vacuumlabs

Iagon Delegation v3

Solo Audit Report v1

December 20, 2024

Contents

Revision table

Report version	Report name	Date	Report URL
1.0	Main audit	2024-12-20	Full report link

1 Executive summary

THIS REPORT DOES NOT PROVIDE ANY WARRANTY OF QUALITY OR SECURITY OF THE AUDITED CODE and should be understood as a best efforts opinion of Vacuumlabs produced upon reviewing the materials provided to Vacuumlabs. Vacuumlabs can only comment on the issues it discovers and Vacuumlabs does not guarantee discovering all the relevant issues. Vacuumlabs also disclaims all warranties or guarantees in relation to the report to the maximum extent permitted by the applicable law. This report is also subject to the full disclaimer in the appendix of this document, which you should read before reading the report.

Project overview

Iagon delegation v3 smart contracts are a small Cardano on-chain part of the Iagon protocol which is a decentralized storage protocol working mostly outside of the blockchain. The delegation smart contracts enable Iagon's *IAG* token holders to stake their tokens and delegate them to registered nodes. It further allows for the delegated tokens to be sold while remaining locked in the smart contracts. The rewards distribution for the delegation is not part of the smart contract system, is determined in the off-chain and distributed separately. The on-chain consists of the following contracts:

- *Reference script*. There is a single valid UTxO on the reference script. It holds a one-shot *Genesis token* minted to bootstrap the protocol. The UTxO holds the configuration of the protocol, including the *IAG* token policy id, the contracts' hashes and the very important *operator* hot key. The data is updatable by the *reference keepers*' multi-signature scheme. They are also noted in the datum. The data here is crucial for the security and it allows updates including updating the contracts. All other scripts read the data from this UTxO when running their validation.
- *Node script*. A UTxO on the node script is a registered node if it is created the proper way — it holds the node token meaning its data has been verified and contains the *pledge* in *IAG*. Some verification happens on-chain, some is purely off-chain. All interactions with the node script happen if and only if both the node owners called *node keepers* and the *operator* agree with such an action. A node can be active or retiring. If the node goes from active to retiring, it is assigned a time by the operator after which it can be withdrawn. The withdrawal process needs be signed off by the operator again.

- *Delegation script*. A UTxO on the delegation script is a registered delegation of an *IAG* stake contained within if it is created the proper way. It also holds the delegation token and its data is verified. Among other things, it specifies which node the *IAG* is delegated to. A delegation can be in the delegated state, it can be retiring or it can be on sale for a specified price per *IAG* unit. It supports multiple actions on it such as an update of the information including a possible redelegation to a different active node, splitting the delegation into multiple smaller delegations with possibly different data, withdrawing from the delegation if it's in the retiring state and the retiring time assigned by the operator during an update already expired, and selling off part of the delegated *IAG* for the specified price when it is in the on sale state. All actions need to be signed off by the ***operator*** and all except for the trade need to be signed off by the ***delegation keeper*** as well.
- *Order script*. A UTxO on the order script is a registered order — a wish to purchase an amount of delegated *IAG* from a specific delegation that is on sale. It holds an order token validating that the info has been verified and the operator has signed its creation off. Furthermore, it holds the Ada purchase price as well as any fees for the trade. The fees are not controlled nor accounted for by the on-chain at all. Once the order is executed, the purchase price is transferred to the seller's address directly, his delegated stake is decreased and a new delegation with the purchased *IAG* for the buyer is created.

The contracts provide a way to store tokens on the Cardano blockchain. They are ***centralized*** and in a lot of ways rely on the ***operator*** key which is a backend hot key stored in Iagon's servers. The issues highlight a lot of ways the operator might misuse this power if it is compromised or dishonest. It is important to note that the operator key can be updated by the reference keepers in the reference datum. That can prevent further damage but can not mitigate damage already done if any.

Audit overview

This was a solo audit employing just a single senior Vacuumlabs smart contract auditor. I started the audit at commit `b52d4ac33064f943758a1cabca05a23a7ad20147` and it lasted from 28 Nov 2024 to 20 Dec 2024. The timeframe is inclusive of periods in which I was awaiting the implementation of fixes by the client. We interacted mostly on Discord and gave feedback in GitHub pull requests. The team fixed some issues to my satisfaction; however, there was 1 major and a lot of minor issues that were either acknowledged or resolved only partially. In most cases, it was argumented that the client relies on the operator being honest and hence do not need to fix those issues as the issues highlight

cases that are not covered in the smart contract logic but depend on the operator key being either compromised, the operator being malicious or there being inadequate back-end controls or possible bugs in place.

The scope of the audit was limited to the smart contract files only. I did not review any tests as part of this audit, even though a large set of tests was a part of the codebase. I appreciate the test coverage. However, there were two critical issues that are not part of the issues listed in this report, that were introduced in the fixes reviewed in Github that broke the happy path and that were not caught by the tests.

I performed 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. See more about the methodology in Methodology.

The commit `10c2140d19b77ff6d2c04be8f16cb947fddcaceb` 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.

Summary of findings

During the audit, we found and reported: 0 critical, 4 major, 0 medium, 13 minor, and 9 informational findings. A lot of findings were acknowledged or resolved only partially, including:

- **ID3-102** — Node and delegation can not withdraw without the operator. The protocol is centralized, but it would be great to have some kind of an emergency recovery for the tokens locked within in case the project goes offline. The issue was partially resolved by having the keepers in the broader ecosystem, so that the liveness of the protocol is not dependent on Iagon alone. However, the recovery mechanism was not implemented. The withdrawal of the value locked is dependent on the project being alive.
- **ID3-301** — Operator as the sole guarantor of unique id computation. The protocol uses token names as ids of the various UTxOs. It is not on-chain guaranteed that those are unique, though. It is up to the operator to enforce this. Most of the on-chain logic depends on this property.

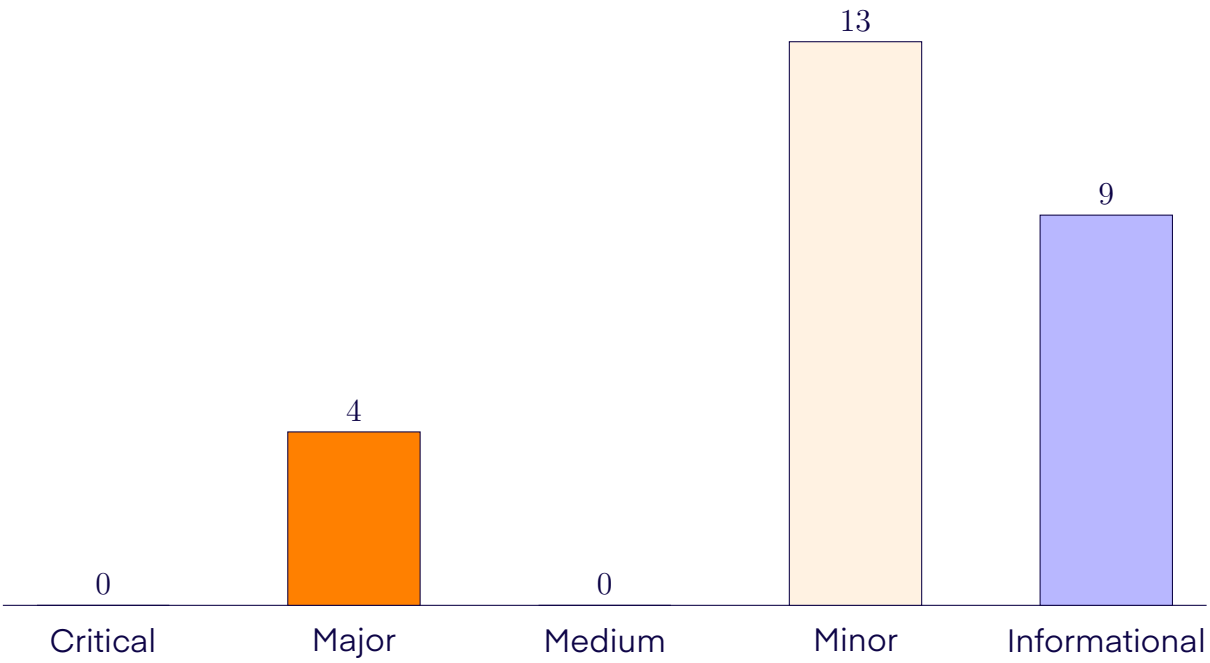
- **ID3-304** and **ID3-402** — Delayed order signature is not necessary and Genesis token name is unnecessarily complex. There is a lot of reduntant validation that is of the codebase and these present two such examples of complex on-chain verifications that are not useful and should be simplified. They were kept, though.
- **ID3-307**, **ID3-310**, **ID3-312** and **ID3-313** — Numerous possible attack vectors that rely on the operator preventing them. However, if the key is compromised or malicious or the code contains bugs, these can happen.
- **ID3-406** — Naming. The code style was improved only partially.

I will reiterate that since the delegation contracts are centralized, there were no critical issues aside from those 2 unreported that were found during the code review process in Github which caused unfeasibility of certain happy path scenarios.

Most of the actual issues reported here assume a malicious role or an error in either the operator or the reference keepers and explain what could go wrong in those cases. It ranged from stealing Ada from all orders described in the issue ID3-104, through blocking all scripts and value described in issues ID3-101 and ID3-103, to all flavors of less subtle attack vectors possible with the operator key — most of which were acknowledged.

The other set of vulnerabilities was related to the code style. It ranged from an unnecessarily complicated code such as issues ID3-304 and ID3-402, which mostly just make it less readable to excessive comments reported in the issue ID3-405, grammar issues, typos and copy paste errors reported in ID3-403 and ID3-404 and a lot of improvements that could be done in variable naming that is part of the issue ID3-406. The code style findings are not meant to be cover all instances of possible improvements as there certainly are a lot of ways to improve the final code further.

2 Severity overview



Findings

ID	TITLE	SEVERITY	STATUS
ID3-101	Reference keepers might block all scripts and value	MAJOR	RESOLVED
ID3-102	Node and delegation can not withdraw without the operator	MAJOR	PARTIALLY RESOLVED
ID3-103	Reference keepers might block all scripts and value II.	MAJOR	RESOLVED
ID3-104	Reference keepers can steal Ada contained in orders	MAJOR	RESOLVED
ID3-301	Operator as the sole guarantor of unique id computation	MINOR	ACKNOWLEDGED

Continued on next page

ID	TITLE	SEVERITY	STATUS
ID3-302	Order tokens can be freely minted	MINOR	RESOLVED
ID3-303	Operator can mint any node, order and delegation tokens	MINOR	RESOLVED
ID3-304	Delayed order signature is not necessary	MINOR	PARTIALLY RESOLVED
ID3-305	Reference keepers might lock themselves out of the protocol	MINOR	RESOLVED
ID3-306	Node can be moved to any address during an update	MINOR	RESOLVED
ID3-307	Delegation seller may receive slightly smaller compensation	MINOR	ACKNOWLEDGED
ID3-308	Delegation can be moved to any address during an update	MINOR	RESOLVED
ID3-309	Split delegation UTxOs are mostly unchecked	MINOR	RESOLVED
ID3-310	The payment output could be used to satisfy a foreign script	MINOR	ACKNOWLEDGED
ID3-311	The payment output could contain dust tokens	MINOR	RESOLVED
ID3-312	The payment output can be staked to any credential	MINOR	ACKNOWLEDGED
ID3-313	Orders do not protect against delegation sale price change	MINOR	ACKNOWLEDGED
ID3-401	`plutus.json` does not correspond to the code	INFORMATIONAL	RESOLVED
ID3-402	Genesis token name is unnecessarily complex	INFORMATIONAL	ACKNOWLEDGED
ID3-403	Grammar issues, typos, semantics	INFORMATIONAL	RESOLVED
ID3-404	Copy paste errors	INFORMATIONAL	RESOLVED

Continued on next page

ID	TITLE	SEVERITY	STATUS
ID3-405	Excessive comments	INFORMATIONAL	RESOLVED
ID3-406	Naming	INFORMATIONAL	PARTIALLY RESOLVED
ID3-407	Delegation can be delegated to a non-existent or inactive node	INFORMATIONAL	RESOLVED
ID3-408	Some fields are not used nor verified on-chain	INFORMATIONAL	RESOLVED
ID3-409	Multiple order trades not possible in a single transaction	INFORMATIONAL	RESOLVED

ID3-101 Reference keepers might block all scripts and value

Category	Vulnerable commit	Severity	Status
Code Issue	`b52d4ac330`	MAJOR	RESOLVED

Description

The reference script keepers, multiple parties responsible for the protocol, are able to block all the scripts and value locked. That includes the node script and all the pledged stake, order and purchase amounts, and delegation and delegated IAG tokens.

They can do this by moving the genesis token from the reference script into a different script as the `that_address`, a new reference UTxO's address is not checked to correspond to its actual address — it's only checked that a good address is put in its datum. Keepers might construct a good datum during a reference UTxO update but misplace it to a malicious script's address. In a follow-up transaction, they are free to do as they want; including sending the token into a UTxO on a public key credential with no datum. All the scripts rely on the reference UTxO's existence and need to be able to parse the `ReferenceDatum` at all times to function. This effectively blocks all validation and keeps the funds locked until and if they decide to return the token and restore the protocol.

By exploiting the same attack surface, the keepers might also update reference datum fields that are supposed to be untouchable, s.a. the IAG token asset class with unforeseen consequences.

The severity is not critical because the keepers are a trusted multi-sig party and their agreement is needed.

Recommendation

I recommend checking that the address of the output reference UTxO is still on the reference script so that the genesis token can not be misplaced.

Resolution

The issue was fixed in the pull request number 9.

Client comment

We assume that the keepers are good honest actors as this will just be lagon but to make sure nothing can happen by accident the following lines are added to `reference.ak`.

```
// that address on that output must be this address
let that_address: Address = that_output.address
// that address defined in the datum must be this address
let that_datum_address: Address =
address.from_script(that_datum.contracts.reference)

// this is added to the end statement

// this address is that address { we'll reference solves the genesis
    problem }
this_address == that_address?,
// that address is the datum address
that_address == that_datum_address?,
```

Now, `that_output` has `that_address` which is now equal to `this_address` being spent from and defined in the datum as `that_datum_address`.

ID3-102 Node and delegation can not withdraw without the operator

Category	Vulnerable commit	Severity	Status
Design Issue	`b52d4ac330`	MAJOR	PARTIALLY RESOLVED

Description

Both nodes and delegations could contain significant value. Currently, if the operator is e.g. non-responsive and the reference keepers do not update them, which could happen if the project goes offline, the funds locked are inaccessible. The operator's signature is required for a withdrawal and there is no way to go around it.

Recommendation

I recommend brainstorming and implementing an emergency-situation scenario for ultimately accessing the locked funds if things ever go south.

Resolution

It was clarified that the reference keepers that are able to update the operator are not only part of the Iagon team. It is a multi-signature scheme 2/3 of Iagon, Cardano Foundation and Eternl. As such, this is a bit broader and partially solves the liveness aspect. There's been no update on the smart contract side, though. The operator needs to be operational and the nodes and delegations can not be withdrawn without the operator.

Client comment

We shall assume that Iagon lives forever for the purposes of the bootstrap model. If Iagon fails, we shall assume we are taking the token with it. We need the exiting of the nodes and delegations to be operator controlled so we can do safe forks into new versions of the protocol as that is planned for the future. This is the bootstrap phase that will lead into the federated phase where this signing condition is relaxed. And our backup plan for the operator key getting exposed / stolen / hacked / etc is having the keepers update the reference datum and change the operator key hash to something else.

ID3-103 Reference keepers might block all scripts and value II.

Category	Vulnerable commit	Severity	Status
Logical Issue	`b52d4ac330`	MAJOR	RESOLVED

Description

Similar to the previous issue ID3-101, the reference keepers might break the reference UTxO and thus make any following actions of other scripts unfeasible as they all require the reference UTxO in the reference inputs. They can achieve this by making the reference keepers' list too long. They could make it so long and the UTxO so big that it would still pass their update reference datum validation, but the validation of scripts that are way more complex (such as the delegation or node scripts) would hit either the transaction size or execution limits.

Recommendation

I recommend limiting the number of keepers to a reasonable small enough number.

Resolution

The issue was fixed in the pull request number 13.

ID3-104 Reference keepers can steal Ada contained in orders

Category	Vulnerable commit	Severity	Status
Design Issue	`b52d4ac330`	MAJOR	RESOLVED

Description

Reference keepers are able to update the reference datum. Among other things, it contains contract script hashes. If they update the delegation contract's hash to a malicious contract, then the orders are unprotected. The process order validation flow delegates a lot of validations to the delegation contract's validator and it takes its hash from the reference datum. If the reference changes, it delegates the validation to a dummy validation. That means that the value contained within can be stolen by the keepers freely.

Recommendation

I suggest rethinking whether the `contracts` field needs to be updatable. The easiest fix is to forbid updating it. The other way to fix this issue is to e.g. put the delegation script hash into the order datum at the time of its creation, so it remains fixed for the duration of the order existence.

Resolution

The issue was fixed in the pull request number 14 by having the delegation script hash put into the order datum and requiring it to be in-sync with the reference datum's delegation hash. In other words, if the script is updated, the order needs to be reclaimed by its keeper and can not be processed. Reference keepers hence can not unlock the Ada contained within.

ID3-301 Operator as the sole guarantor of unique id computation

Category	Vulnerable commit	Severity	Status
Design Issue	b52d4ac330	MINOR	ACKNOWLEDGED

Description

The token names of the protocol tokens such as order tokens, node tokens and delegation tokens are supposed to be unique. A lot of on-chain validations rely on this uniqueness. However, there is no on-chain mechanism guaranteeing it. The only way to guarantee the unique ids is to rely on the operator to not make a mistake, compute the id correctly and not re-use a token name.

Recommendation

I suggest implementing unique token name generation in the on-chain validation.

Resolution

The issue was acknowledged with the following comment:

Client comment

This is by design. We have old node names that must be constant and ported over on-chain that follow a db standard that the previous team created. We decided that in future versions of the protocol there will be a token name oracle that will just spit out unique names when any mint action occurs. In this version, it is entirely on the operator just as it was from the old versions of the codebase from the previous team.

ID3-302 Order tokens can be freely minted

Category	Vulnerable commit	Severity	Status
Logical Issue	`b52d4ac330`	MINOR	RESOLVED

Description

Order tokens are supposed to be unique in the token name. There is also a validation on its mint, checking that an order is properly created. All this can be avoided if the tokens are minted in the following way:

1. An attacker creates an order in a valid way.
2. The attacker removes his order. Only his signature is required for this.
3. In the removing transaction, he needs to burn his order token. However, he can also mint any other order tokens of arbitrary token name and amounts as the minting policy runs just once, the `Burn` redeemer is used and it is checked only that *a* token is actually burned.

He can use the tokens in any way, skipping the mint validation and confusing the back-end services.

Recommendation

I recommend tightening the burning validation logic, checking that only burning of tokens is allowed and that there is no side mint happening.

Resolution

The fix was implemented in the pull request number 9. There's been one more introduced critical issue highlighted and fixed before merging, during the code review process.

Client comment

This is the only action that is not regulated by the operator as leaving the order contract does not change the delegation or node stake status / amounts. Instead of checking if just the token is burned, a more controlled burn occurs.

```
//
// anyone can burn it
```

```
fn burn(token) -> {
  // get the mint value here
  let mint_value_list: List<(PolicyId, AssetName, Int)> =
    mint
      |> assets.flatten()
  // find the input that holds the order token
  expect Some(input) =
    search.for_input_by_token(inputs, currency_symbol, token)
  // destruct the delegation datum and into do some datum
   validation
  expect OrderDatum { assigned_delegation_token, .. } =
    when input.output.datum is {
      InlineDatum(inbound_datum) -> inbound_datum
      // just fail here if the datum isn't inline so we don't want
   that to happen
      _ -> fail @"Inline_Datum_Only_For_Order_Datum"
    }
  // The order is either being processed with one other delegation
   plus or
  // it is being removed. We can tell this by the length of the
   minted value.
  // This restriction makes a single order per transaction but
   limits any
  // nonsensical minting they may occur.
  //
  // There are two entries to the mint value, a zero to
   consuming
  or {
    // Use either a mint and burn in same order
    or {
      // use an or as the ordering may not be constant
      mint_value_list == [
        (currency_symbol, 0n, -1),
        (delegation, assigned_delegation_token, 1),
      ],
      mint_value_list == [
        (delegation, assigned_delegation_token, 1),
        (currency_symbol, 0n, -1),
```

```
        },
    },
    // we mint just a token
    mint_value_list == [{currency_symbol, token, -1}],
  }
}
```

The order burn redeemer got an overhaul. The order token must be burned by itself or it must be burned along with the mint of the assigned delegation token. This should account for the remove endpoint and the process endpoint.

ID3-303 Operator can mint any node, order and delegation tokens

Category	Vulnerable commit	Severity	Status
Logical Issue	`b52d4ac330`	MINOR	RESOLVED

Description

The operator's signature is necessary for any of those mints. As already mentioned, it is not on-chain checked that the token name is unique. Furthermore, it is also possible to avoid the mint validation and mint any number of any node, order and delegation tokens to any address. The operator can mint any side tokens during a normal mint operation. This is because it is only checked that there is a mint happening, not that the mint is exclusive to the token name `tkn` specified in the redeemer:

```
1 assets.quantity_of(mint, currency_symbol, tkn) == 1
```

As the minting policy validation runs just once for all token names minted, the mint field could contain any number of additional records under the same currency symbol and different token names. By doing that, he could e.g. confuse the backend systems.

Recommendation

I suggest validating all the minted token names or restricting the number of such tokens under the currency symbol — in all mint validations.

Resolution

The issue was resolved in the pull requests number 19 and 20.

Client comment

We assume the operator is a good honest actor as this will just be Iagon. We assume that all token names generated will be unique. This feature is by design. We want the operator to have the ability to do many mints inside of a single transaction. We will create comprehensive backend tests to ensure this functionality does not confuse our own backend system.

ID3-304 Delayed order signature is not necessary

Category	Vulnerable commit	Severity	Status
Design Issue	`b52d4ac330`	MINOR	PARTIALLY RESOLVED

Description

When an order is created, the order keeper needs to sign the transaction. This is verified in the minting policy of the order token. There's also a message containing a subset of order datum fields which is verified to be signed by the keeper and it's kept in the datum. When an order is processed, the (delayed) signature of this message is again verified both in the order spend validation and in the new delegation minting policy validation.

Overall, there's quite some validation to support this. However, it is totally fine to remove it. Here's why: As the order keeper (the new delegation keeper) signs the transaction where he creates the order, he signs off all the fields in the order datum already. As the message propogated on is just a subset of the fields and since it is checked that the order token is part of the order input, it is enough to just rely on the order datum.

Recommendation

I suggest to remove unnecessary code handling the delayed signatures as it is not necessary and gives the impression that it is important. Furthermore, if it were true that only that delayed signature was guaranteeing the keeper's agreement, such signature would be dependent on the uniqueness of the ids as the message could have been replayed anytime and would have to contain more fields. Transaction signatures are immune to this.

Resolution

The code of delayed signatures is mostly kept, some was removed in the pull request number 12; but it was clarified to not be important. It serves more as a second factor, as metadata for the backend.

ID3-305 Reference keepers might lock themselves out of the protocol

Category	Vulnerable commit	Severity	Status
Logical Issue	`b52d4ac330`	MINOR	RESOLVED

Description

Reference keepers are able to update core protocol data contained within the reference datum. There is also the list of the keepers part of that datum, which works as a multi-signature scheme with a threshold of signatures needed. It is checked that enough of them sign any transaction modifying the data. Among other things, they are able to modify the list of the reference keepers itself. It is not verified, though, that the new list is feasible. Hence they might lock themselves out of the protocol if they are not careful.

Recommendation

I suggest checking not only that the old keepers achieve the multi-sig threshold on any reference datum update, but instead that the potentially new keepers agree on that change as well.

Resolution

The issue was fixed in the pull request number 13.

ID3-306 Node can be moved to any address during an update

Category	Vulnerable commit	Severity	Status
Logical Issue	d76cae86e7	MINOR	RESOLVED

Description

During a node update flow, the address of the new node UTxO is unchecked. Hence, the node value including the pledge and the node token can be moved to any address. For example, if the operator is malicious and the node keepers do not verify the transaction properly, relying on the smart contract, the value could be moved to a malicious address controlled purely by the operator, for him to steal it later.

The severity is minor, though, as both the operator and the node keepers need to sign this transaction off.

Recommendation

I suggest checking that the address of the new node UTxO is unchanged.

Resolution

The issue was resolved in the pull request number 16.

ID3-307 Delegation seller may receive slightly smaller compensation

Category	Vulnerable commit	Severity	Status
Logical Issue	d76cae86e7	MINOR	ACKNOWLEDGED

Description

The delegation seller sets a price, called the `unit_price`, per `1` whole unit of `IAG` that has 6 decimals in the delegation datum when he's putting the delegation on sale. The price computation in the smart contract is governed by the following formula:

```
[ if output.value !> assets.lovelace_of ] >= purchased_amount *
    unit_price / 1_000_000?,
```

Since the `unit_price` is in lovelace per `1` whole `IAG`, and the `purchased_amount` is in the number of the smallest `IAG` units, it needs to be divided by the decimals for the units to match. The division found in the implementation floors the result down. As a result, the seller is exposed and potentially loses on this. The discrepancy is up to `1` lovelace only, though.

For example, if an attacker purchased `1` smallest `IAG` unit and the `unit_price` would be almost 1 ADA, this formula would allow him to pay zero. In reality, though, the off-chain code would likely not even process this trade as there's not enough min Ada to create both the new delegation and the payment outputs. It is still possible to pay one lovelace less, though.

Recommendation

I suggest removing rounding from the formula by changing it to the following:

```
[ if output.value !> assets.lovelace_of ] * 1_000_000 >=
    purchased_amount * unit_price?,
```

Client comment

We are fine with this. This seems like a non issue as well. The blockchain does not allow this type of trade to ever occur as a UTxO can not have 0 lovelace. Switching the formula to account for this edge case seems like a nothing burger.

My comment

As described in the description, yes, this specific 0 lovelace error is impossible and just showcases the error with the formula. 1 lovelace difference is achievable for bigger amounts as well.

ID3-308 Delegation can be moved to any address during an update

Category	Vulnerable commit	Severity	Status
Logical Issue	`6cfd1e4b0c`	MINOR	RESOLVED

Description

Similar to the issue ID3-306, the new delegation's address is unchecked in the update delegation flow. Hence, the validation path is more or less equivalent to just requiring the signatures of both the operator and the delegation keeper. This seems unwanted.

Recommendation

I suggest checking that the delegation's address is unchanged.

Resolution

The issue was fixed in the pull request number 15.

ID3-309 Split delegation UTxOs are mostly unchecked

Category	Vulnerable commit	Severity	Status
Design Issue	`6cfd1e4b0c`	MINOR	RESOLVED

Description

In the split delegation flow, an old delegation UTxO is split into potentially numerous smaller delegation UTxOs. These newly created delegations are mostly unchecked. Their address can be arbitrary; they do not have to be on the delegation script. Their datum is totally unchecked as well. Their only validation is on their value — they need to hold a delegation token and the `IAG` stake according to the split mapping provided.

Note: I suppose the error was introduced because the minting policy's workings were misunderstood. A minting policy is run just once for any mixture of burns and mints in a single transaction. In this case, since there is also a burn happening, it is enough to call the minting policy once, with the `Burn(this_datum.delegation_token)` redeemer. That skips all validations that normally occur for newly created delegations.

Recommendation

I suggest making sure to thoroughly check all the new delegation UTxOs, for example by updating the minting policy to run the validations for each token name minted.

Resolution

The issue was fixed in the pull request number 15.

ID3-310 The payment output could be used to satisfy a foreign script

Category	Vulnerable commit	Severity	Status
Design Issue	1119a2aa5c	MINOR	ACKNOWLEDGED

Description

When a trade happens, it is checked that a payment output of enough Ada amount is created at the delegation seller's address. However, it is not checked that no other foreign script is present in the transaction. Hence, it is theoretically possible for the operator to misuse this, include a foreign script s.a. an NFT marketplace's listing that also requires a payment to the same address and double-satisfy both scripts by paying just once. This assumes that no adequate protection is implemented in the foreign script.

Recommendation

I recommend checking that no additional script input is included in a delegation trade transaction.

Client comment

This assumes the operator is doing that. We are assuming the operator will not be doing that since we are in control of the operator. Also, it would restrict us from doing something fun like issuing NFTs for purchases or some other idea that could come up later that would basically be a foreign script working off the double satisfaction on purpose.

ID3-311 The payment output could contain dust tokens

Category	Vulnerable commit	Severity	Status
Logical Issue	1119a2aa5c	MINOR	RESOLVED

Description

It is not checked that the payment output created as part of the delegation trade flow is free of dust tokens. By putting arbitrary zero value unburnable tokens inside the UTxO, some of the Ada that is checked to be contained within the UTxO is locked inside it — it serves as min Ada which is bigger because of the dust tokens. This means that the seller effectively receives less usable value out of the trade.

Recommendation

I suggest checking that the payment output contains only Ada.

Resolution

The issue was fixed in the pull request number 17.

ID3-312 The payment output can be staked to any credential

Category	Vulnerable commit	Severity	Status
Design Issue	1119a2aa5c	MINOR	ACKNOWLEDGED

Description

The staking credential of the payment output that is created as part of the delegation trade flow is unchecked. That means, that the operator is free to choose any staking credential for the payment output, including his own. Furthermore, some Cardano wallets may not show UTxOs with arbitrary staking credentials to their owner.

Recommendation

I suggest including the full address of the delegation keeper inside the delegation datum and using that to check the payment output's full address.

Client comment

This one is kind of on the wallets for not showing spendable utxos. Users need to use better wallets. If I remember correctly, only Nami does this.

My comment

I suggest documenting this edge case in the project's FAQ so that users can resolve it more easily even if they use Nami. Client acknowledged that they'll let their support team know.

ID3-313 Orders do not protect against delegation sale price change

Category	Vulnerable commit	Severity	Status
Design Issue	1119a2aa5c	MINOR	ACKNOWLEDGED

Description

If a delegation sale price changes, existing orders are still valid. If the price increases, there's likely not enough Ada contained in them, so the trade will likely not be executed. However, if the price drops, the difference between the amount that is contained within the order UTxO and the required payment can be taken by the operator.

Recommendation

I suggest reimbursing the buyer in case the price drops after the order is created.

Client comment

The operator will not allow price changes if orders are in the contract for a trade. We control updating and process orders so we can regulate this.

ID3-401 `plutus.json` does not correspond to the code

Category	Vulnerable commit	Severity	Status
Code Issue	`b52d4ac330`	INFORMATIONAL	RESOLVED

Description

The code is newer than it's compiled artefact.

Recommendation

I suggest re-compiling the code every time there are changes, so it's kept up to date.

Resolution

The issue was fixed in the pull request number 9. The parameters are already applied so care needs to be taken to re-compile it with the correct mainnet configuration once it will be known. This script is out of scope of the audit.

ID3-402 Genesis token name is unnecessarily complex

Category	Vulnerable commit	Severity	Status
Design Issue	`b52d4ac330`	INFORMATIONAL	ACKNOWLEDGED

Description

The genesis token is a one-shot token marking the reference datum. It is parametrized by a UTxO that needs to be spent in order to be minted, hence the one-shot property. As it is parametrized by it, the policy id of the genesis token already reflects the uniqueness. The token name is computed to capture a similar kind of uniqueness. However, it is not strictly necessary.

Recommendation

I recommend simplifying the code by removing the genesis token name computation and related validations. You can e.g. freely fix the token name to a constant token name.

Resolution

The issue was acknowledged with the following comment:

Client comment

We will leave it as is for now. We may decide at a later time to make a custom name for fun like `Iagon Delegation` or just shorten the name itself by taking the first `N` bytes or something.

My comment

As mentioned in the description, it is totally up to the team what token name they choose. They can keep the existing logic as there's no security impact in that. The code is unnecessarily bloated but that's all.

ID3-403 Grammar issues, typos, semantics

Category	Vulnerable commit	Severity	Status
Documentation	`b52d4ac330`	INFORMATIONAL	RESOLVED

Description

A non-exhaustive list of grammar and similar issues:

- In the `lib/search` file, there's an instance of "faster then"; should be "faster than".
- In the `validator/node` file, there's "...type cast this here to validated"; should be "...to validate...".
- In the `lib/reference` file, there's "new keepers must have ... and be logically."; the second part is not grammatically correct and should be elaborated on.
- In the `validator/delegation` file, there's "node being delegated too"; should be "...delegated to".
- In the `validator/reference` file, there's:
  - "This attempts to makes sure..."; should be "to make sure".
  - "It is suppose to be..."; should be "supposed to...".
  - "updated via the mutlsig"; there's a missing "i", should be "multisig".
- In the `lib/order` and `validator/order` files, there's a mention of a "min utxo". You probably mean the min Ada (utxo). Min utxo is not the same thing.

Recommendation

I suggest checking the code again and fixing the errors as described in the description.

Resolution

The issue was addressed in the pull request number 9.

ID3-404 Copy paste errors

Category	Vulnerable commit	Severity	Status
Code Style	`b52d4ac330`	INFORMATIONAL	RESOLVED

Description

In the order validator, there are instances of copy paste errors from a different similar validation. A non-exhaustive list mostly in the mint and burn parts of the validator:

- "mint a delegation token"; should be "mint an order token".
- "find the output that holds the delegation token"; should be "...order token".
- "destruct the delegation datum"; should be "...the order datum".

Recommendation

I suggest checking the code again and fixing the errors as described in the description.

Resolution

The issue was resolved in the pull request number 9.

ID3-405 Excessive comments

Category	Vulnerable commit	Severity	Status
Documentation	`b52d4ac330`	INFORMATIONAL	RESOLVED

Description

The code is documented and that is a good thing. However, almost every line has a comment. It is not always helpful as the comment sometimes just repeats what the code just a line below already says or could say. Let's look at three simple examples:

- A function parameter named `inputs` is commented a line above like this: "this will be the reference inputs". You can rename the variable to `reference_inputs` and remove the comment.
- A datum field called `token` has a comment "this is IAG". You can rename the field to `iag_token` and remove the comment.
- A comment "operator must sign it" above a well named boolean variable `has_valid_operator_signature` does not say anything new and can be freely removed without sacrificing readability.

Recommendation

Even though it is counterintuitive, I suggest making use of comments only in cases when there is an additional non-trivial logic, background or reasoning to be explained. Instead of repeating the code in comments, strive for a readable code with good and clear naming and elaborate on the non-trivial parts in the comments.

Resolution

The issue was addressed in the pull request number 9.

ID3-406 Naming

Category	Vulnerable commit	Severity	Status
Code Style	`b52d4ac330`	INFORMATIONAL	PARTIALLY RESOLVED

Description

Across the codebase, there's a number of instances of not self-explanatory naming. A non-exhaustive list includes the following highlighted examples:

- The `price` variable is not actually the total price, it's more of a `unit_price` instead.
- The `amount` variable part of the `Trade` redeemer is not clear in the validation path, as it mixes with other amounts in the computation. Renaming to e.g. `purchased_amount` makes it clearer.
- The `node_token` field that is part of the order datum. It is unclear whether it should match the seller's `node_token` or the newly purchased delegation's `node_token`. Furthermore, the `node_token` is used as a form of an id of the node. Maybe let's use `delegate_to_node_token` instead?
- The `assigned_delegation_token` variable; as all token names are assigned and there's two delegations in the transaction where this is used, it is important to say that this is the newly created (as part of the purchase order flow) delegation token in my opinion, so e.g. `new_delegation_token` could be more descriptive.
- The `do_multisig` function could be renamed to e.g. `count_signatories` to better describe what it does.

Recommendation

I suggest checking the code again and renaming the variables to more descriptive names.

Resolution

Most of the points were resolved in the pull request number 9. The last point was not resolved as the Aiken stdlib-style naming notation wanted to be kept. The `node_token` field was not renamed as well.

ID3-407 Delegation can be delegated to a non-existent or inactive node

Category	Vulnerable commit	Severity	Status
Design Issue	`6cfd1e4b0c`	INFORMATIONAL	RESOLVED

Description

Unlike in the delegation creation flow which requires the node to be active and to be put in the reference inputs, it is not required if the `node_token` is updated in the update delegation flow or in the split delegation flow. As a result, the delegation can be updated to be delegated to a non-existent or inactive node. Both the operator and the delegation keeper need to sign off this transaction.

Recommendation

I suggest rethinking whether the checks are required and unifying the code validations — either loosening the requirement in the delegation token mint, or adding the checks in the update delegation and split delegation flows.

Resolution

It is verified that the node exists and is active in both the delegation update and the delegation split flows. The former was implemented in the pull request number 14 whereas the latter in the pull request number 15.

ID3-408 Some fields are not used nor verified on-chain

Category	Vulnerable commit	Severity	Status
Documentation	1119a2aa5c	INFORMATIONAL	RESOLVED

Description

The fields such as `node_size`, `min_delegation`, `max_delegation`, `margin_pct`, `delegation_size`, `additional`, `locked_until` are practically not used on-chain. More or less, they can store any data and it is up to the operator to ensure that they are valid.

Recommendation

I suggest commenting on the fields and clearly stating that they are not used on-chain and that the operator is responsible for setting them correctly.

Resolution

The comment was improved in the pull request number 18.

ID3-409 Multiple order trades not possible in a single transaction

Category	Vulnerable commit	Severity	Status
Documentation	1119a2aa5c	INFORMATIONAL	RESOLVED

Description

There is the following comment in the order validator: "This is a one-to-one trade and is meant to be parallelized with orders to other delegations and tx chained with other orders using this delegation." However, this is no longer possible as the minting policy allows exactly one burn in a transaction.

Recommendation

I suggest updating the comment to reflect the reality — a single order execution in a transaction.

Resolution

The comment was improved in the pull request number 18.

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 VacuumLabs Bohemia s.r.o. (Vacuumlabs) and Juno Development OÜ (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 Vacuumlabs'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 Vacuumlabs 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, Vacuumlabs 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. Vacuumlabs 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 Vacuumlabs 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. Vacuumlabs 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. VACUUMLABS 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 Vacuumlabs has conducted an analysis to the best of its ability, it is Vacuumlabs'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 Vacuumlabs 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. VACUUMLABS'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 `10c2140d19b77ff6d2c04be8f16cb947fddcaceb` after all the fixes have been implemented.

SHA256 hash	Filename
e7066...98857	lib/types/common.ak
e0ccc...660f4	lib/types/delegation.ak
12866...eb325	lib/types/node.ak
c3775...3c701	lib/types/order.ak
bcb1d...3738f	lib/types/reference.ak
61aa2...6bf54	lib/util/search.ak
33a88...94d66	lib/util/verify.ak
43dfc...faee4	lib/validate/backup.ak
c2b1e...68216	lib/validate/mint_delegation.ak
9d24b...40f91	validators/delegation.ak
1be8c...2e5bc	validators/genesis.ak
cdaf5...b5e8e	validators/node.ak
e7fcb...78757	validators/order.ak

Continued on next page

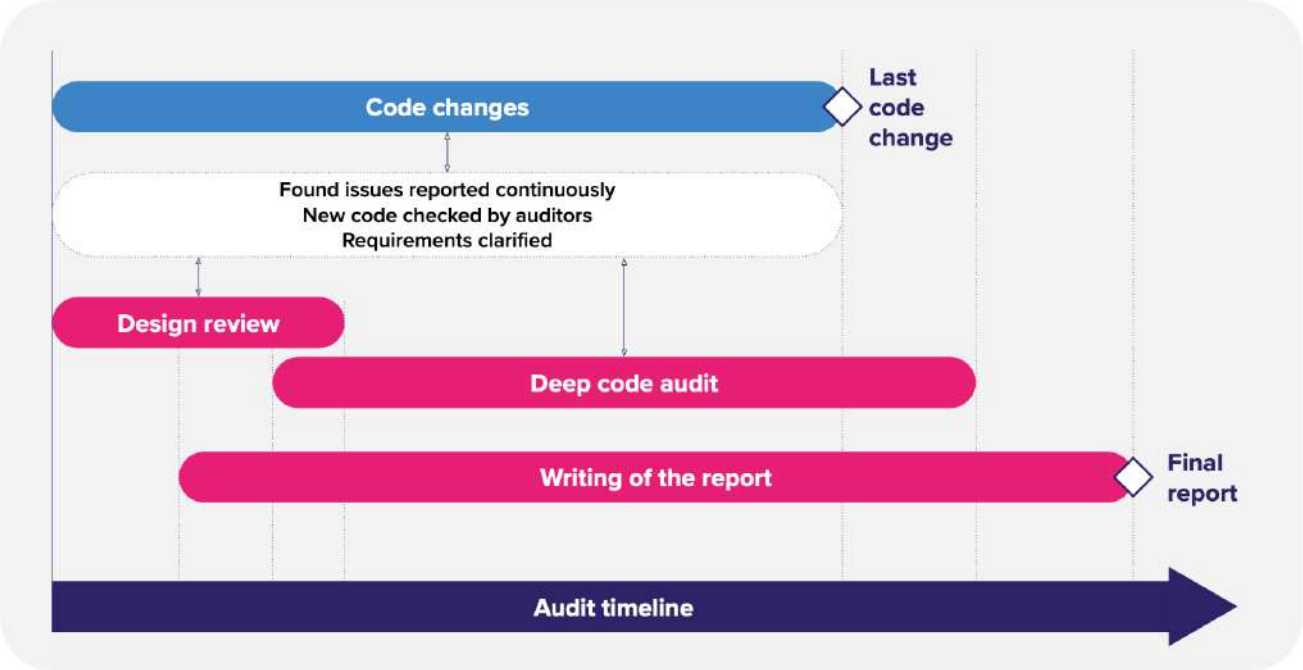
SHA256 hash	Filename
`48bda...0804d`	`validators/reference.ak`

Please note that I did not audit other files that are also present at the final commit.

C Methodology

Vacuumlabs' 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.



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

During our manual audits, we focus on several types of attacks, including but not limited to:

1. Double satisfaction
2. Theft of funds
3. Violation of business requirements
4. Token uniqueness attacks
5. Faking timestamps
6. Locking funds indefinitely
7. Denial of service
8. Unauthorized minting
9. Loss of staking rewards

D Issue classification

Severity levels

The following table explains the different severities.

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.

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.

v1.0: Main audit

Revision date: 2024-12-20
Final commit: `10c2140d19b77ff6d2c04be8f16cb947fddcaceb`

We conducted the audit of the main application. To see the files audited, see Audited Files.

Full report for this revision can be found at url.

F About us

Vacuumlabs has been building crypto projects since the early days.

- We helped create WingRiders, currently the second largest decentralized exchange on Cardano (based on TVL).
- We are behind the popular AdaLite wallet. It was later improved into a multichain wallet NuFi.
- We built the Cardano applications for the hardware wallets Ledger and Trezor.
- We built the first version of the cutting-edge decentralized NFT marketplace Jam On Bread on Cardano with truly unique features and superior speed of both the interface and transactions.

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:
audit@vacuumlabs.com