Invariant

Pondora v2

Solo Audit Report v1

June 5, 2025

Contents

Revision table

Report version	Report name	Date	Report URL
1.0	Main audit	2025-06-05	Full report link

1 Executive summary

Project overview

Pondora v2 brings smart account functionality to Cardano. A user can lock their assets in a *Pond* smart contract and approve a set of instructions called *intents*; they say how, when, and under what conditions a part of those funds may be used. Think of it as a programmable vault: you decide the rules on-the-go and let somebody else execute them for you when the conditions are right, potentially for a fee.

This audit is about the base Pondora contract which holds the funds and decides whether intents are authorized and then delegates the validation to them. The audit also includes a few initial *modules* that are configurable validators executing these intents:

- **Payment unlock**. A module that releases exactly the amount required for an on-chain payment while returning the remainder to the smart account.
- **Defragmentation**. A module that merges many potentially smaller UTxOs into one to keep the account tidy and reduce future fees. There is a set minimum of how many UTxOs are required to be consolidated.
- **Dependencies module**. A module that allows users to incentivize batchers to execute their intents by adding a service fee which unlocks upon the execution of a specified set of other intents — the dependencies. The fee can be unlocked only if all of the specified intents are executed in the same transaction. However, the intents need to be separately authorized and so it is not a module that can be used to atomically combine intents.
- **Nonce helper**. A helper that allows a user to make a given intent executable only once. The intent has to support it.

Intents are bound to certain user UTxOs by *labels*. Labels can be arbitrary strings set in the datums. If they are not set this way, the balance can be spent by a best-effort estimate of the wanted label. All intents are authorized with a certain label in mind — this effectively binds the intent to a specific subset of the user's UTxOs. An intent is not executable if the label does not match. It is best to always set the label explicitly in the datums.

Intents can also be authorized for use with a *wildcard label*, which allows them to be used with any label at all and therefore should not be overused. All modules explicitly either allow or disallow wildcard support. Of the audited modules, only the defragmentation module supports wildcards; it can be authorized for use with any label. However, defragmentation still occurs per *(owner, label)* pairs.

Intents are approved either on-chain or off-chain. They are all aggregated into compact data structures called Merkle trees. A user signs off a root of his tree, a compact representation of all the intents he authorized. The whole tree is stored off-chain only. A user should sign only root hashes he trusts and the platform should provide a way to verify the root hash against the off-chain tree. It is important to note that signing off an arbitrary not-verified root hash can authorize a lot of intents that could potentially be used to empty the account, both instantly and later in time. The team plans to show enough details in the UX for technical users to be able to verify the authenticity of the root hash. I recommend doing it — especially for larger accounts.

The off-chain authorizations have been improved over the course of the audit to include a mandatory limited validity period, as they are generally irrevocable. The on-chain authorizations use a smart contract holding the last validated root hash and so intents authorized this way are revocable at any time. As such, it is recommended to ensure tight-enough validity periods are set or to prefer on-chain authorizations even though there are network fees associated with them.

Even though the execution of the intents can potentially be done by anyone, at launch, the Pondora team will run a batcher that will collect intents from multiple users and settle them over fewer bigger transactions. Such a batcher is necessary in the beginning since the data including the intents themselves is not fully publicly accessible yet until the Decentralized Order Routing Aggregator (*DORA*) is launched.

Audit overview

I started the audit at commit *7ea31ffffa* and it lasted from April 7, 2025 to May 15, 2025. The timeframe is inclusive of periods in which I was awaiting the implementa-

tion of fixes by the client. We interacted mostly on Discord. The team fixed all issues to my satisfaction, except for 1 medium that was acknowledged as intended, 2 minor and 6 informational findings that were either acknowledged or partially resolved. They do not represent security threats to the system and can be mitigated by proper expectation management and communication. I found the team highly responsive and collaborative throughout the audit process, and the documentation provided was of good quality, which significantly aided my understanding of the system.

The scope of the audit was limited to the smart contract files only. I did not review any tests as part of this audit. I performed a design review along with a deep manual audit of the code and reported findings along with remediation suggestions to the team in a continuous fashion, allowing the time for a proper remediation that I reviewed afterwards.

There were multiple significant design refactors during the audit period, including a change in the ownership model to recognize the stake credential as the source of truth and a big redesign of the “and” module that resulted in it being replaced by the dependencies module. There were also multiple big performance inspired refactors and a number of issues that were quickly caught and fixed that are not included in this report. See more about our methodology in Methodology.

The commit `bf21576ffd7cca45f4e59ec0c9460ca63d361394` represents the final version of the code. The status of any issue in this report reflects its status at that commit. You can see all the files audited and their hashes in Audited files. The smart contract language used is Aiken and the contracts are intended to run on Cardano. To avoid any doubt, I did not audit Aiken itself or the external `merkle-patricia-forestry` library, which were assumed to function correctly.

Summary of findings

During the audit, I identified and reported 9 critical, 3 major, 2 medium, 7 minor, and 10 informational findings. For details on severity and status classification, please refer to Classification.

All critical and major findings were fixed prior to the final commit. Nine findings (one medium, two minor and six informational) were either acknowledged as acceptable trade-offs or only partially resolved:

- **POND2-202**: Defragmentation of native token balances unlocks min Ada within. This was acknowledged as an intended source of revenue for the batcher. There are no issues as long as the communication from the team is clear and the expectations are managed.

- **POND2-305**: Near-full and small spends of Ada UTxOs can be troublesome. This was partially resolved by allowing for full spends. The edge cases of near-full and very small spends can be mitigated by additional deposits.
- **POND2-307**: Unlock module vulnerable to cross-script double satisfaction. This was acknowledged as acceptable since users are expected to set unique identifiers when making payments. If they do, the issue is mitigated and there is no double satisfaction present.
- **POND2-401**: Staking credential registration validation will be required soon. This informational finding was acknowledged without immediate action as it will only affect future Cardano eras, not this one. As long as the deployment is done before that, there is no issue.
- **POND2-403**: Pond unlock *`ByOwner`* contains additional fields. This was acknowledged, with the fields kept for schema consistency reasons.
- **POND2-404**: Nonce helper can check that it is registered just once. This was acknowledged as it is not a problem as long as timestamps are set correctly.
- **POND2-406**: Naming and dead code issues. This was partially resolved, with naming improved but some unused functions retained for future use.
- **POND2-407**: Code style suggestions. This was partially resolved with variable shadowing fixed and custom functions kept for performance reasons, but code duplication was not addressed.
- **POND2-409**: Intents using nonce could be blocked if nonce value is found. This was acknowledged as an intentional design tradeoff to save execution costs. However, once *`DORA`* is launched, the issue will be more pronounced as the nonce values will be public and I suggest reconsidering after that. The team has acknowledged this and has indicated that the module will be completely replaced as part of the transition to *`DORA`*.

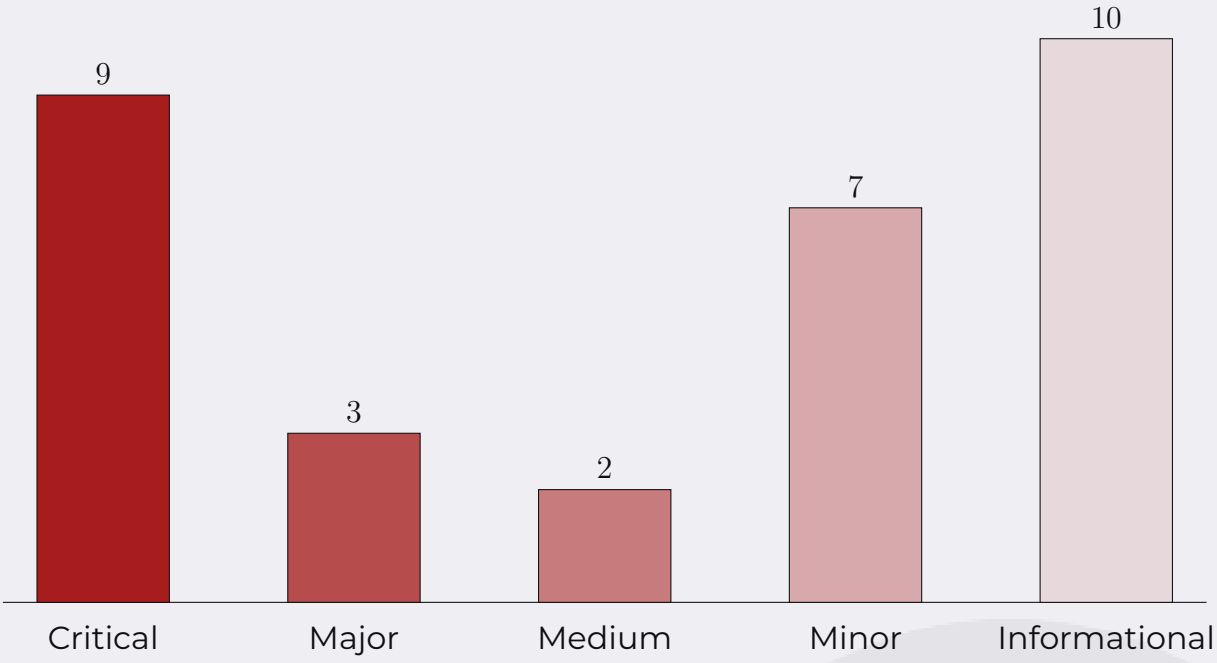
Overall, the reported findings can be broadly categorized as below:

- **Special cases of Cardano, the protocol, or malicious transaction construction:** Issues related to Cardano edge cases (POND2-001), special transaction construction (POND2-003, POND2-009) or protocol edge cases (POND2-002, POND2-301, POND2-302, POND2-303, POND2-304, POND2-306).
- **Protocol re-design:** The original “and” module required a complete redesign and was replaced by the dependencies module (POND2-103). Similarly, some signatures could not be revoked and a validity was added (POND2-201).

- **Ownership model ambiguities:** The source-of-truth ambiguities of pond UTxO ownership causing high-impact issues (POND2-004, POND2-005, POND2-006).
- **Logical or code oversights:** Code oversights in iteration logic that created vulnerabilities (POND2-007, POND2-008) as well as incorrect label handling (POND2-101).
- **Denial of Service vectors:** Burning on-chain intent tokens or nonce front-running (POND2-102, POND2-409).
- **Documentation and code style:** Non-critical improvements for clarity and maintainability (POND2-402, POND2-403, POND2-404, POND2-405, POND2-406, POND2-407, POND2-408).

Overall, remediation efforts have markedly improved Pondora v2's security posture.

2 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
POND2-001	Nonce helper script may be bypassed	CRITICAL	RESOLVED
POND2-002	Nonce helper validator self-reference	CRITICAL	RESOLVED
POND2-003	Account helper does not prevent unauthorized drain of funds	CRITICAL	RESOLVED
POND2-004	Address-based accounting lets an attacker steal funds	CRITICAL	RESOLVED
POND2-005	All funds can be stolen by pretending to be an owner of any UTxO	CRITICAL	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
POND2-006	Value can be stolen as anyone can sign intent authorization	CRITICAL	RESOLVED
POND2-007	Pond native validator skips all validation after an unknown redeemer	CRITICAL	RESOLVED
POND2-008	Service fee module stops processing after the first match	CRITICAL	RESOLVED
POND2-009	Service fee module observes intents from non-pond scripts	CRITICAL	RESOLVED
POND2-101	Arbitrary label injection when a label is present in the datum	MAJOR	RESOLVED
POND2-102	Attacker can repeatedly DoS intent authorizations and keep Ada	MAJOR	RESOLVED
POND2-103	Modules do not support indirect calls s.a. the *and* module's	MAJOR	RESOLVED
POND2-201	Signed intents can not be revoked once issued	MEDIUM	RESOLVED
POND2-202	Defragmentation of native token balances unlocks min Ada within	MEDIUM	ACKNOWLEDGED
POND2-301	Nonce helper overwrites a valid match if it almost finds another	MINOR	RESOLVED
POND2-302	Multiple users can not redeem the same intent in one transaction	MINOR	RESOLVED
POND2-303	Nonce helper prevents batch unlocks by a single intent	MINOR	RESOLVED
POND2-304	The account functions do not prevent excessive fragmentation	MINOR	RESOLVED
POND2-305	Near-full and small spends of Ada UTxOs can be troublesome	MINOR	PARTIALLY RESOLVED
POND2-306	Quantity is unchecked for `LabelOutRef` label	MINOR	RESOLVED
POND2-307	Unlock module vulnerable to cross-script double satisfaction	MINOR	ACKNOWLEDGED

Continued on next page

ID	TITLE	SEVERITY	STATUS
POND2-401	Staking credential registration validation will be required soon	INFORMATIONAL	ACKNOWLEDGED
POND2-402	Documentation imprecise about allowing only 1 intent token mint	INFORMATIONAL	RESOLVED
POND2-403	Pond unlock `ByOwner` contains additional fields	INFORMATIONAL	ACKNOWLEDGED
POND2-404	Nonce helper can check that it is registered just once	INFORMATIONAL	ACKNOWLEDGED
POND2-405	Typos and incorrect documentation	INFORMATIONAL	RESOLVED
POND2-406	Naming and dead code	INFORMATIONAL	PARTIALLY RESOLVED
POND2-407	Code style suggestions	INFORMATIONAL	PARTIALLY RESOLVED
POND2-408	Suggest documenting negative quantity impact	INFORMATIONAL	RESOLVED
POND2-409	Intents using nonce could be blocked if nonce value is found	INFORMATIONAL	ACKNOWLEDGED
POND2-410	Pond native validator potentially checks all pond versions	INFORMATIONAL	RESOLVED

POND2-001 Nonce helper script may be bypassed

Category	Vulnerable commit	Severity	Status
Design Issue	`4496c3eeb8`	CRITICAL	RESOLVED

Description

In the current and past Cardano eras including Conway, it's possible to register a script-based stake credential without supplying the corresponding script witness, thanks to the `stake_registration` certificate[1]. As a result, the nonce helper script that is designed to enforce one-time usage is never invoked if an attacker chooses that certificate type. This completely skips any validation in the nonce helper registration; namely it skips the validation that just a single associated intent is run.

Recommendation

I recommend using a different mechanism that guarantees a one-time usage. The simplest way would be to expect the `stake_reg_deleg_cert` certificate instead of a simple registration as that may or may not require the witness.

An alternative, more messy approach is to use a different method altogether, s.a. using requiring a spend of a specific UTxO. That UTxO might be spendable by anybody given a new nonce helper validator is run. The nonce helper validator might be a withdraw validator and the UTxO might be locked on the same script. Further, the validation might check that any Ada deposited into it is returned to the owner that deposited it.

Resolution

The issue was fixed in the commit `3264d02113`. Instead of pure registration certificate, the client now uses the `RegisterAndDelegateCredential`. This certificate type requires a script witness to be included in the transaction and thus fixes the issue.

[1] https://plutus.cardano.intersectmbo.org/docs/working-with-scripts/script-purposes#certifying

POND2-002 Nonce helper validator self-reference

Category	Vulnerable commit	Severity	Status
Design Issue	`3264d02113`	CRITICAL	RESOLVED

Description

The nonce helper ensures that an intent that requires it is executed only once in a pre-defined time window. To do that, the logic relies on the intent being executed in the transaction where the nonce's stake credential is registered. The nonce helper is actually a validator controlling this registration.

This issue suggests that the nonce helper logic might not work. The reason is technical — there is a circular dependency in its hash usage; so it's not possible to compute it properly and hence use it in a proper way. Specifically:

- The modules s.a. the `unlock_fungible_outputs_for_payment` module require the nonce hash to be included in the module arguments. They check that they're the same as the intent's nonce property.
- They check that a `Certificate` registering and delegating this script hash is included in the transaction. This means that the script is run in the transaction.
- The actual nonce helper script also runs a validation logic. It checks that there is exactly one intent being satisfied that mentions this nonce in the intent's `nonce` property.

However, the way that the nonce helper retrieves its nonce hash is to have it supplied as a validator argument. It compares the pond's intent nonce with this value supplied via the argument.

Validator arguments influence the final script as they are baked into it at the compilation time. They also change its hash. As such, there is a circular dependency of this hash whereas you can not supply the resulting hash of the script into it as that would change the hash of the script.

As a result, it is not possible to self-reference itself via an argument. If the argument is a credential different to the nonce helper scripts' credential, then there is no guarantee that the nonce logic is actually run.

Recommendation

I suggest removing the *`intent_nonce`* argument from the nonce helper validator. Instead, the validator can extract its credential from parsing the certificate's credential.

Resolution

The issue was fixed in the commit *`a36b8c0e7e`* .

POND2-003 Account helper does not prevent unauthorized drain of funds

Category	Vulnerable commit	Severity	Status
Logical Issue	*3264d02113*	**CRITICAL**	**RESOLVED**

Description

As mentioned in the documentation, the accounts helper should be checked for in any module modifying fungible balances. It assumes all intents are authorized, computes all quantities vs unlocked quantities, computes the change that should be returned and verifies that the change outputs exist.

However, the script trusts too much. It iterates only over redeemers. Both addresses, quantities and other intent data s.a. unlocked quantities are trusted. Input sanitation also does not really happen. Even though it is true that redeemers coming from actual pond validators are checked inside the pond validator, it is not true for potentially arbitrary data coming from redeemers of other scripts that might be maliciously included among the inputs in such a way that the redeemers could be parsed into the `PondRedeemer` data schema.

An attack unlocking all fungible balances using e.g. the defragment module would look as follows:

For every address and label combination that you want to unlock, include an input on a malicious non-pond script address. Use a redeemer that follows the `PondRedeemer` schema and match the address and label fields. Set the quantity field to e.g. 1 and a fake intent's `unlocked_quantity` to approximately the value you want to steal from the locked pond UTxO. This setup would resemble a valid setup whereas the user had multiple pond UTxOs and they try to use more intents which should all be authorized. It just so happens that the locked value and unlocked quantity are equal and so the accounts helper allows near-zero value change output.

Recommendation

It is dangerous to parse only redeemers and check their type conformity to `PondRedeemer`. It might be okay in some cases, but if you need to be able to trust some data or validation, you need to check that the input is actually locked at the proper

pond script address. In the accounts helper case, you need to check this. I also suggest checking all other modules/helpers for similar mistakes.

Resolution

The issue was fixed by the commit `6104a3db66`. A new required field `pond_script` is now included in any `Intent`. And it is checked in all modules that the input's payment credential is the same as the `pond_script` specified.

POND2-004 Address-based accounting lets an attacker steal funds

Category	Vulnerable commit	Severity	Status
Design Issue	`a36b8c0e7e`	CRITICAL	RESOLVED

Description

The *`accounts helper`* — and every module that relies on it (e.g. the *`defragment module`*) — assumes that the tuple *`(script address, label)`* uniquely identifies the owner of a Pond UTxO. In reality the true owner is most often stored in the *`owner`* datum field while the script address may contain no stake credential at all or a generic one shared by many users. Consequently two different users can lock outputs at the same *`Pond`* script address under the same label yet have different owners. When such outputs are spent together the helper merges them, computes a single change value and accepts any change output that matches the tag/quantity — without ever checking that its datum's *`owner`* matches all owners that were merged. An attack would look as follows:

- A victim has money locked at a pond address with e.g. the *`defragment module`* enabled.
- An attacker prepares or already has a tiny UTxO at the exact same address as the victim, including the staking credential; but with a different *`owner`* set in the *`datum`*. In the end, if the *`owner`* field is set, the staking credential doesn't really make a difference. If needed, multiple UTxOs can be created to satisfy the defragmentation's *`min_inputs`* requirement.
- The attacker builds a transaction that spends both UTxOs using a legitimate defragment intent (*`quantity_unlocked = 0`*), creates one change UTxO with the combined amount at the same address; but with the *`owner`* set in the *`datum`* to the attacker's credential.

The attacker now took control of the victim's funds. Even though the funds are still at the same address with potentially the victim's staking credential, it is the attacker that can withdraw the funds.

Recommendation

I suggest including the *owner* in the key that the *accounts helper* (and *defragment module*) uses for its bookkeeping, e.g. *change_key = (address, owner, label) ▷ builtin.serialise_data*. Moreover, I suggest reviewing all other helpers and modules for similar assumptions about address meaning owner uniqueness. Anyone can set the staking credential to anything.

Resolution

The issue was fixed by the commit *f7d86a9e1d*. The *owner* is now part of the accounting key. Also, the *owner* is now set to be derived only from the staking credential, it has been removed from the *datum*.

POND2-005 All funds can be stolen by pretending to be an owner of any UTxO

Category	Vulnerable commit	Severity	Status
Design Issue	`f7d86a9e1d`	CRITICAL	RESOLVED

Description

After a refactor, the owner of a particular UTxO is not checked against its staking credential inside the pond validator itself. Instead, it is delegated to be checked inside any module. Inside the validator, the owner supplied in the redeemer is trusted. It is important to realize that as such, it can be set arbitrarily.

The authorized intents' validation is verified against this supplied owner, though. Meaning, any non-intended module could actually be authorized by an attacker drafting a transaction setting himself as the owner of a UTxO of somebody else.

To sum it up, anybody can pretend to be the owner of somebody else's UTxO, authorize an intent of a module doing no validation at all, and the pond validator would unlock any UTxO for them.

Recommendation

I suggest checking that the owner supplied in the `PondRedeemer` is equal to the staking credential of the UTxO inside the pond validator.

Resolution

The issue was fixed by the commit `41e6c0a32b`. Even though the `owner` datum field is left in the code for optimization reasons, and it is not always checked inside the pond validator against the staking credential which is the source of truth of the ownership; it is checked against the redeemer inside the validator and it is checked against the true owner defined in the staking credential in all modules. In non-datum use cases, it is checked directly. This is enough to fix the issue as there is no incentive for an honest user to mis-set the `owner` field when creating a UTxO themselves and it is checked to be equal to the staking credential in cases where modules validate creation of such UTxOs on behalf of users.

POND2-006 Value can be stolen as anyone can sign intent authorization

Category	Vulnerable commit	Severity	Status
Logical Issue	`33106d36ae`	CRITICAL	RESOLVED

Description

It should be possible to spend pond UTxOs only if their owner signed off the intents and they are executed in the transaction. The signature verification is not complete in the off-chain case, though. It is verified that a valid signature is provided that authorizes the merkle tree hash where the intents are included. However, it is not checked that the `key` which is checked to have signed the message containing the root hash actually belongs to the UTxO's `owner`.

As a result, anyone can spend any UTxO by signing any intents themselves and providing **their** key and signature to authorize them.

Recommendation

I recommend adding a check checking that the digest of the `key` field is actually the `owner` credential.

Resolution

The issue was fixed by the commit `9a77954d5f` .

POND2-007 Pond native validator skips all validation after an unknown redeemer

Category	Vulnerable commit	Severity	Status
Logical Issue	`0954a600a6`	CRITICAL	RESOLVED

Description

After a big refactor in the pond native validator, the validation is not executed per instance of pond UTxO, but it's executed once for all pond input UTxOs at once in a single withdrawal validator instead. This validator iterates over the redeemers and inputs. When it finds a redeemer that is not a `PondRedeemer`, it is supposed to skip that redeemer, continue checking the next one, and so on. However, it ends the iteration logic altogether.

This might result in a completely skipped validation if e.g. the first redeemer is not a `PondRedeemer`. It is easy to achieve that for an attacker; it is enough to include any other lexicographically early script with a different redeemer schema in the transaction.

Practically, the skipped validation means that all value within all input pond UTxOs is unlocked and free to be taken.

Recommendation

I suggest making sure to recursively call the `validate_continuing_redeemers_and_inputs` function further if the redeemer is not recognized, as is done in the other modules that iterate redeemers and inputs in a similar way.

Resolution

The issue was fixed in the commit `47d7674a50` .

POND2-008 Service fee module stops processing after the first match

Category	Vulnerable commit	Severity	Status
Logical Issue	`47d7674a50`	CRITICAL	RESOLVED

Description

The `validate_continuing_redeemers_and_inputs` function in the module service fee validator fails to continue iterating through remaining redeemers after successfully processing a matching input/redeemer pair. When the function finds a matching input for the current redeemer's `outref`, it processes that pair but then immediately returns the accumulated results without continuing to the remaining redeemers.

This premature termination means that any redeemers appearing after a successfully processed redeemer will be completely ignored. This potentially includes all service fee redemptions. The validation that the associated redemptions are executed will be skipped.

An attacker could therefore collect all service fees from all users without executing anything for them. What's more, since the service fee module itself does not include the accounts-change logic, all label-matching UTxOs of all users that authorized any service fee module could be fully unlocked, regardless of the `unlocked_quantity` specified.

Recommendation

The function should continue iterating through all redeemers even after successfully processing a matching input/redeemer pair. After processing a match, it should recursively call itself with the remaining redeemers, inputs and updated accumulating variables.

Resolution

The issue was fixed by the commit `889c952c51` .

POND2-009 Service fee module observes intents from non-pond scripts

Category	Vulnerable commit	Severity	Status
Design Issue	`47d7674a50`	CRITICAL	RESOLVED

Description

The `module_service_fee.ak` validator has a critical vulnerability where it fails to verify that observed intents are actually included across valid pond scripts. When observing redemptions, the code simply adds the intents to the observed dictionary without verifying the specified `pond_script` matches the expected script hash.

This is somewhat understandable since the protocol wants to support multiple pond variants. However, it is important to observe only valid intents that are authorized by the owner and actually executed — which can only be guaranteed if the script address is an actual pond script.

Since it is not checked, an attacker can create dummy input UTxOs on a malicious always validating script, set the staking credential to the targeted owner and add all the required redemptions there. Since the script is not a pond script, it is neither checked that the redemptions are authorized nor executed. Hence, the attacker is free to set the `pond_script` redeemer variable to his malicious script.

Furthermore, the attacker would include all label-matching targeted user's actual pond UTxOs and include the service fee redemption only. The redemption is authorized and the validation would observe the redemptions on the invalid script as well. Since the service fee module itself does not contain the accounts-change logic, the attacker can steal all the contents from the target user's UTxOs without executing anything for them and without the limitations of the specified `unlocked_quantity`.

Recommendation

I suggest adding the `pond_script` inside both the required/observed lists. When a service fee redemption is detected, I suggest requiring same-script redemptions. Alternatively, you could make the service fee module's arguments a `List<Intent>` which would allow cross-script redemptions while being super clear about which intent should be observed on which script.

Resolution

The issue was fixed by the commit `a6b1e86e9f`. The arguments are now lists of tuples of owners, labels and the intents themselves that include the `pond_script` field.

POND2-101 Arbitrary label injection when a label is present in the datum

Category	Vulnerable commit	Severity	Status
Logical Issue	`f7d86a9e1d`	MAJOR	RESOLVED

Description

When a label is set in a pond datum, it should take precedence. However, when it is checked that the label from the redeemer corresponds with the UTxO, the validation is actually skipped and the label from the redeemer is trusted. This can confuse accounting logic of some modules or even make it look as-if there was a module authorized if a match is found with a previously used label and the same intent. That's because the owner's signature and the downstream modules will all be evaluated against this forged label, not against the real one stored in the output.

Recommendation

I suggest making sure that if a label is specified in the pond datum, it is checked to be equal to the label in the redeemer.

Resolution

The issue was fixed by the commit `c75334b397`. The label from the datum is now checked to be equal to the label in the redeemer.

POND2-102 Attacker can repeatedly DoS intent authorizations and keep Ada

Category	Vulnerable commit	Severity	Status
Logical Issue	`33106d36ae`	MAJOR	RESOLVED

Description

The scripts support two kinds of intent authorizations. This issue is about the on-chain one, whereas there is an intent token minted by the owner and put inside a UTxO at the intent script with the authorized merkle tree's root hash in the datum. Such UTxOs can be referenced to authorize pond UTxO spends.

However, the spending conditions of the script where the authorization is saved are not strict enough. They ensure that the intent token is not misplaced. However, they do not prevent a malicious party from simply unlocking anyone's intent UTxO, burning their intent token and unlocking some min Ada locked within.

Aside from a small financial motivation that is there, especially when done at scale, this effectively destroys the proof that the party authorized some intents until they either re-create the intent UTxO, which can be DoS-ed again; or provide an off-chain authorization.

Recommendation

I suggest adding a check that allows an intent UTxO spend if and only if the owner of the UTxO signs the transaction. The check is somewhat there, but the owner whose signature is checked is taken just from the redeemer and does not have to correspond to the intent token's asset name contained within the UTxO — the **real** owner of the UTxO.

Resolution

The issue was fixed in the commit `639776eb50`. A mint or burn of intent tokens is allowed only by an owner matching the token's asset name. That is enough to fix the issue.

POND2-103 Modules do not support indirect calls s.a. the *and* module's

Category	Vulnerable commit	Severity	Status
Design Issue	*f23c7278c6*	MAJOR	RESOLVED

Description

Modules tend to search for pond UTxO redemptions and check where they are directly referenced in the redemptions. If found, they include some validations. Sometimes, they include a transaction level validation s.a. verifying correct account changes for all redemptions, which are not exlusive to module-related UTxOs only.

As such, when modules s.a. the *and* module are used, the relevant modules that are called indirectly do not recognize their call. Additionally, there is currently no way of supplying module arguments to them for this indirect call.

In practice, this means that both these use cases mentioned in the *and* module documentation don't really work:

- The *and* module can not atomically combine multiple payments if those payments use a module s.a. the `module_unlock_fungible_outputs_for_payment` module. That's because arguments including the required payment details can not be supplied to the module and the module does not recognize its call from the *and* module.
- The defragmentation module does not require module arguments. However, it also doesn't work well when invoked indirectly as it also checks only direct redemptions mentioning the defragmentation module. As a result, even though the use case of specifying a positive `unlocked_quantity` with an *and* redemption including only the defragmentation as a means of paying a third party a small fee for the defragmentation service is valid and the fee can be taken, the defragmentation is not checked. The `inputs_dict` would be empty, so the defragmentation does not have to happen or can happen on any small number of inputs as the `min_inputs` check is not checked either.

The *and* module can work with modules that do not require module arguments and can recognize such indirect call, or don't need to recognize it.

Recommendation

For the *and* module to work as described, the modules need to somehow recognize an indirect call. The simplest but not really nice way is to have a version of the modules that are capable of this, they know about the *and* module's script hash and can find their mention in its arguments. Furthermore, I suggest adding an optional module arguments per each indirect call. Nicer solutions are certainly possible as well.

Resolution

The issue was fixed by the commit `47d7674a50` by phasing out the *and* module and the atomicity guarantees. A service fee module was added instead, that can be used to unlock a service fee if a list of redemptions are all executed in the same transaction. The redemptions need to be separately authorized, but that's not a problem as it explicitly does not guarantee the atomicity. However, only if they are all executed in the same transaction, the service fee can be unlocked. Furthermore, the arguments are now supported and checked. There are no indirect calls anymore, all redemptions including the service fee redemption and the underlying modules are included in the redemptions separately.

POND2-201 Signed intents can not be revoked once issued

Category	Vulnerable commit	Severity	Status
Design Issue	*4496c3eeb8*	**MEDIUM**	**RESOLVED**

Description

When using an off-chain signature for intent authorization (instead of a reference input), there is no on-chain mechanism to ”unsign” or revoke that signature. Once the signature is created, any party possessing it or simply observing it from the blockchain can keep submitting transactions referencing the same signed root hash and proof, and the contract code has no method to invalidate it. Assuming there’s no other helper module included such as the nonce helper.

Note that this is a generic design issue. The modules that were part of the codebase at the time did include timestamps in their module-specific arguments.

Recommendation

Consider adding an expiration timestamp to all off-chain signed data.

Resolution

The issue was fixed in the commit *dc95bbd829* — a *valid_until* timestamp was added to all off-chain authorized messages.

POND2-202 Defragmentation of native token balances unlocks min Ada within

Category	Vulnerable commit	Severity	Status
Design Issue	*f23c7278c6*	**MEDIUM**	**ACKNOWLEDGED**

Description

When a party defragments another party's tokens either directly using the defragmentation module or indirectly via using the ambiguity of the number of change outputs enforced in the accounts helper in e.g. the unlock for payment module, the party can unlock and keep all leftover min Ada. Say it's a constant 2 Ada per UTxO, if a party merges 50 UTxOs with a native token label, they can keep 48 Ada for themselves.

Recommendation

I suggest controlling the Ada amounts contained within the UTxOs in case the account functions are involved even for non-Ada labels as different rebalances can happen and Ada can be claimed.

Resolution

The issue was acknowledged as intended. The described impact is present and is considered a way of funding the batcher service as the transaction fees are meant to be paid by the protocol.

POND2-301 Nonce helper overwrites a valid match if it almost finds another

Category	Vulnerable commit	Severity	Status
Code Issue	`4496c3eeb8`	**MINOR**	**RESOLVED**

Description

In the nonce helper registration validation, once a redeemer successfully matches the `output_reference` and sets the running total to 1, a subsequent redeemer that meets other criteria but does not use the nonce can reset the total to 0 (see line #54). This would result in the transaction not validating as zero matches were found instead of one.

Recommendation

I suggest correcting the inner loop's initial value in the nonce helper registration validation, setting it to `total` instead of 0, such that it does not reset the counter.

Resolution

The issue was fixed in the commit `c25ddc009e` .

POND2-302 Multiple users can not redeem the same intent in one transaction

Category	Vulnerable commit	Severity	Status
Logical Issue	`a36b8c0e7e`	**MINOR**	**RESOLVED**

Description

Inside the `accounts helper`, the `all_redemptions` set uses the following hash as its key: `blake2b_256((label, intent))`. If two different owners spend UTxOs that reference the same intent under the same label, the second redemption is silently ignored.

This results in the `quantity_spent` being under-counted, so the helper believes more change should be returned. Mostly, it just means that it is not possible to use the same intents by other owners in a single transaction. However, it might also interfere with a module's validation if it assumes a quantity is unlocked when it isn't.

Recommendation

I suggest including the `owner` and `address` in the uniqueness key, in addition to the `label` and `intent` that are currently used, before inserting into the `all_redemptions` set.

Resolution

The issue was fixed by the commit `41e6c0a32b`. The `owner` or the `address`, that includes the `owner`, is now part of the uniqueness key.

POND2-303 Nonce helper prevents batch unlocks by a single intent

Category	Vulnerable commit	Severity	Status
Design Issue	*33106d36ae*	**MINOR**	**RESOLVED**

Description

The nonce helper validator enforces a strict requirement that exactly one redemption can mention it. This prevents batch unlocks where a single intent with a nonce is used to unlock multiple complete UTxOs in the same transaction. While modules such as the *module_unlock_fungible_outputs_for_payment* validator have built-in support for tracking which intents have been validated using an *intent_hash* dictionary, this capability cannot be fully utilized due to the nonce constraint.

Recommendation

If batch unlocks are desired for intents using the nonce helper, consider modifying the nonce helper to allow multiple redemptions that share the same nonce when those redemptions are part of the same single logical operation.

Resolution

The issue was fixed by the commit *e1fc0c8418* .

POND2-304 The account functions do not prevent excessive fragmentation

Category	Vulnerable commit	Severity	Status
Design Issue	`f23c7278c6`	**MINOR**	**RESOLVED**

Description

When a module such as the `module_unlock_fungible_outputs_for_payment` is used, it is checked exactly how much is spent. However, it is not checked into how many UTxOs the change is split.

For example, it could be possible for a single pond UTxO containing 100 Ada to be used for a 10 Ada payment and the rest could be split across 45 UTxOs. All of them would be small enough to be used for any purpose aside from a defragmentation.

Recommendation

I suggest limiting into how many UTxOs the change is divided.

Resolution

The issue was fixed by the commit `7c6d925a2c` by requiring a single change output per an address/label combination.

POND2-305 Near-full and small spends of Ada UTxOs can be troublesome

Category	Vulnerable commit	Severity	Status
Logical Issue	*f23c7278c6*	**MINOR**	**PARTIALLY RESOLVED**

Description

Cardano ledger enforces a minimum amount of Ada on every UTxO. Imagine a pond UTxO containing a lot of Ada that is all or almost all supposed to be paid somewhere utilizing the unlock for payment module. The change would need to be detected as well. Now, since the UTxO is spent entirely or almost entirely, the change is 0 lovelace or other small amount smaller than the required min Ada amount. It needs to be present in the transaction outputs, but the ledger does not allow that. As such, those intents can not be used.

On the other hand, if the payment is too small, it might be impossible to create the payment UTxO.

Recommendation

I suggest loosening the strict equality checks, especially for Ada amounts. It is okay to detect more Ada. Also, I suggest allowing full spends by being okay with not seeing a change output if the amount contained within it should be zero.

Resolution

The issue was partially fixed by the commit *d6e6123225* . The change output is not required to be present if the amount is 0. However, the change is still compared exactly to the required change and so small UTxOs and near-full spends might not be possible and would require further user deposit. This is acknowledged by the client.

POND2-306 Quantity is unchecked for *LabelOutRef* label

Category	Vulnerable commit	Severity	Status
Logical Issue	*f23c7278c6*	**MINOR**	**RESOLVED**

Description

The quantity field that is part of the pond redeemer is not checked at all when the label type is *LabelOutRef*. As such, it can be set to anything by the party that builds the transaction. For example, if a user signed an intent that allows a payment with the UTxO, the overall required change could be influenced by setting the quantity to different values.

Recommendation

I suggest validating the quantity field even for the output reference use case.

Resolution

The issue was fixed in the commit *1c32920c32* by requiring zero quantity for the output reference use case — requiring the usage of different label types for fungible balances.

POND2-307 Unlock module vulnerable to cross-script double satisfaction

Category	Vulnerable commit	Severity	Status
Design Issue	*f23c7278c6*	**MINOR**	**ACKNOWLEDGED**

Description

The unlock for payment module expects a certain payment output to be created. There might be multiple different scripts expecting such a payment at that address, with that value and potentially that datum. It is important to realize that this is potentially vulnerable to a cross-script double satisfaction that way.

Recommendation

Users are encouraged to not reuse addresses and include unique datums for the required payment in the unlock for payment module. The scripts could also prevent this by limiting script inputs in the transaction to only the known own script inputs to fully eliminate the cross-script double satisfaction possibility.

Resolution

The issue was acknowledged and is not present if users always require payments to smart accounts and set unique IDs as tags, or set some kind of unique identifiers in case of external payments — this is how it is intended as confirmed by the client. The issue then remains only if users diverge from this, reuse tags or require payments to addresses other than smart accounts or external protocols supporting unique identifiers tagging the UTxOs. To stress, also direct payments to user addresses should include such unique identifiers.

POND2-401 Staking credential registration validation will be required soon

Category	Vulnerable commit	Severity	Status
Design Issue	*4496c3eeb8*	**INFORMATIONAL**	**ACKNOWLEDGED**

Description

Some scripts rely solely on the withdrawal path for their validation logic and do not include checks for stake registration. Even though the current and prior Cardano eras including Conway allow script-based staking credentials to be registered without a script witness via the `stake_registration` certificate, this certificate type will be deprecated in the next era. All stake credential registrations will require including the script witness. And, it is necessary for the stake to be registered so that the withdrawal can happen.

Recommendation

Consider adding a simple, even if an always-true registration validation logic to all staking validators to be future-proof.

Resolution

The issue was acknowledged. There is no impact. The client is trying to keep the code as lean as possible for now to save on script size. Since they'll be launching soon, they can register these without issue. But after the `stake_credential` is deprecated, any newly launched modules will include the always-succeeds registration.

POND2-402 Documentation imprecise about allowing only 1 intent token mint

Category	Vulnerable commit	Severity	Status
Documentation	*33106d36ae*	**INFORMATIONAL**	**RESOLVED**

Description

The documentation in the *all_outputs_paid_to_script* function mentions that there can be only a single intent token minted inside a transaction. However, it is only checked that there is a maximum of one such token per output UTxO, not per transaction. There can be more outputs with a single token in that transaction.

Recommendation

I suggest keeping the logic but rewording the comment to reflect the above fact.

Resolution

The issue was fixed in the commit *639776eb50* .

POND2-403 Pond unlock *ByOwner* contains additional fields

Category	Vulnerable commit	Severity	Status
Documentation	*33106d36ae*	**INFORMATIONAL**	**ACKNOWLEDGED**

Description

The pond validator *ByOwner* redeemer contains two additional fields, *key* and *signature*. They are not used in the validator even though their use case is described in the *types.ak* file.

Recommendation

I suggest removing the fields and the related documentation.

Resolution

The issue was acknowledged by the client. The fields, even if not used now, are kept in the validator to have the same schema for other pond validator variants.

POND2-404 Nonce helper can check that it is registered just once

Category	Vulnerable commit	Severity	Status
Code Issue	`f23c7278c6`	**INFORMATIONAL**	**ACKNOWLEDGED**

Description

Currently, a single nonce registration in a specified time window is already guaranteed by a nonce having an `intent_valid_until` field and allowing a deregistration and thus re-registration just after that date. However, that relies a bit on a clever orchestration of the valid until dates between the nonce helper and the intent using it.

Recommendation

Similar to de-registration checking that it is after the time window, I suggest checking that it is inside the time window within the nonce registration validation. That guarantees that it is registered just once within the nonce script.

Resolution

The issue was acknowledged. There is no impact as long as the timestamps are set correctly.

POND2-405 Typos and incorrect documentation

Category	Vulnerable commit	Severity	Status
Code Style	*f23c7278c6*	**INFORMATIONAL**	**RESOLVED**

Description

Across the code, there are a number of typos including: “lable”, “primarly”, “verfication”, “qauntity”, must be repaid “expect”, “ouputs”, through “nativate” transaction signing, “varian”.

Also, there’s a number of imprecise or misplaced documentation:

- “first determine if any inputs are unlocked for defragmentation” — The code it refers to only checks whether the particular validated input is unlocked for defragmentation.
- “we now ensure the actual output value holds what is expected” — A misplaced comment, it’s actually above the address check, not the value check.

Recommendation

I suggest correcting the typos and moving or correcting the documentation in all cases mentioned.

Resolution

The issue was fixed by the commit *5613f39373* .

POND2-406 Naming and dead code

Category	Vulnerable commit	Severity	Status
Code Style	*f23c7278c6*	**INFORMATIONAL**	**PARTIALLY RESOLVED**

Description

There's a few of places where the naming could be better:

- The *cip8_sig_structure_from_root* function's *root* argument is actually the whole message, not only the root hash.
- There is a mention of an "order book token" inside the pond validator which likely refers to the intent auth token.

There's also one unused function: *must_find*.

Recommendation

I suggest removing the unused function for leaner code and renaming the highlighted terms to not cause confusion.

Resolution

The issue was fixed by the commit *7c6d925a2c* . The unused function *must_find* was left in the code, though — as it is used across the other not-yet-audited modules.

POND2-407 Code style suggestions

Category	Vulnerable commit	Severity	Status
Code Style	*f23c7278c6*	**INFORMATIONAL**	**PARTIALLY RESOLVED**

Description

There is a number of code style suggestions that came to mind when reading the codebase:

- The *utils.quantity_of* function could make use of a standard Aiken library function doing the same, if the cost savings are not significant. Similarly, you could use standard library functions instead of implementing the *find_input_by_outref_or_fail* function.
- Similarly, if there are not significant savings, you could use branched validators based on the script context instead of catching them all via *else(_)* and branching manually based on the parsed script context.
- In the *must_be_before*, you handle all cases including the non-finite bounds. However, it can be simplified similar to the *must_be_after* function where you could require finite bounds.
- There are multiple occassions of shadowing a variable by introducing a new variable later on in the code which is named the same. Examples include variables *module* inside the *and* module, *time* in the *must_be_before* function.

Further, there are number of code duplications including the standardized mandatory module checks and the iterating accounts-inputs-like logic that could be extracted into a commonly re-used function which could be more readable and safer at the same time.

Recommendation

I suggest implementing the improvements mentioned in the description to enhance code readability and maintainability.

Resolution

The custom functions are kept as they perform better. Shadowing was resolved by the commit *5613f39373* . The code duplications are not addressed.

POND2-408 Suggest documenting negative quantity impact

Category	Vulnerable commit	Severity	Status
Documentation	`f23c7278c6`	**INFORMATIONAL**	**RESOLVED**

Description

The `quantity` field in the pond datum is somewhat trusted. Any computation needs to check that it is not negative as any other party's UTxO can be created by anyone and the field can easily end up being negative in those. Such quantities are currently discarded in the accounts code but any module needs to make sure that it doesn't trust negative quantities.

Recommendation

I suggest adding a comment explaining the danger on top of the `quantity` field in the pond datum.

Resolution

The importance of checking the quantity inside of modules was elaborated on in the commit `7c6d925a2c` .

POND2-409 Intents using nonce could be blocked if nonce value is found

Category	Vulnerable commit	Severity	Status
Design Issue	`e1fc0c8418`	**INFORMATIONAL**	**ACKNOWLEDGED**

Description

After some refactors, it was decided to not check the exact pond script presence inside the nonce helper to save some execution costs as multiple nonce helpers might be included in a single batch transaction.

This means, that a third party might register a nonce credential without the intent being executed, provided they have means of finding out the exact nonce credential. Once registered, the intent will be blocked as the nonce can not be registered multiple times.

In the initial phases of the project, the nonce values will not be public until used or until the launch of Decentralized Order Routing Aggregator (*DORA*).

Recommendation

As this is an intentional tradeoff, the recommendation is to just be clear to users about this and reconsider once *DORA* is launched as it might cause some headaches afterwards.

POND2-410 Pond native validator potentially checks all pond versions

Category	Vulnerable commit	Severity	Status
Design Issue	`0954a600a6`	**INFORMATIONAL**	**RESOLVED**

Description

After a big refactor of the pond validations, the newly introduced pond native withdrawal validator checks that the validation holds for all inputs that are referenced by a `PondRedeemer`. That might mean that it checks it for multiple different versions of the pond validator, not only for the `pond_native.ak` version. As such, it might not be possible to include multiple different pond versions in a single transaction.

Recommendation

If transactions consisting of multiple pond versions are intended, consider validating only pond native inputs in the `pond_native.ak` validation. Technically, it could be easier if the `pond.ak` spend validator and `pond_native.ak` withdrawal validator are combined into a single validator so they have the same hash and can easily find whether the other one is run/present in the transaction.

Resolution

The issue was fixed in the commit `f29b67998b`. An alternative approach was discussed and implemented — a version tag was added to the `PondRedeemer` that can be checked by the `pond_native.ak` validator to be precise and skipped in the `pond_native.ak` validator if it is not the native version.

A Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the agreement between Invariant0 LLC (Invariant0) and Pond Labs LLC (Client) (the Agreement), or the scope of services, and terms and conditions provided to the Client in connection with the Agreement, and shall be used only subject to and to the extent permitted by such terms and conditions. This report may not be transmitted, disclosed, referred to, modified by, or relied upon by any person for any purposes without Invariant0's prior written consent.

This report is not, nor should be considered, an endorsement, approval or disapproval of any particular project, team, code, technology, asset or anything else. This report is not, nor should be considered, an indication of the economics or value of any technology, product or asset created by any team or project that contracts Invariant0 to perform a smart contract assessment. This report does not provide any warranty or guarantee regarding the quality or nature of the technology analysed, nor does it provide any indication of the technology's proprietors, business, business model or legal compliance.

To the fullest extent permitted by law, Invariant0 disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. This report is provided on an as-is, where-is, and as-available basis. Invariant0 does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by Client or any third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services, assets and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Invariant0 will not be a party to or in any way be responsible for monitoring any transaction between you and client and/or any third-party providers of products or services.

This report should not be used in any way by anyone to make decisions around investment or involvement with any particular project, services or assets, especially not to make decisions to buy or sell any assets or products. This report provides general information and is not tailored to anyone's specific situation, its content, access, and/or usage thereof, including any associated services or materials, shall not

be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or other advice.

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Invariant0 prepared this report as an informational exercise documenting the due diligence involved in the course of development of the Client's smart contract only, and THIS REPORT MAKES NO CLAIMS OR GUARANTEES CONCERNING THE SMART CONTRACT'S OPERATION ON DEPLOYMENT OR POST-DEPLOYMENT. This report provides no opinion or guarantee on the security of the code, smart contracts, project, the related assets or anything else at the time of deployment or post deployment. Smart contracts can be invoked by anyone on the internet and as such carry substantial risk. INVARIANT0 HAS NO DUTY TO MONITOR CLIENT'S OPERATION OF THE PROJECT AND UPDATE THE REPORT ACCORDINGLY.

THE INFORMATION CONTAINED IN THIS REPORT MAY NOT BE COMPLETE NOR INCLUSIVE OF ALL VULNERABILITIES. This report is not comprehensive in scope, it excludes a number of components critical to the correct operation of this system. You agree that your access to and/or use of, including but not limited to, any associated services, products, protocols, platforms, content, assets, and materials will be at your sole risk. On its own, it cannot be considered a sufficient assessment of the correctness of the code or any technology. This report represents an extensive assessing process intending to help Client increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology, however blockchain technology and cryptographic assets present a high level of ongoing risk, including but not limited to unknown risks and flaws.

While Invariant0 has conducted an analysis to the best of its ability, it is Invariant0's recommendation to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring and/or other auditing and monitoring in line with the industry best practice. The possibility of human error in the manual review process is highly real, and Invariant0 recommends seeking multiple independent opinions on any claims which impact any functioning of the code, project, smart contracts, systems, technology or involvement of any funds or assets. INVARIANT0'S POSITION IS THAT EACH COMPANY AND INDIVIDUAL ARE RESPONSIBLE FOR THEIR OWN DUE DILIGENCE AND CONTINUOUS SECURITY.

B Audited files

The files and their hashes reflect the final state at commit *bf21576ffd7cca45f4e59ec0c9460ca63d361394* after all the fixes have been implemented.

SHA256 hash	Filename
76473...a79b6	lib/pondora/accounts.ak
91293...7df3b	lib/pondora/types.ak
d014c...265b4	lib/pondora/utils.ak
20775...9112b	validators/intents_native.ak
351a8...28a42	validators/modules/helper_nonce.ak
ac3c6...1a76e	validators/modules/module_defragment.ak
a83e3...48892	validators/modules/module_dependencies.ak
4ab4a...21081	validators/modules/module_unlock_fungible_outputs_for_payment.ak
7b550...974eb	validators/pond_native.ak
20443...f7826	validators/pond.ak

Please note that we did not audit Aiken itself or the external *merkle-patricia-forestry* library used, which were assumed to function correctly.

C Methodology

At Invariant0, our agile methodology for performing security audits consists of several key phases:

1. Design reviews form the initial stage of our audits. The goal of the design review is to find larger issues which result in large changes to the code fast.
2. During the deep code audit, we verify the correctness of the given code and scrutinize it for potential vulnerabilities. We also verify the client's fixes for all discovered vulnerabilities. We provide our clients with status reports on a continuous basis providing them a clear up-to-date status of all the issues found so far.
3. We conclude the audit by handing over a final audit report which contains descriptions and resolutions for all the identified vulnerabilities as well as an executive summary featuring project overview, collaboration notes and a brief summary of findings found.

Throughout our entire audit process, we report issues as soon as they are found and verified. We communicate with the client for the duration of the whole audit. During our audits, we check several key properties of the code:

- Vulnerabilities in the code.
- Adherence of the code to the documented business logic.
- Potential issues in the design that are not vulnerabilities.
- Code quality and potential for improvement.

During our manual audits, we focus on a wide range of attacks; including but not limited to double satisfaction, theft of funds, violation of business requirements, token uniqueness attacks, faking timestamps, locking funds indefinitely, denial of service, unauthorized minting, and loss of staking rewards.

D Issue classification

Severity levels

The following table explains the different severities. In general, the darker the color, the more severe the issue.

Severity	Impact
CRITICAL	Theft of user funds, permanent freezing of funds, protocol insolvency, etc.
MAJOR	Theft of unclaimed yield, permanent freezing of unclaimed yield, temporary freezing of funds, etc.
MEDIUM	Smart contract unable to operate, partial theft of funds/yield, etc.
MINOR	Contract fails to deliver promised returns, but does not lose user funds.
INFORMATIONAL	Best practices, code style, readability, documentation, etc.

Resolution status

The following table explains the different resolution statuses. In general, the darker the color, the more attention the issue requires.

Resolution status	Description
RESOLVED	Fix applied.
PARTIALLY RESOLVED	Fix applied partially.
ACKNOWLEDGED	Acknowledged by the project to be fixed later or out of scope.
PENDING	Still waiting for a fix or an official response.

Categories of issues

The following table explains the different categories of issues.

Category	Description
Design Issue	High-level issues in the design. Often large in scope, requiring changes to the design or massive code changes to fix.
Logical Issue	Medium-sized issues, often in between the design and the implementation. The changes required in the design should be small-scaled (e.g. clarifying details), but they can affect the code significantly.
Code Issue	Small in size, fixable solely through the implementation. This category covers all sorts of bugs, deviations from specification, etc.
Code Style	Parts of the code that work properly but are possible sources of later issues (e.g. inconsistent naming, dead code).
Documentation	Small issues that relate to any part of the documentation (design specification, code documentation, or other audited documents). This category does not cover faulty design.
Optimization	Ideas on how to increase performance or decrease costs.

E Report revisions

This appendix contains the changelog of this report. Please note that the versions of the reports used here do not correspond with the audited application versions.

Report v1.0: Main audit

Revision date: 2025-06-05
Final commit: *bf21576ffd7cca45f4e59ec0c9460ca63d361394*
Report link: Full report link

We conducted the audit of the base Pondora native contracts and a few initial modules including the payment unlock module, the defragmentation module, the service fee module, and the nonce helper. To see the files audited, see Audited files.

F About us

Invariant0, the fresh new face of Vacuumlabs Auditing.

- You probably already know our team as former Vacuumlabs Auditing.
- We helped create WingRiders, currently the second largest decentralized exchange on Cardano (based on TVL).
- We prevented a massive $180M vulnerability in production.
- To date, we found and reported over 373+ vulnerabilities, delivered 21+ reports and thus protected over $150M in assets for our clients and their users.
- Our clients include *FluidTokens, Liqwid, WingRiders, Iagon*, and many more.

Our auditing team is chosen from the best.

- Talent from esteemed Cardano projects: WingRiders and NuFi.
- Rich experience across Google, traditional finance, trading and ethical hacking.
- Award-winning programmers from ACM ICPC, TopCoder and International Olympiad in Informatics.
- Driven by passion for program correctness, security, game theory and the blockchain technology.

Contact us:
info@invariant0.com