different roles.

Status

The initial design included separation of mints and burns, but the team felt that modules are highly trusted already, so that it is not worth adding the extra logic.

3S-MPLV2T-N07

In **`RecapitalizationModule`**, remove unnecessary check

Id	3S-MPLV2T-N07
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

In the **`RecapitalizationModule`** on the `_validateWindows()` function the first require doesn't need to check if both arrays length is larger than 0. You only need to check one since afterwards you check if the length mismatch. So if one isn't 0 the other can't be 0, and removing the second check will save some gas.

Status

Issue addressed here: maple-labs/mplv2#49

3S-MPLV2T-N08

In the **`RecapitalizationModule`**, in function **`_claimable()`**, **`lastClaimableWindowId_`** update can be skipped if the **`windowId_`** has not changed

Id	3S-MPLV2T-N08
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

`claim()` gets the current claimable amount and always updates the `lastClaimableWindowId_`. As the windows can have any duration (expected 1 year according to the wiki), it's likely that in some calls, the update is only a storage rewrite.

Recommendation

In function **`claim()`**, cache **`lastClaimedWindowId`** before sending it to **`_claimable()`** and then only write to **`lastClaimedWindowId`** if the previously cached value differs from the new value.

Status

The team would rather keep the code tight than to add a new if statement to prevent a re-write. This function will be called by the DAO and not very often, so the additional write is not a concern.

3S-MPLV2T-N09

In **`MapleToken`**, **`addModule()`** and **`removeModule()`** could revert if the call leads to a stale update

Id	3S-MPLV2T-N09
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

In **`MapleToken`**, functions `addModule()` and `removeModule()` do not verify if they are actually updating the **`isModule`** mapping. An explicit check could be added for this to prevent unexpected behaviour and emitting a duplicated **`ModuleAdded`** or **`ModuleRemoved`** event. The extra gas costs should be worth this addition.

Status

The team doesn't feel this addition is necessary and would rather leave the code as is.

3S-MPLV2T-N10

In the **`RecapitalizationModule`**, it's possible to schedule a window with duplicated **`windowStart`**

Id	3S-MPLV2T-N10
Classification	None
Severity	None
Likelihood	
Category	Bug

Description

The `claim()` call will set **`lastClaimedWindowId`** to the most recent window. If a `schedule()` call is placed in the same **`block.timestamp`**, and the **`lastClaimedWindowId`** has this same **`windowStart`**, then a window is scheduled with the same **`windowStart`** as the previous one. It does not have any effect on the outcome of the **`RecapitalizationModule`**, but it's something to keep in mind.

For example,

W1 -- W2, where W2 has **`windowStart`** 1000

at **`block.timestamp == 1000`**, call **`claim()`**

`lastClaimedWindowId` is W2. At the same **`block.timestamp`** call **`schedule()`** with W3 having **`windowStart`** 1000

W1 -- W2 -- W3, as **`_findInsertionPoint()`** starts at **`W2`** and returns it (`windowStart_ <= window_.windowStart`).

But then, as can be seen in the tests, there is no real impact. The current issuance rate is the introduced one, scheduling will replace the introduced window and claiming will return the correct value, given that the interval between W2 and W3 is 0, so is the claimable amount.

Here is a test illustrating this behaviour.

Recommendation

Place a check in **`schedule()`** that the **`windowStart`** of the first introduced window is bigger than the insertion point. This should add very little gas overhead because this window was already fetched from storage to check the `windowStart`.

Status

Issue addressed here: maple-labs/mplv2#50

3S-MPLV2T-N11

Common constants or pieces of code to several files can be joined together in a library, preventing mistakes

Id	3S-MPLV2T-N11
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

Sometimes several Smart Contracts implement the same constants or functionality. In this case, most of the time it's better to handle the duplicated logic in the same place. This avoids potential mistakes of having different logic across Smart Contracts when it should be the same. One way to achieve this is using a library with common internal functions or constants.

Recommendation

As an example, the library **`Constants`** can be defined in a **`Constants.sol`** file:

```
library Constants {
    bytes32 internal constant GLOBALS_SLOT        =
bytes32(uint256(keccak256("eip1967.proxy.globals")) - 1);
    bytes32 internal constant IMPLEMENTATION_SLOT =
bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1);
}
```

Then, to use it, do

```
bytes32 internal constant GLOBALS_SLOT        = Constants.GLOBALS_SLOT;
bytes32 internal constant IMPLEMENTATION_SLOT =
Constants.IMPLEMENTATION_SLOT;
```

The following pieces of code can be grouped together:

1. **`GLOBALS_SLOT`** and **`IMPLEMENTATION_SLOT`** in `MapleToken` and `MapleTokenProxy`.

2. The body of the modifier **`onlyScheduled`** in `MapleToken`, the body of the function **`setImplementation()`** in `MapleTokenProxy` and in the `InflationModule`.

Status

The team doesn't feel this addition is necessary and would rather leave the code as is.

3S-MPLV2T-N12

The compiler does not reuse the body of the modifiers, leading to excess bytecode and higher deployment costs and contract size

Id	3S-MPLV2T-N12
Classification	None
Severity	None
Likelihood	
Category	Optimization

Description

When using modifiers, the code in their body is written to bytecode everytime the modifier is used (the compiler does not create a separate code block and jump there when called), which leads to excessive bytecode and deployment costs.

Recommendation

When the modifier is used more than once, create an internal function that handles its logic. This reduces the deployment costs significantly and only increases the runtime cost by 1 jump operation.

This pattern was recognized in the following places:

1. **`onlyScheduled`** and **`onlyGovernor`** in **`MapleToken`** in functions `` `addModule()` `` and `` `removeModule()` ``.

Status

The team is not particularly worried about deployment costs and decided not to take action on this issue.

3S-MPLV2T-N13

Use the **`onlyGovernor`** modifier in the **`MapleTokenProxy`**, similarly to the **`MapleToken`** and **`RecapitalizationModule`**

Id	3S-MPLV2T-N13
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

In **`MapleTokenProxy`**, the function **`setImplementation()`** checks if **`msg.sender`** is the **`governor`** in its body, but in `MapleToken` and `RecapitalizationModule`, there is a modifier to check it. Consider using the same approach on both instances for better readability and consistency.

Status

In the MapleTokenProxy there's just a single function that needs it, so the team feels it's cleaner to have it in the function body.

3S-MPLV2T-N14

Place hardcoded values as constants to increase readability

Id	3S-MPLV2T-N14
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

Placing hardcoded values as constants instead of inline code increases readability.

Recommendation

The following values are hardcoded and could be refactored:

1. The initial mint amounts in the **`MapleTokenInitializer`**.

Status

Issue addressed here: maple-labs/mplv2#46

3S-MPLV2T-N15

In **`MapleToken`**, function **`implementation()`** can be marked external

Id	3S-MPLV2T-N15
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices

Description

For better readability, as the function **`implementation()`** is never called internally, it's best to set its visibility to **`external`**. The gas costs seem to be the same for both cases, having tested with **`forge test --gas-report`**.

Status

Issue addressed here: maple-labs/mplv2#44

3S-MPLV2T-N16

Use the latest solidity version

Id	3S-MPLV2T-N16
Classification	None
Severity	None
Likelihood	
Category	Suggestion

Description

The codebase is using an outdated solidity version (0.8.18). It is recommended to always use the latest version which has fixes for known bugs and optimizations (**`push0`** opcode in 0.8.20).

Recommendation

Update the solidity version to the latest available (0.8.21 at the time of writing).

Status

The team would rather play on the conservative side and not use a too recent solidity version, so they chose to stay with the 0.8.18 version.