Set Protocol v2 Audit

OPENZEPPELIN SECURITY | SEPTEMBER 4, 2020

Security Audits

Set Protocol provides a mechanism for a single ERC20 token to represent a combination of other tokens on Ethereum. Version 2 of the protocol has the same goal but changes the architecture. We started auditing commit `59d62e16ecee32bb6c5dcc87dd47392cc3427171` and then we switched to commit `b8286f431547823ff3935a1343cd4cf4d77585a1`, when the original commit was partially obsoleted. It should be noted that these commits (and the corresponding links in the report) refer to a private repository and may not be publicly accessible. The scope includes the following contracts:

- `contracts/interfaces/IController.sol`
- `contracts/interfaces/IManagerIssuanceHook.sol`
- `contracts/interfaces/IOracle.sol`
- `contracts/interfaces/IOracleAdapter.sol`
- `contracts/interfaces/ISetToken.sol`
- `contracts/lib/AddressArrayUtils.sol`
- `contracts/lib/ExplicitERC20.sol`
- `contracts/lib/PreciseUnitMath.sol`
- `contracts/protocol/Controller.sol`
- `contracts/protocol/PriceOracle.sol`
- `contracts/protocol/SetToken.sol`
- `contracts/protocol/SetTokenCreator.sol`
- `contracts/protocol/IntegrationRegistry.sol`
- `contracts/protocol/lib/ModuleBase.sol`

- `contracts/protocol/modules/StreamingFeeModule.sol`

All external code and contract dependencies were assumed to work as documented.

We reviewed the contracts with a team of 2 auditors over the course of 2.5 weeks.

Update

Many of the issues listed below have been fixed by the Set Protocol team. The remaining issues are either acknowledged and accepted, or included in the roadmap. Our analysis of the mitigations is limited to the specified pull requests and disregards all other unrelated changes to the code base.

Here we present our findings, along with the relevant updates for each of them.

Summary

Overall, we are happy with the design and health of the code base. We are pleased to see the use of small, encapsulated functions and isolated contracts. We also appreciate the extensive documentation and testing suite.

System Overview

The protocol is designed to be modular and extensible. To that end, each `SetToken` contract only contains core functions to change the distribution of external (component) ERC20 tokens that the set represents, and to mint and burn the aggregating Set tokens. Users must interact with module contracts that attach to Sets to define the logic of token issuance, redemption and Set modifications.

Every set has a module that allows users to deposit the component tokens in exchange for Set tokens. Subsequently, Set token holders can redeem them in exchange for the underlying ERC20s that the tokens represent. In this audit, we reviewed this module as well as another module that allows a Set manager to collect a percentage of the Set value every block. In the future, new modules will be introduced to support trading and integration with exchanges.

different parts of the system to recognize each other so the system can grow over time.

Privileged Roles

Many of the contracts have privileged roles that can significantly affect the usefulness and safety of the system. Specifically:

- The `Controller` contract has an owner that chooses the contracts that comprise the system. This includes all of the modules, resources and the factories that can be used to make Sets. The owner also has the ability to remove Sets as desired, and specify protocol fees and other fee types that conforming modules will pay. Moreover, user deposits are achieved by granting token allowances to the Controller, so a malicious owner could deploy a module that simply takes these tokens. Naturally, this is a very powerful role and all users must trust the governance mechanism to use these powers fairly and wisely.
- The `IntegrationRegistry` tracks third party integrations that can be used in the system. It also has an owner that can add, remove and edit the integrations as desired. The audited code does not use the registry, but as the protocol grows, the governance mechanism that controls the registry will be able to effect the usefulness and safety of dependent modules.
- The `PriceOracle` has a list of oracles and adapters that can be used to retrieve third party prices. It also has an owner role that can choose the supported oracles, adapters and price pairs. The audited code did not rely on this contract, but naturally any module that requires prices is implicitly trusting the governance mechanism to choose the oracles and adapters fairly.
- Anyone can use the system to deploy a new `SetToken` contract with a list of component ERC20 tokens and their distributions. The protocol enforces some functional restrictions and recommendations but this is essentially a free choice. A malicious or poor choice of token contracts can undermine the value of the Set, so it is up to the set creators to choose sensible component distributions, and users will need to trust the creator or validate the choices.
- Lastly, the set creator can choose an arbitrary manager. The manager has the ability to select any combination of modules (from a list approved by the `Controller`'s owner), and modify the combination while the Set is operational. The modules themselves may also

percentage and recipient. It is expected that future modules will allow the manager to rearrange the component distributions within the Set, and users must trust them to make fair and sensible choices.

Ecosystem Dependencies

As the ecosystem becomes more interconnected, understanding the external dependencies and assumptions has become an increasingly crucial component of system security. To this end, we would like to discuss how the Set Protocol V2 depend on the surrounding ecosystem.

Other than the interaction with arbitrary ERC20 tokens, the audited code does not have any third party dependencies. However, it contains architectural components that support extensibility. The `IntegrationRegistry` will contain an explicit list of third-party integrations and, naturally, they will affect any Set that has a module that uses them. Since we didn't review any integrations, the particular dependencies are unknown.

Separately, the `PriceOracle` contract provides a mechanism to introduce asset prices into the system. It should be noted that the contract simply returns the first available price, so any mechanisms to aggregate prices or prevent manipulation are deferred to the external oracle contracts.

Lastly, the `StreamingFeeModule` uses time-based logic to calculate and implement the manager fee. This means that fee accumulation is subject to Ethereum availability, and very long delays between updates (for instance, during periods of high Ethereum congestion) will reduce the compounding rate. In practice, this is unlikely to be a significant effect.

Critical severity

None.

High severity

None.

The `StreamingFeeModule` allows the manager to set a zero fee. However, when it is subsequently changed, the `accrueFee` function will revert, making it impossible to change the fee to a non-zero value. Consider simply updating the `lastStreamingFeeTimestamp` and returning successfully when attempting to accrue a zero fee.

Update: *Fixed in PR#118.*

[M02] Bypass contract invariants

The `SetToken` functionality is spread across multiple modules, which undermines the reentrancy protection. To see this, consider the `BasicIssuanceModule` redemption code, which uses a strict transfer to ensure that the transfer does not collect additional fees. This is achieved with a guard condition that checks the `TokenSet` contract's component balance before and after each transfer.

On the other hand, if the component token does collect a fee and also invokes receive hooks (for instance, if it is an ERC777 token), the recipient will have an opportunity to run arbitrary code to compensate for the fee. To do so, they would need a mechanism to increase the `SetToken`'s component balance in a way that does not relinquish control of the tokens. In particular, if the recipient were able to use the receive hook to issue new Set Tokens, they could increase the `SetToken` balance to counteract the fee. In this case, the `nonReentrant` modifier on both functions prevents this attack.

However, the mitigation only works because both `nonReentrant` functions are on the same contract. In contrast, if the streaming fee is accrued during a redemption event using the same mechanism, different components would be redeemed at different conversion rates. Since the two relevant functions are on different contracts, the `nonReentrant` modifier would not be a mitigation.

Since managers can choose any arbitrary components, the protocol ultimately relies on the due diligence of managers and users to avoid these complex and fragile scenarios. This could be avoided by choosing components that do not invoke send or receive hooks. Nevertheless, we

Since the internal state can be modified by multiple system contracts (for example, component tokens are sent from the `SetToken` contract but accepted from the `Controller` contract), a simple Reentrancy Guard does not cover all cases. Consider implementing a system-wide reentrancy guard, possibly coordinating through the `Controller` contract, whenever external tokens are sent and received. Additionally, to limit the attack surface, consider including this protection on the privileged functions as well as the public ones.

[M03] Unstable Functionality

Each `SetToken` manager has the ability to add new modules to modify the functionality of the Set. They are restricted to modules that are recognized by the `Controller`, but this still provides opportunities to defraud users.

For example, they could bypass their agreed maximum streaming fee by removing the module and then adding it again with a higher bound. Alternatively, they could add the `StreamingFeeModule` to a Set that was not supposed to have it. In either case, users would be charged a fee that they did not agree to.

Consider introducing a time delay before transitioning modules from the `PENDING` to `INITIALIZED` state so users have the option of opting out of any disagreeable changes.

[M04] Additive rounding errors

The `calculateDefaultEditPositionUnit` determines the new position unit based on the change in the total notional component tokens held. However, using the change in position may incur excessive rounding errors, particularly if the change is small. Additionally, multiple small changes will cause these errors to accumulate. Consider calculating the new position unit directly using the `_postTotalNotional` and `_setTokenSupply` values.

[M05] Uncontained Token Errors

In the `BasicIssuanceModule`, redeeming Set tokens is achieved by looping through all the components and sending them to the `_to` address. If any of the token transfers fail, the whole operation is reverted. This could occur if one of the transfers is subject to a fee, a timelock, a blacklist, or any other non-standard (or incorrectly implemented) features. This means that the

Since managers can choose any arbitrary components, the protocol ultimately relies on the due diligence of managers and users to avoid these complex and fragile scenarios. Nevertheless, we believe that a robust protocol should limit the consequences of unexpected behavior wherever possible.

To that end, consider handling redemption as an internal accounting operation, where users can subsequently withdraw any subset of the components to which they are entitled. This would allow users to recover some of the value of poorly created Sets.

Low severity

[L01] Incorrect Event

In the `BasicIssuanceModule`, the `SetTokenIssued` event always sets the `_hookContract` variable to zero, even if the hook exists. Consider setting the parameter to the relevant hook contract.

***Update:** Fixed in commit 65b787f. We did not review this commit except to note that it fixed this particular issue.*

[L02] Parameter Validation

The following is a list of code snippets that could benefit from additional validation:

- The `initialize` function of the `Controller` contract does not check for duplicates in the passed `_factories`, `_modules` or `_resources` arrays. This could lead to inconsistencies in the internal state, particularly if any of the items are removed. Consider checking for duplicates before assigning these arrays.
- Similarly, the `PriceOracle` constructor does not check for duplicates in the passed `_adapters` array
- The `addFee`, `editFee` and `editFeeRecipient` functions of the `Controller` contract do not prevent a zero fee recipient with a non-zero fee. If this occurs, the `mintTraderAndProtocolFee` function of the `StreamingFeeModule` contract will revert for any non-zero fee. This will effectively disable the module.

functions should not depend on external validation. Consider explicitly checking that the array has at least one element, or using `SafeMath` for the subtraction.

- Similarly, the `pop` function of `AddressArrayUtils` implicitly assumes the array is not empty and that the `index` is within the array. Consider explicitly checking these conditions.
- The `SetToken` contract does not perform consistency checks when positions are changed.
- the `addComponent` function does not validate that the input is non-zero, or that it doesn't already exist in the `components` array.
- the `addExternalPositionModule` function does not validate that the component is in the `_components` mapping or if the module already exists in the `externalPositionModules` array. It also does not add any new position to the `externalPositions` mapping but the existence of the module is used to count the positions.
- the `editExternalPositionUnit` function does not validate that the component or position exist.

Update: *Partially fixed in PR#118. The* `editFeeRecipient` *now ensures new fee recipients are non-zero. For completeness, the constructor should also ensure the fee recipient is not initialized to zero. The* `hasDuplicate` *and* `pop` *functions of* `AddressArrayUtils` *now perform the recommended validations.*

[L03] Obsolete Artifacts

The protocol is comprised of several different contracts that redundantly track the inter-contract trust relationships. However, there are scenarios where they can get out of synchronization, leaving obsolete records across the system and potentially disabling other contracts.

In particular, if a Set, Factory, Module or Resource is removed from the `Controller` contract, the controller is not also removed from the system contract, which will still respond to some queries and appear to be functional. This may be acceptable because the system contract is now defunct and its state is no longer relevant. However, it does leave the possibility that the defunct contract will be added to the controller again in its current state.

Lastly, if a module is removed when it has locked a `SetToken`, it will be unable to unlock the token, leaving the token contract disabled.

Consider including `shutdown` or `cleanup` functions to allow contracts to remove obsolete records, which will also ensure that renewed system contracts are reintroduced into the protocol in a fresh state.

[L04] Incorrect calculation

The `preciseDivRoundAwayFromZero` function of the `PreciseUnitMath` library returns zero when dividing a non-zero value by zero instead of reverting. Consider simplifying the input validation of this function (and the `preciseDivCeil` function) to only revert if `b` is zero.

***Update:** Fixed in PR#115. The subsequent condition can now be simplified, although the function no longer appears in the latest version of the contract.*

[L05] Missing NatSpec comments

Many functions in the code base have excellent Ethereum Natural Specification Format (NatSpec) comments. However, here are some examples of missing or incomplete comments:

- Most functions in the `SetToken` contract do not have any NatSpec comments.
- The `remove` and `pop` functions of `AddressArrayUtils` do not list their parameters.
- The `getDefaultPositionUnit` and `calculateDefaultEditPositionUnit` functions of the `Position` contract do not have `@return` comments.

Consider reviewing the code base for missing or incomplete NatSpec comments and adding them where appropriate.

***Update:** Partially fixed in PR#118. The specified `AddressArrayUtils` and `Position` NatSpec comments have been added.*

[L06] Index Event Parameters

To facilitate off-chain searching and filtering for specific events, consider indexing the `_issuer`, `_redeemer` and `_to` parameters of the `SetTokenIssued` and `SetTokenRedeemed`

[L07] Stranded Modules

Before a module is initialized on a `SetToken`, it transitions through the `PENDING` state. However, only initialized modules can be removed. This means that the `SetToken` manager does not have a mechanism to remove obsolete modules in the `PENDING` state that cannot or will not be initialized. To avoid this edge case, consider introducing a mechanism to remove a pending module.

***Update:** Fixed in PR#118.*

Notes & Additional Information

[N01] Misleading Comments

- The streaming fee validation error message says the fee must be less than the maximum, but it could be equal to the maximum.
- The comment on the `contains` function of `AddressArrayUtils` library suggests that its return value depends on the first occurrence of the value and the indexing scheme. In fact, it just returns a boolean.
- The `ModuleBase` constructor comment claims it maps asset pairs to their oracles but that does not apply to this function.
- The comment on the `getDefaultPositionalUnit` function of the `Position` contract is an incomplete thought.
- The comment describing the `StreamingFeeModule` inflation formula has the brackets in the wrong position. The left hand side of the equation should be “feeQuantity / (feeQuantity + totalSupply)”
- The inflation calculation incorrectly reports the scale factor as 10e18 instead of 1e18.

Consider updating the comments accordingly.

***Update:** Partially fixed in PR#118. The streaming fee validation error message has been corrected.*

[N02] Inconsistent naming of internal functions

`_defaultPositionVirtualUnit` function in the `SetToken` contract and the `_getDirectOrInversePrice` function in the `PriceOracle` contract.

However, there are some instances where the internal functions do not start with an underscore, for example the `nameHash` function in the `IntegrationRegistry` contract and the `isSetPendingInitialization` function in the `ModuleBase` contract.

Consider using a consistent style to improve the readability of the code.

[N03] Inherit interfaces

The code base implements interfaces to allow contracts to interact with each other. However, the contracts do not inherit their own interfaces. For clarity and consistency, consider explicitly inheriting the relevant interface contracts, which will allow the compiler to confirm that they are implemented correctly.

[N04] Messy code

- The `editDefaultPosition` function of the `Position` contract executes the same update line on all three branches of the `if` statement. Consider moving it outside the branches.
- The `SetToken` `components` array is initialized by pushing every element of another array. Consider using a direct assignment.
- The `PreciseUnitMath` library has multiple examples of a complicated same-sign conditional. This would be clearer if there were brackets around each half of the condition.
- The `SetToken` contract has multiple examples of a redundant non-empty array check before looping through the array. The loop condition already handles the empty array case.

Update: *Fixed in PR#118. The pull request does not address the `PreciseUnitMath` conditional but it no longer appears in the latest version of the contract.*

[N05] Unused import statements

Consider removing the unused import of IERC20 in the BasicIssuanceModule contract.

Update: *Fixed in PR#118. The import is now used.*

suggestions:

- The `onlyValidInitialization` modifier of the `ModuleBase` contract should be `onlyValidAndPendingSet` for consistency with `onlyValidAndInitializedSet`
- The `_mintTraderAndProtocolFee` function of the `StreamingFeeModule` should be `_mintManagerAndProtocolFee`

***Update:** Fixed in PR#118.*

[N07] Extended attack surface

The new version of the protocol assigns user funds to the relevant `SetToken` contract instead of a shared vault. This helps to isolate `SetToken` contracts from each other.

Following the same principle, consider allowing `SetToken` contracts to transfer user funds to themselves using their own allowances, rather than through the `Controller` allowance. This would eliminate the potential for a module to spend user funds that were intended for an unrelated Set.

We should emphasize that we did not identify a mechanism to achieve this using the audited contracts. The recommendation is simply based on the principle of reducing the attack surface.

***Update:** Fixed in PR#119. The expected change was for each Set to have its own allowance, but instead incoming transfers now use the module's allowance. This still prevents malicious modules from spending funds on behalf of other modules, and it allows greater flexibility when users interact with multiple Sets. Since a given module will be shared across multiple `SetToken` contracts, it is still possible for a module to spend user funds intended for an unrelated Set, but well-written modules would prevent this.*

[N08] Invoke Return Value

The `Invoke` library contains utility functions to invoke particular functions on a `SetToken` contract. In all cases, the return value is discarded. For maximum generality, consider forwarding the return value to the caller so they can react to it, if desired.

[N09] Use delete to clear variables

The `delete` key better conveys the intention and is also more idiomatic. Consider replacing assignments of zero with `delete` statements.

Update: *Fixed in PR#118.*

[N10] OpenZeppelin Contract's dependency is not pinned

To prevent unexpected behaviors in case breaking changes are released in future updates of the OpenZeppelin Contracts's library, consider pinning the version of this dependency in the package.json file.

[N11] Typographical errors

Consider fixing the following typographical errors:

- Throughout the `SetToken` and `PriceOracle` contracts, "privileged" is misspelled as "priveleged"
- In line 29 of `ExplicitERC20`, "transferred" is misspelled as "transfered"
- In line 54 of `StreamingFeeModule`, "manager" is misspelled as "maanager"
- In line 59 of `AddressArrayUtils.sol`: "occurrence" is misspelled as "occurence"
- In line 99 of `PreciseUnitMath.sol`: there should be a space between the sentences
- In line 215 of `Controller.sol`: "and" should be "an"
- In line 446 of `SetToken.sol`: "purposes" is misspelled as "purposses"
- In line 29 of `IntegrationRegistry.sol`: "connected" is misspelled as "conected"
- In line 134 of `Position.sol`: "accidentally" is misspelled as "accidentically"

Update: *Partially fixed in PR#118. The `ExplicitERC20.sol`, `StreamingFeeModule` and the "privelege" typographical errors have not been addressed.*

Conclusions

No critical nor high severity issues were found. Several recommendations were made to improve the project's overall quality and reduce its attack surface.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin