



Agora, Agora-Pro

Audit Report, v1

December 1, 2022

Contents

Disclaimer	1
1 Executive summary	3
Project overview	3
Glossary	3
Methodology	4
2 Severity overview	7
AGO-001: Stake state token can be taken away	10
AGO-002: Acting on behalf of delegatee role + Unlocking delegated stakes	11
AGO-003: Fake proposal can be created and GAT minted without any voting happening	12
AGO-004: Multiple GATs can be minted into fake scripts	13
AGO-005: Stake lock can be removed without retracting votes .	14
AGO-101: Delegatee can steal delegated inputs	15
AGO-102: Stake ST minting policy does not check LockedBy: Stake can be used to downvote without voting	16
AGO-103: pdestroy doesn't check outputs: Stake owner is able to subtract votes even if they did not vote	17
AGO-104: Attacker can fail any voted-on/locked proposal	18
AGO-105: Stake ST token name not checked: Stake can be used to downvote without voting	19
AGO-106: Stake can retract all votes in its cooldown period . . .	20
AGO-201: GATs are equal in the potential damage they can cause	21
AGO-202: getStakeDatum is staking credential sensitive	22
AGO-203: Stakes can be frozen effectively forever by the multisig entity	23
AGO-204: Governor can be DoSed by creating Proposals without passing min GT limit	24
AGO-301: Ambiguity in destroying multiple stakes	25
AGO-302: Some strict inequalities should not be strict	26
AGO-303: tooLate in code can also mean early: An attacker can fast track a proposal into Finished state	27

AGO-304: Other minor inconsistencies	28
AGO-305: Other naming suggestions	29
AGO-306: Proposal can be DoSed by using <code>UnLockStake</code> with no input stakes	30
AGO-307: Proposal can be DoSed by using <code>UnLockStake</code> with relevant cosigners' stakes	31
AGO-308: Proposal can be DoSed by voting and unlocking re- peatedly	32
AGO-309: Incorrect token references across the code	33
AGO-401: Staking credential is undefined	34
AGO-402: <code>PUnLock</code> : Ambiguous proposal redeemer name . . .	35
AGO-403: Delegatee can not vote with delegated and own stakes in one transaction	36

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 Liqwid 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 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 hyper-linked 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.**

1 Executive summary

Project overview

Agora is a Cardano native on-chain governance module written in Plutarch. It provides the necessary building blocks for any on-chain decentralized application to adopt and extend itself into a DAO-governed application. It consists of an open-source module (Agora) and a proprietary extension (Agora-Pro).

The module builds on the underlying project's governance tokens (GTs). All users owning GTs can deposit them into stakes. With stakes, they can create proposals, vote for effects (not) to take place, deposit more GTs and destroy stakes while retrieving the deposited tokens. Users can also choose to delegate the voting power to a trusted delegatee. There are multiple thresholds and time restrictions in place to prevent abuse.

An effect can alter the state or mechanics of either the governance module or the underlying system by taking hold of a governance authority token (GAT). This is the token that ultimately gives the effect the credibility to do so. As such, the minting process is controlled in such a way that a GAT can be minted only into an effect validator that controls the following usage (often including the burning of the GAT). The effect validator essentially encodes the wanted effect. The application validators need to be aware of GATs' existence and be able to yield the validation to such tokens and their own validation.

The extension builds on this and tags the GATs with an authority policy (AuthCheck) that is able to complement the effect validator's role (not only) in case the GAT's ultimate effect is intended to be processed in parallel. Furthermore, it adds the possibility to freeze stake withdrawals for a limited time. The Agora module is usable on its own or can be combined with the outlined extensions.

The effect validator's hash along with the datum and an optional AuthCheck tag (described below) is what is voted over in proposals. It is up to the team and community to make due diligence and vote only for safe, ideally audited effects. A vulnerability in an effect can have unforeseen consequences.

Moreover, the security of Agora governance module relies on the proper instantiation of configuration parameters such as the voting thresholds, time constraints, etc. It is up for a specific project making use of Agora to make sure the configuration is sound given i.e. the properties, supply and distribution of GTs, etc.

Glossary

- **GT:** Governance token. Tokens that allow users to vote in the governance system.

- **ST:** State (thread) token. Tokens identifying valid UTxOs that went through proper validations. There are stake STs, proposal STs, the governor ST, the stake freeze ST, etc. Taking hold of these often means breaking invariants and bypassing validations.
- **GAT:** Governance authority token. A token giving the credibility to modify the system.
- **AuthCheck:** Authority check. An optional GAT tag used in Agora-Pro. A minting policy checking that an effect takes place lawfully.
- **Governor:** A single instance validator governing the minting of GATs and the creation of proposals.

Methodology

The first phase of our audit collaboration was a design review. We reviewed the high-level design and suggested improvements. The most notable improvements include a more user friendly cosigning system, the ability to batch votes and thus vote even with lower stakes, ... We started the design review phase at commits:

- Agora repository: `e59fd16fe9fc45bb329de46ada1ad81ae7e37854`
- Agora-Pro repository: `fe7426af946dca283505609d8454236aab73b9d6`

After the design review, we conducted a deeper manual audit of the code and reported findings along with suggestions to the team in multiple batches, allowing the time for a proper remediation that we reviewed afterwards.

Our manual process focused on several types of attacks, including but not limited to:

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

The audit lasted from 7 September 2022 to 1 December 2022. We interacted mostly on Slack and gave feedback in GitHub pull requests. The team fixed all issues, except one that was acknowledged.

Files audited

The files and their hashes reflect the final state after all the fixes have been implemented. The respective commits:

- Agora repository: 76b3e8f197331cecd12e6fc75a82ab4de8e842c4
- Agora-Pro repository: 44090e06c9457e52caf00e3d4eaca3aa24d73ef9

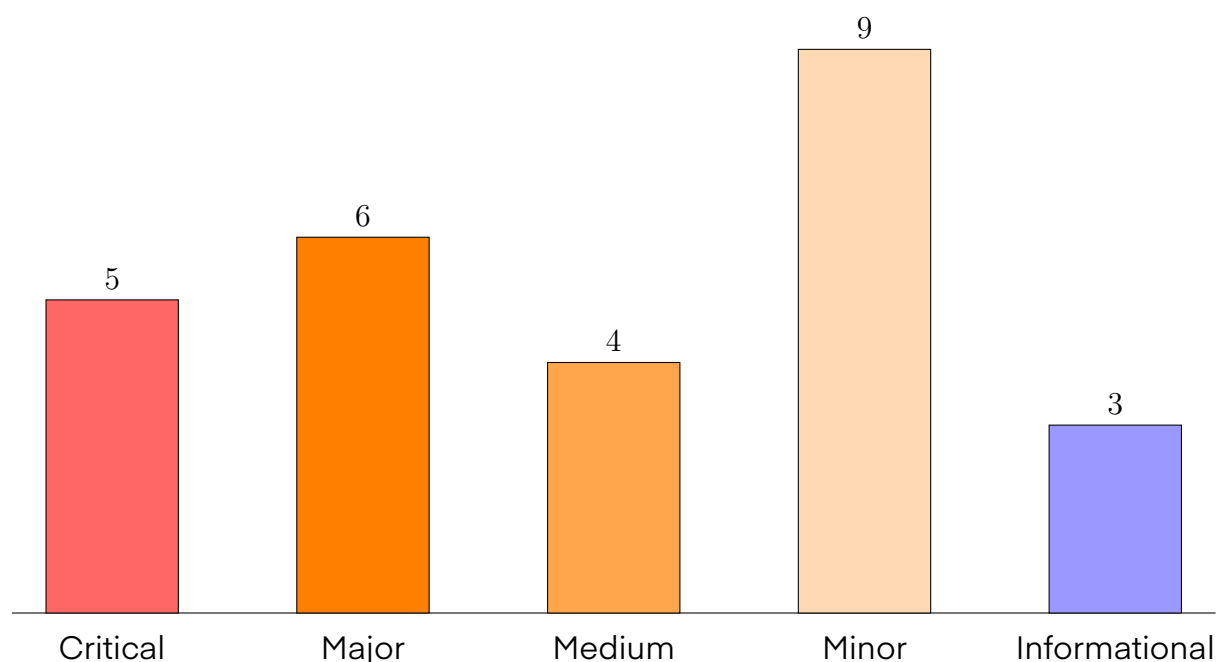
SHA256 hash	Filename
775a5...8e1e7	agora/Agora/Aeson/Orphans.hs
ad3ed...0bdc3	agora/Agora/AuthorityToken.hs
fd8cd...2d790	agora/Agora/Bootstrap.hs
3a85e...f5967	agora/Agora/Credential.hs
0def6...804d5	agora/Agora/Governor.hs
29e25...79e7e	agora/Agora/Governor/Scripts.hs
85fda...b116a	agora/Agora/Linker.hs
7dfd6...43628	agora/Agora/Plutarch/Orphans.hs
f59f9...90627	agora/Agora/Proposal.hs
c0ee3...0bec6	agora/Agora/Proposal/Scripts.hs
7c329...4e17b	agora/Agora/Proposal/Time.hs
e5032...d37a2	agora/Agora/SafeMoney.hs
10adf...8be3d	agora/Agora/Stake.hs
71968...6547e	agora/Agora/Stake/Redeemers.hs
e0cc8...59cc8	agora/Agora/Stake/Scripts.hs
eaf03...2b101	agora/Agora/Utils.hs
eeb39...8e886	agora/PPrelude.hs
940ed...a6823	agora-pro/Agora/Pro/AuthorityToken/Check.hs
e627b...3163c	agora-pro/Agora/Pro/AuthorityToken/Check/Burning.hs
4cb43...37546	agora-pro/Agora/Pro/Bootstrap.hs
56827...a514d	agora-pro/Agora/Pro/Linker.hs
0b401...ae6d6	agora-pro/Agora/Pro/MultiSig.hs
ee630...1253f	agora-pro/Agora/Pro/MultiSig/Scripts.hs
f7afb...bedc6	agora-pro/Agora/Pro/MultiSig/Utils.hs
57c47...89ab1	agora-pro/Agora/Pro/ScriptGuardian.hs

fb091...aa2b5	agora-pro/Agora/Pro/Stake/Scripts.hs
1c285...83504	agora-pro/Agora/Pro/StakeFreeze.hs
3cdc1...9ef63	agora-pro/Agora/Pro/StakeFreeze/Scripts.hs
f19b8...dc82f	agora-pro/Agora/Pro/StakeFreeze/Utils.hs
e2168...74371	agora-pro/Agora/Pro/Utils.hs
eeb39...8e886	agora-pro/PPrelude.hs

The code uses multiple functions from Liquid-Plutarch-Extra repository. We have audited those used in the files and revisions above as well.
















Please note that we did not audit the files not included in the above list that may or may not be part of the commit hash. We also did not assess the security of Plutarch itself. The assessment builds on the assumption that Plutarch is secure and delivers what it promises.




2 Severity overview





Findings

AGO-001	Stake state token can be taken away	■ Critical	Resolved ✓
AGO-002	Acting on behalf of delegatee role + Unlocking delegated stakes	■ Critical	Resolved ✓
AGO-003	Fake proposal can be created and GAT minted without any voting happening	■ Critical	Resolved ✓
AGO-004	Multiple GATs can be minted into fake scripts	■ Critical	Resolved ✓
AGO-005	Stake lock can be removed without retracting votes	■ Critical	Resolved ✓
AGO-101	Delegatee can steal delegated inputs	■ Major	Resolved ✓
AGO-102	Stake ST minting policy does not check LockedBy: Stake can be used to downvote without voting	■ Major	Resolved ✓

AGO-103	<code>pdestroy</code> doesn't check outputs: Stake owner is able to subtract votes even if they did not vote	 Major	Resolved 
AGO-104	Attacker can fail any voted-on/locked proposal	 Major	Resolved 
AGO-105	Stake ST token name not checked: Stake can be used to downvote without voting	 Major	Resolved 
AGO-106	Stake can retract all votes in its cooldown period	 Major	Resolved 
AGO-201	GATs are equal in the potential damage they can cause	 Medium	Acknowledged
AGO-202	<code>getStakeDatum</code> is staking credential sensitive	 Medium	Resolved 
AGO-203	Stakes can be frozen effectively forever by the multisig entity	 Medium	Resolved 
AGO-204	Governor can be DoSed by creating Proposals without passing min GT limit	 Medium	Resolved 
AGO-301	Ambiguity in destroying multiple stakes	 Minor	Resolved 
AGO-302	Some strict inequalities should not be strict	 Minor	Resolved 
AGO-303	<code>tooLate</code> in code can also mean early: An attacker can fast track a proposal into Finished state	 Minor	Resolved 
AGO-304	Other minor inconsistencies	 Minor	Resolved 
AGO-305	Other naming suggestions	 Minor	Resolved 
AGO-306	Proposal can be DoSed by using <code>UnlockStake</code> with no input stakes	 Minor	Resolved 
AGO-307	Proposal can be DoSed by using <code>UnlockStake</code> with relevant cosigners' stakes	 Minor	Resolved 
AGO-308	Proposal can be DoSed by voting and unlocking repeatedly	 Minor	Resolved 
AGO-309	Incorrect token references across the code	 Minor	Resolved 

AGO-401	Staking credential is undefined	 Informational	Resolved ✓
AGO-402	PUnlock: Ambiguous proposal redeemer name	 Informational	Resolved ✓
AGO-403	Delegatee can not vote with delegated and own stakes in one transaction	 Informational	Resolved ✓

AGO-001: Stake state token can be taken away

Severity:  Critical
Category: Logical Issue
Status: Resolved 

Description

Stake validator does not enforce that the stake state token (stake ST) does not leave the UTxO. A possible attack can be demonstrated by an attacker creating their stake normally that includes minting a stake ST into it. As the attacker is the owner of his stake, he can construct a transaction where he destroys the stake. He can, however, decide not to burn the stake ST and take it for himself instead.

By taking control of a stake ST, he can now construct any malicious stake UTxO with any malicious datum that would be trusted by other components as the stake ST is trusted to verify the validity of the datum information. We show that this is a critical vulnerability, as the stake datum, among other things, includes the notion of the amount of staked governance tokens (GTs) that is further relied upon. The attacker could therefore increase his voting power to effective infinity. That enables him to vote for his proposals. At the same time, it enables him to downvote opposing proposals trying to stop him. Once he gets his proposal through, he is able to mint a GAT and fully compromise the system.

Recommendation

Make sure that a stake ST can never leave a valid stake UTxO. Moreover, we suggest considering whether `stakedAmount` field could not be omitted from the stake datum. Checking the actual count of GTs would increase the difficulty of the proposed attack significantly.

Resolution

Fixed in the pull request number 195. `stakedAmount` field was left in the datum. However, the attack vector was eliminated.

AGO-002: Acting on behalf of delegatee role + Unlocking delegated stakes

Severity: ■ Critical
Category: Logical Issue
Status: Resolved ✓

Description

Suppose there are n different stakes delegated to the same delegatee D , each of the n stakes having a different owner. Attacker A (one of the n stake owners) can do the following:

- Put all n stakes into a transaction signed by A . Assume that A 's stake is the first input in the transaction.
- These do not have the same owner but they are delegated to D , so `allHaveSameDelegatee` in `stakeValidator` evaluates to true.
- `ownerSignsTransaction` evaluates to true, as A , the owner of the first stake input, signed the transaction.
- `PSignedByOwner` is written into `PSigContext.signedBy`.

As a result, `pisSignedBy` now always evaluates to true (it gets `PSignedByOwner` as the signature context). A got the same privileges as delegatee D just by delegating to D in terms of voting and retracting votes. D does not need to be a part of that transaction.

Moreover, A can also destroy all the n stakes and retrieve the GTs locked within as he authenticates as the owner of those all (recall that the signature context is `PSignedByOwner`).

Recommendation

The problem lies in the code determining the signature context without knowing the redeemer. We suggest coupling together the checks for owner signature with ownership of all inputs.

Resolution

Fixed in the pull request number 195 according to our recommendation.

AGO-003: Fake proposal can be created and GAT minted without any voting happening

Severity: ■ Critical

Category: Logical Issue

Status: Resolved ✓

Description

An attacker can do the following:

- Pick any legitimate locked proposal UTxO p in Execution period.
- Create a transaction:
 - Spend a proposal p with the `PAdvanceProposal` redeemer (Locked \rightarrow Finished).
 - Spend the governor UTxO with the `PMintGATs` redeemer.
 - Mint GATs normally (for the valid proposal p).
 - Also mint a valid proposal ST and take it into an attacker controlled address (note that the minting simply checks for the presence of governor ST).
- The UTxO with the new proposal ST is not checked anywhere since the policy relies on the governor, but the governor checks the proposal ST minting only under the `PCreateProposal` redeemer code block.
- The attacker can take his proposal ST and put it into any maliciously created proposal. Say she already marks the proposal as Locked, in the Execution period with the winner equal to a malicious effect.

As a result, the attacker mints a GAT and takes over the underlying system.

Recommendations

- Disallow minting of proposal STs during `PMintGATs` and `PMutateGovernor` redeemers (inside the Governor validator).
- `proposalPolicy` should disallow proposal ST in the inputs when minting.

Resolution

Fixed in the pull request number 205, by checking the governor redeemer in the proposal policy.

AGO-004: Multiple GATs can be minted into fake scripts

Severity: ■ Critical

Category: Bug

Status: Resolved ✓

Description

An attacker can create a transaction:

- Spend the governor UTxO with the `PMintGATs` redeemer.
- Spend proposal p with the `PAdvanceProposal` redeemer (Locked \rightarrow Finished). The winning effect group is e .
- Mint GATs normally for all the effects in the winning effect group e , one GAT per effect.
- Mint multiple (at least 2) GATs into a UTxO u with an attacker controlled validator.
- The `isAuthorityUTx0` flag is true for UTxOs with one GAT but it is false for u , which makes u being ignored in the `governorValidator` and the governor validation passes.
- `authorityTokenPolicy` has no check for the minted quantity. The minting policy passes.

The multiple GATs in u can be used to compromise the system.


Recommendations

- Disallow minting of multiple tokens at the same UTxO in `authorityTokenPolicy`, the governor validator (`isAuthorityUTx0`) and potentially also in the `authorityTokensValidIn` function.
- Check that the governor's redeemer is `PMintGATs` when minting GATs in the `authorityTokenPolicy`, similar to the proposal policy checking the governor's redeemer.


Resolution

Fixed in the pull request number 208 mostly according to our recommendations: there is a new guard against minting multiple tokens in the governor validator and the authority token policy has a check for the correct governor's redeemer.

AGO-005: Stake lock can be removed without retracting votes

Severity:  Critical

Category: Bug

Status: Resolved 

Description

This was introduced by the rewrite of `removeLocks`, in particular by the removal of `pnot` in a `pfilter # handleLock` and the formulation of some of the inner filter conditions. An attacker can do the following:

- Vote with his valid stake s on a proposal p 's effect group e .
- Vote with the stake s on a different proposal q 's effect group f .
- Wait until the stake voting cooldown period is over.
- Retract votes on f .
- Retracting the votes removes also the lock given on e .
- The attacker can vote on e with the same stake s again.

The process can be repeated any number of times, effectively making an attacker-desired effect group e win even with a small attacker's stake.

Recommendations

We suggest fixing the logic in the `removeLocks` function. `pfilter` keeps only those locks for which it evaluates to `true`. It is needed to make sure to keep all the locks that are not to be removed, e.g. those that do not belong to the particular proposal whose votes are being retracted.

On a slightly unrelated note, we also suggest not making the `unlockCooldown` argument of `removeLocks` a `Maybe` type. The `unlockCooldown` is always a number. Currently, setting it to `pnothing` influences the `removeLocks` functionality and makes it less transparent.

Resolution

Fixed in the pull request number 212.

AGO-101: Delegatee can steal delegated inputs

Severity:  Major
Category: Logical Issue
Status: Resolved 

Description

Some stake inputs do not need to have corresponding stake outputs. The function `pbatchUpdateInputs` uses `pfoldr # (pdeleteBy # f) # x # y` to check that elements of y can be transformed into elements of x by applying the function f . However, this will also work for cases when x is shorter than y , in particular if x is empty.

As a result, `pbatchUpdateInputs` checks for “subsets” instead of “set equality”. All handlers using `pbatchUpdateInputs` are affected.

Using this vulnerability, a delegatee can construct a voting transaction with multiple stakes that delegate to him. He can omit some stake outputs corresponding to stake inputs he spends, stealing the tokens contained in them (governance tokens, stake STs (per AGO-001) and min-Ada).

Recommendation

Add a check for the length of x and y or make the function `pdeleteBy` fail when it doesn't delete anything.

Resolution

Fixed independently in the pull request number 183 by introducing `pmustDeleteBy`.

AGO-102: Stake ST minting policy does not check `LockedBy`: Stake can be used to down-vote without voting

Severity:  Major
Category: Logical Issue
Status: Resolved 

Description

Suppose an attacker who owns GTs and does not have a stake created yet. The stake ST minting policy does not validate the emptiness of `StakeDatum.LockedBy` upon a stake UtxO creation. The attacker can put locks into their stake datum. This makes it look as if they voted on existing proposals before, even though they did not. The attack consists of these steps:

- The attacker can create a new stake UtxO, lock GTs in it and mint a stake ST into it. The new datum would already contain locks for existing effects.
- The stake the attacker owns resembles a properly created stake that has been used to vote on existing effects. The problem is, it has not been used to vote.
- The stake can be used to retract votes from those effects.
- Retracting votes unlocks the stake, it can be destroyed and the process can be repeated.

As a result, the attacker has retracted votes for effects for which they had not voted. In the worst case, the number of votes can become (near-)zero and the process can be freely repeated. The attacker can therefore freely sabotage any ongoing voting, possibly even changing the winning outcome.



Recommendation

Validate that `StakeDatum.LockedBy` is an empty list in the stake ST minting policy when the Stake is being created.

Resolution

Fixed independently in the pull request number 199.

AGO-103: pdestroy doesn't check outputs: Stake owner is able to subtract votes even if they did not vote

Severity:  Major
Category: Bug
Status: Resolved 

Description

The `pdestroy` function in `Agora/Stake/Redeemers.hs` does not check stake outputs. This means that an attacker can call the `PDestroy` redeemer on their stake and create a new stake UTxO with a datum that doesn't pass certain checks. In particular, there is no check on the value of `StakeDatum.LockedBy`.

- Create a transaction with a single stake input with the `PDestroy` redeemer. Use no other inputs and exactly one stake output.
- As `pdestroy` does not check the value of `StakeDatum.LockedBy` in the output stake datum, the attacker can add any proposal locks, e.g. for an effect e on a proposal p . The attacker is now the owner of a seemingly valid stake UTxO that is marked to have voted on p .
- In another transaction, the attacker takes the output of the previous transaction, applying the `PRetractVotes` redeemer on the effect e of proposal p .
- The attacker can repeat this process with their GTs for as long as they want as they ended up with an unlocked stake UTxO that can be destroyed again, another proposal can be down-voted, etc.

The attacker has retracted votes for the effect e for which they had not voted. In the worst case, the number of votes for e can become (near-)zero and the process can be freely repeated.



Recommendation

Add a check that there are no stake outputs when `PDestroy` redeemer is called.

Resolution

Fixed in the pull request number 200. `PDestroy` stake redeemer now requires no output stakes.

AGO-104: Attacker can fail any voted-on/locked proposal

Severity:  Major
Category: Bug
Status: Resolved 

Description

An attacker can change any voted-on/locked proposal to Finished too early. He can therefore effectively block all effects he does not agree with.

- `currentTime` in `agora/Agora/Proposal/Scripts.hs` is the transaction validity interval, not the current time. There is no limit on its length.
- The attacker can move the upper bound so far in the future that it is after the draft, voting, lock and execution periods. All `inDraftPeriod`, `inVotingPeriod`, `inLockingPeriod`, `inExecutionPeriod` are false when dealing with the `PAdvanceProposal` proposal redeemer.
- In particular, `notTooLate = false` and `notTooEarly = true` in the `PVotingReady` case.
- The only accepted output proposal status is `Finished`, so the attacker can move the proposal to the `Finished` state before the voting period is over.
- Also, `notTooLate = false` and `notTooEarly = true` in the `PLocked` case. The attacker can then make the proposal `Finished` without involving the governor. Even a proposal that passed a quorum will not get its effects executed.

Recommendation

- Limit the length of the transaction validity range (similar to `isTightEnough` check when creating a new proposal).
- Use the lower bound of the validity range as the current time (a point in time) in functions like `isDraftPeriod`, ...

Resolution

Fixed in the pull request number 200 by checking that the whole transaction validity range is contained within the relevant time period.

AGO-105: Stake ST token name not checked: Stake can be used to downvote without voting

Severity:  Major
Category: Logical Issue
Status: Resolved 

Description

In `stakePolicy`, the token name of the stake ST is set to be the same as the validator hash of the receiving script. However, `stakeValidator` never checks the token name, only the currency symbol. The attacker can do the following:

- Create a new valid stake ST s sent to `stakeValidator`.
- Create a new (invalid) stake ST t sent to f (a fake script).
- Send t to `stakeValidator` and add any proposal effect e into its `LockedBy` field.
- Use the `PDelegateTo` redeemer in one transaction for both stake inputs with s and t state tokens respectively. Swap the output datums, so the output with s gets the invalid datum d from the stake input with t . The fake script f always accepts, so this is a valid transaction.
- Use the `PRetractVotes` redeemer with the UTxO containing s and d to subtract votes from e . Note that s is a token with a valid token name, so the validation in `proposalValidator` passes.



Recommendation

Add an assertion for the proper stake ST token name to `stakeValidator`. It should be the same as the validator hash of the `stakeValidator`.

Resolution

Fixed in the pull request number 200 according to our recommendation.

AGO-106: Stake can retract all votes in its cooldown period

Severity:  Major
Category: Bug
Status: Resolved 

Description

An attacker can do the following:

- Vote with his valid stake s on a proposal p 's effect group e .
- The stake is now in a cooldown period to prevent voting and retracting votes too often (see AGO-308).
- The attacker can still retract votes, though. The votes would be retracted from e .
- However, the lock on the stake s would not be removed in the process. The reason is that the stake voting cooldown check is a filter condition filtering the locks and not a guard.
- As a consequence, there would be no change in the attacker's locks on s .
- As there's no change in the locks on s , the process can be repeated multiple times, bringing the vote count on e effectively to zero.

The attacker can manipulate the voting process. He can do so either directly by downvoting effects he does not agree with or by downvoting all other effects so that his preferred effect wins (given the effect got at least the quorum of votes).

Recommendations

We suggest using `guardC` which fails the transaction if there exists any stake lock that is in a cooldown period and that should be retracted. It can be left in the same place where the filtering now happens, in the `removeLocks` function.

Resolution

Fixed in the pull request number 212.

AGO-201: GATs are equal in the potential damage they can cause

Severity:  Medium

Category: Design Issue

Status: Acknowledged

Description

If any malicious proposal passes (e.g. due to insufficient quorum limits) or a seemingly legit proposal's effect validator or an AuthCheck that contains a critical vulnerability, it could collapse the whole underlying system (by mutating the governor first).

Recommendation

We think that there could be multiple tiers of GATs based on the criticality of the component they are able to update. The aim is to limit the damage of what could be done to the system. Analogous to a constitution amendment requiring more votes in the parliament.

Resolution

The team acknowledged the finding and explained it below:

"For protocols that want to control the maximum potential impact that GATs have, they actually can encode this in their validators. Instead of yielding to all GATs, they can instead choose to yield only under certain conditions, and yield only certain parts of their logic."

AGO-202: getStakeDatum is staking credential sensitive

Severity:  Medium
Category: Design Issue
Status: Resolved 

Description

The function `getStakeDatum` is used to find all input and output stake UTxOs in a transaction. It looks for UTxOs owning stake STs, but narrows them down by having to be on the correct **address**. The address is determined by the currently validated stake input. This is tricky as the staking credential is also part of the address and does not need to be constant across the protocol. By having multiple stakes with different staking credentials, every stake would see only a subset of them as stake inputs / outputs in that transaction.

Moreover, there could be a change of the staking credential in some stakes in a transaction. That could further complicate the amount of identified stake inputs vs. stake outputs and could potentially allow for another way to steal stake ST. The attack would likely depend on other issues, though, hence we are not exploring this further.

Recommendation

We suggest updating the filtering to search for all the inputs / outputs holding the stake ST. The invariant that stake STs can only be part of stake UTxOs should be maintained regardless. The fact could be further asserted at this place, but we would discourage using it as a filter condition.

Resolution

Fixed in the pull request number 195 according to our recommendation.

AGO-203: Stakes can be frozen effectively forever by the multisig entity

Severity:  Medium

Category: Logical Issue

Status: Resolved 

Description

The length of the transaction validity range is not checked in the StakeFreeze validator. The frozen time just needs to fall into the transaction validity interval. As a result, the multisig entity that is normally able to freeze the stakes for a short time is able to freeze them effectively forever. This can be done by setting the transaction validity range's upper bound to the far future (e.g. *now* + 1000 years) and choosing the frozen time to be equal to e.g. *now* + 999 years.

Stake owners are unable to destroy their stakes and to retrieve their governance tokens for 999 years.

Recommendation

We suggest taking the lower bound of the transaction validity range to set the frozen time. The lower bound is guaranteed to be in the past. It is in the multisig entity's interest to move it as much into the future as possible. Together, this likely results in the frozen time being near the actual time of the transaction construction.

Resolution

Fixed in the pull request number 18 in the `agora-pro` repository.

AGO-204: Governor can be DoSed by creating Proposals without passing min GT limit

Severity:  Medium

Category: Logical Issue

Status: Resolved 

Description

The function `getStakeDatum` in `agora/Agora/Governor/Scripts.hs` only checks the currency symbol of the stake ST token. An attacker can take advantage of this and create a proposal even without holding any GTs. The attacker can do the following:

- Mint stake ST token t with the wrong token name (as this is not enforced) into a script which always accepts.
- Set the datum to anything that conforms to the `StakeDatum` format, in particular setting high enough value for the `stakedAmount`.
- Use t with the fake datum to create a proposal. The high value of the `stakedAmount` field will exceed the proposal creation threshold.



Recommendation

Propagate the `sstAssetClass` into the `governorValidator` instead of the `sstSymbol` and then use it to also check the correct token name. Moreover, think about extracting a safe version of `getStakeDatum` into a utility function to be used across the codebase.

Resolution

Fixed in the pull request number 205 according to our recommendation.

AGO-301: Ambiguity in destroying multiple stakes

Severity:  Minor
Category: Logical Issue
Status: Resolved 

Description

It seems that destroying multiple stakes in a single transaction is allowed in the staking validator now. However, burning multiple stake STs is not allowed in the policy.



Recommendation

Allow burning multiple stake STs in the minting policy at once. Furthermore, we suggest making sure that all stake STs are burned if there is a stake being destroyed to prevent partial burn.

Resolution

Fixed in the pull request number 195 according to our recommendation.

AGO-302: Some strict inequalities should not be strict

Severity:  Minor
Category: Inconsistency
Status: Resolved 

Description

There are places where values are compared against min/max thresholds with strict inequality. However, minimum and maximum are more commonly understood as inclusive, e.g. a user could expect that the defined minimum stake for an action is enough, not that she needs a bit more (strict inequality). Examples include comparisons involving all the `ProposalThresholds` fields (except `stakedAmount`) and the `maximumCosigners` parameter.



Recommendation

Make the inequalities not strict.

Resolution

Fixed in pull requests number 200 and 208.

AGO-303: tooLate in code can also mean early: An attacker can fast track a proposal into Finished state

Severity:  Minor
Category: Bug
Status: Resolved 

Description

`ProposalDatum.startingTime` has to be within `createProposalTimeRangeMaxWidth` of the time when the proposal has been created. E.g. if the range is 120 minutes, the `startingTime` can be set 100 minutes into the future. A user can submit such a proposal with its starting time as far into the future as possible in hopes of getting more cosigners or votes, as this allows for an extra 100 minutes to get votes. However, if an action is made before the 100 minutes are over, the proposal can be moved to the Finished state. Further interactions with the proposal are thus not possible:

- In `Agora/Proposal/Scripts.hs`, `inDraftPeriod` is false, because the first 100 minutes are before the draft period. It implies that `notTooLate` is false. "Not not too late" translates to "too late" in English, however it is "too early" in the code.
- When `notTooLate` is false, only the Finished proposal output status is accepted.
- The attacker can change the proposal output status to Finished.
- Not much material damage has been done, other than the fees that have been paid by the proposal author and possibly the cosigners.



Recommendation

It depends on the fix of AGO-104. We suggest forcing the `startingTime` into the past by taking the lower bound of the transaction validity range which is always in the past. Then implement a similar updated `notTooLate` and `notTooEarly` logic compared to the other proposal statuses. Note that someone can still theoretically choose the transaction validity range's lower bound to be lower compared to the proposal's `startingTime`. The validation needs to be robust against this.

Resolution

Fixed in the pull request number 200 alongside the fix for the issue AGO-104.

AGO-304: Other minor inconsistencies

Severity:  Minor
Category: Inconsistency
Status: Resolved 

Description

The inconsistencies include:

- `pinsetUniqueBy` works only on sorted lists. This is not mentioned anywhere, please reflect it in the name and/or docstring to avoid incorrect usage.
- The `multisigPolicy` makes sure that the `MultiSig` is feasible (requires the signatures in that transaction). The `multisigValidator` does not enforce this upon its mutation, though. We suggest adding this.
- A proposal can be created or cosigned by the stake's delegatee as well. This is a non-revertable action by the stake owner. Some docs and comments mention that this shouldn't be possible. Should it be possible?
- Functions `pisFrozen` and `pisWithinCooldownPeriod` work on the lower bound of the transaction validity range. If the stakes are not frozen or they are not in the cooldown period, an attacker can push the lower bound into the past to make the function result look like they were frozen or in the cooldown period. Thanks to the aligned incentives, it is not worth doing. Adding the function documentation and/or updating the name of the functions would help clarify this, though.

Recommendation

Recommendations are part of the findings' descriptions.

Resolution

The first issue was upstreamed to `Liquid-Plutarch-Extra` and the docstring clarified the behavior. The second issue was deemed not intended and not desired, hence not implemented. The third issue was fixed in the pull request number 208 and the fourth issue was fixed in the pull request number 18 in the `agora-pro` repository.

AGO-305: Other naming suggestions

Severity:  Minor

Category: Naming

Status: Resolved 

Description

Naming suggestions include:

- LQ is used in some comments over the code. However, GT is a more generic naming; taking into account that Liqwid is not the only party using Agora governance module.
- Trace messages such as *"Execute threshold is less than or equal to 0"* do not correspond to reality now since the inequality changed (while fixing the issue AGO-302). *"or equal to"* should be dropped.
- `maximumProposalsPerStake` is a limit on the number of created proposals per stake. The total limit of locks is not enforced. We suggest updating the naming.
- `PCurrentTime` naming in Liqwid-Plutarch-Extra can be confusing, because it refers to the whole transaction validity range. The validity range can be very long with the actual current time somewhere in the long range.
- `Linker.hs` contains a line with `at = gstAssetClass`. Authority token naming is often used in terms of a GAT. Hence even though such naming makes sense on its own, it can be confusing in the context.



Recommendation

Recommendations are part of the descriptions.

Resolution

Fixed mostly in the pull request number 208, the fourth point has been fixed in the pull request number 214 and number 22 in the `agora-pro` repository.

AGO-306: Proposal can be DoSed by using UnlockStake with no input stakes

Severity:  Minor
Category: Bug
Status: Resolved 

Description

An attacker can:

- Spend a proposal with the PUnlockStake redeemer.
- Include no stake inputs.
- The transaction succeeds as the stake validation (e.g. `pnot # $ pisIrrelevant`) is not called when no stake is included.
- This is an empty transaction which does not change the proposal state. However, it changes the proposal UTxO reference and thus complicates the voting of other users at that time.

Repeated many times over, the voting period can be soon over and the voting impacted. The attacker pays just the transaction fees.



Recommendation

We suggest validating that at least one relevant stake is included when PUnlockStake redeemer is used on a proposal (meaning there is a state change on the proposal).

Resolution

Fixed in the pull request number 208.

AGO-307: Proposal can be DoSed by using UnLockStake with relevant cosigners' stakes

Severity:  Minor
Category: Bug
Status: Resolved 

Description

An attacker can:

- Spend a proposal p with the PUnLockStake redeemer.
- Include her stake s with which she cosigned (or created) the proposal previously.
- s is relevant to p (the `pisIrrelevant` function returns false).
- However, it is not possible to unlock the stake in the voting period as she has not voted. She has just cosigned (or created) the proposal p .
- The proposal state does not change as a result.
- The stake validation passes as well and does not remove any lock.

Together, this is another type of an empty transaction that can DoS the proposal. Repeated many times over, the voting period can be soon over and the voting impacted. The attacker pays just the transaction fees.

Recommendation

- The `pisIrrelevant` function should take into account which period the transaction takes place in. In the voting period, the stake should be an `pisVoter` and not just a `pnot # $ pisIrrelevant`.
- Update the stake validator to also not allow an empty `PRetractVote`, requiring that any lock is actually removed.

Resolution

Fixed in the pull request number 208 by requiring a change taking place in the proposal votes.

AGO-308: Proposal can be DoSed by voting and unlocking repeatedly

Severity:  Minor

Category: Bug

Status: Resolved 

Description

An attacker can:

- Create a stake s . Suppose the total amount of GTs staked in s exceeds the minimum voting requirements.
- In another transaction, vote in a chosen proposal p with s .
- In a follow-up transaction, unlock the stake s on the proposal p .
- Repeat the voting and unlocking transactions.

Repeated many times over, the voting period can be soon over and the voting impacted. The attacker pays just the transaction fees.

Recommendation



Up to a discussion as it involves business decisions. Some ideas include:

- Unlocking the stake with a cooldown period.
- Unlocking the stake just after the proposal is finished.
- Maintaining a score (how many voting / other interactions were performed over a certain rolling time period) and making sure the stake is not abused. This would also need to take into account the possibility of destroying the stake and creating it over again.

Resolution

Fixed in the pull request number 209.

AGO-309: Incorrect token references across the code

Severity:  Minor
Category: Naming
Status: Resolved 

Description

There are multiple instances of different tokens being referenced by name across the codebase. Non-exhaustive examples include:

- `stakePolicy`, `stakeValidator` all use `gstClass` which is consistent with the surrounding comments (referring to the governor ST). However, it is actually the governance token that is passed there, or `gtClass`.
- `authorityTokenPolicy` takes `atAssetClass` as an argument when in fact it is the `gstClass` used there instead.

Recommendation

We suggest going through the outlined places and nearby comments and correcting the references. Additionally, consider using longer and more descriptive variable names to avoid such typos in the future.

Resolution

Fixed in the pull request number 208.

AGO-401: Staking credential is undefined

Severity:  Informational

Category: Design Issue

Status: Resolved 

Description

The staking credential across the protocol's script UTxOs is undefined. Agora does not seem to hold a lot of ADA in its UTxOs. Therefore not having any logic regarding the staking credential does not itself look like a problem.

However, it could be a missed opportunity (collectively, the locked min-Ada in all the stakes can be a substantial amount). What's more, it may make off-chain looking for UTxOs harder as the addresses can differ. It is still doable by looking for the stakes by the stake ST instead of the address, though.

Recommendation

This is not a security issue and hence does not require any change.

Resolution

Fixed in the pull request number 208.

AGO-402: PUnl ock: Ambiguous proposal redeemer name

Severity:  Informational

Category: Naming

Status: Resolved 

Description

The redeemer typically denotes an action on an object. A proposal redeemer names an action on the proposal. A proposal can be Locked, so the redeemer PUnl ock suggests that the proposal is being unlocked, but in reality the stake is being unlocked.

Recommendation

Suggested better names: PUnl ockStake or PRe tractVotes.

Resolution

Fixed in the pull request number 200 by renaming to PUnl ockStake.

AGO-403: Delegatee can not vote with delegated and own stakes in one transaction

Severity:  Informational

Category: Design Issue

Status: Resolved 

Description

Currently, the stakes that are included in one transaction need to all have **either** the same owner **or** the same delegatee. The common use case, however, is in our opinion the delegatee constructing a single voting transaction with his own **plus** the stakes delegated to him. Since it is currently not possible to delegate a stake to the stake's owner, the delegatee is unable to make such a transaction. He can achieve the end goal by making two separate transactions only which is unintuitive.

Recommendation

Up to a discussion whether such behavior is wanted. In general, however, it could be possible to vote given either the owner or the delegatee of every stake separately authorizes the transaction. We do not see a need to strictly enforce either the same owner or the same delegatee of all the stakes, it could be a mix of the two.






Resolution

Fixed in the pull requests number 208 and 212.

Appendix


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.



Contact us:

audit@vacuumlabs.com