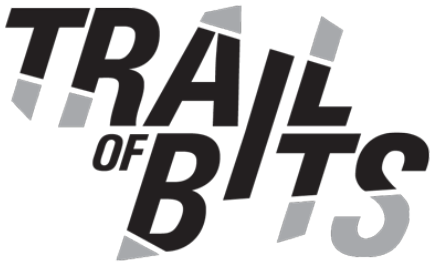

StakerDao wAlgo

Security Assessment

November 24, 2020

Prepared For:
Jonas Lamis | *StakerDao*
jonas@stakerdao.com

Christian Arita | *StakerDao*
christian@stakerdao.com

Pablo Yabo | *Rand*
pablo@randlabs.io

Prepared By:
Josselin Feist | *Trail of Bits*
josselin@trailofbits.com

Natalie Chin | *Trail of Bits*
natalie.chin@trailofbits.com

Changelog:

November 24, 2020:	Initial report delivered
December 11, 2020:	Added Appendix F with initial retest results
December 17, 2020:	Updated Appendix F

Executive Summary

From November 9 through November 23, StakerDao engaged Trail of Bits to review the security of the StakerDao wAlgo codebase. Trail of Bits conducted this assessment over the course of 4 person-weeks with 2 engineers working from `22b998a` from the StakerDao wAlgo repository.

During the first week, we reviewed the contracts focusing on understanding the contracts and how users interact with the vaults. During the second week, we continued the review with a focus on the `app-vault` operations.

Additionally, we built `tealer`, a static analyzer, to help the review of Teal code. The analyzer can show the control-flow-graph of a Teal code and integrate bug detectors and graph visualizations for Teal programs (Appendix E).

Our review resulted in 14 findings, from high to informational severity. The high severity issues include:

- The lack of access control on update and delete operations, allowing anyone to change the app-vault program (TOB-VAULT-001)
- The lack of clear state program check on the vault contract, allowing an attacker to drain a vault without paying back the wAlgo (TOB-VAULT-002)
- The lack of RekeyTo checks on mint and burn operations (TOB-VAULT-003, TOB-VAULT-004), allowing an attacker to drain a vault without paying back the wAlgo
- The lack of access control check on the minter account, which can be abused to drain the algo from a vault without paying back the wAlgo (TOB-VAULT-005)

All these high issues are the result of Algorand specificities and highlight the underlying risks of the platform. One of the main risks comes from the opt-in default behavior of Algorand - which allows all the operations unless explicitly refused. Appendix D contains checks that we recommend to follow when writing Teal code.

Overall the codebase represents a significant work in progress. Algorand is a new technology for which the security risks are not well known, and the tooling is low, or non-existent. Working on a young technology requires thorough documentation and testing, which the current codebase fails to achieve (TOB-VAULT-012, TOB-VAULT-013). Additionally, the mix of template and hardcoded value result in the programs not being up to date (TOB-VAULT-006, TOB-VAULT-007) and is another indicator that the codebase is not ready for production.

We recommend that StakerDao:

- Address all the reported findings,
- Thoroughly document the components, the actors, and their privileges

- Write a specification of each function, including its access controls, the operations allowed, and the arithmetic components
- Write a high level documentation of the system, and highlight the nature of each contract (stateful, stateless), their operations and their expected restrictions
- Follow an opt-out pattern by default, and only allow authorized operations (Appendix D)
- Fix the tests, and improve their coverage
- Integrate `tealer` (Appendix E) in the development process
- Consider a second security review

Update December 17, 2020: Trail of Bits reviewed fixes implemented for the issues presented in this report. See the results from this fix reviews in Appendix F: Fix Log.

Project Dashboard

Application Summary

Name	wAlgo
Version	22b998a
Type	Teal
Platforms	Algorand

Engagement Summary

Dates	November 9 2020 - November 23 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	8	■■■■■■■■
Total Medium-Severity Issues	2	■■
Total Low-Severity Issues	3	■■■
Total Informational-Severity Issues	1	■
Total Undetermined-Severity Issues	0	
Total	14	

Category Breakdown

Access Controls	3	■■■
Configuration	1	■
Data Validation	5	■■■■■
Error Reporting	2	■■
Undefined Behavior	1	■
Total	14	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Weak.** We found many high severity issues allowing an attacker to bypass the access controls protection and drain all the funds. Overall the codebase would benefit from a stricter authorization schema, broader documentation on the actors and their privileges and better associated testing.
Arithmetic	**Moderate.** The codebase relies only partially on arithmetic operation, mostly on the fee computation. We found an issue on the burning fee computation, showing that this area would benefit from more tests. Fuzzing or formal methods would help improve the confidence in the arithmetic.
Assembly Use	**Weak.** Several of the high severity issues are related to the usage of TEAL and Algorand specificities.
Centralization	**Moderate.** While the system is not meant to be decentralized in nature, we found that critical owner operations were not documented. Users could believe that the system operators have lower privileges than they have.
Upgradeability	**Weak.** We found that anyone can upgrade the `app-vault` program. This follows our evaluation of the access controls, and shows that the system would benefit from a broader documentation and testing on the actors and their privileges.
Function Composition	**Moderate.** The mixing of hardcoded parameters and templates in the codebase is error-prone, and leads the `app-vault` not using the updated vault bytecode.
Front-Running	**Satisfactory.** We found only a minor denial of service attack vector which allows attackers to prevent a user from opening a vault by sending funds into the user's vault address.
Key Management	**Not Considered.**
Monitoring	**Not Considered.**
Specification	**Weak.** A README and comments were provided, but code lacked fundamental documentation.
Testing & Verification	**Weak.** We found many issues in the current tests, including their mishandling of exceptions and areas where tests were not present.

Engagement Goals

The engagement was scoped to provide a security assessment of the StakerDao smart contracts in the stakerdao-vault repository.

Specifically, we sought to answer the following questions:

- Can an attacker escalate privilege to execute unauthorized operations?
- Is it possible to steal funds?
- Can participants manipulate the contract in unexpected ways?
- Is it possible for an attacker to mint more wAlgo than they have staked?

Coverage

App-vault approval and clear state programs. We reviewed the approval program and looked for flaws that would allow arbitrary users to perform privileged operations. We investigated if an attacker could withdraw the algo from a vault without paying back the expected wAlgo, and reviewed every user operation.

Minter and vault contracts. We reviewed the stateless contracts and looked for ways to call the minter or the vault without following the `app-vault` access controls and data validation. We reviewed the vault to see if it can be manipulated by a user that is not its creator.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

⛶ **Check that the user transactions are of `NoOp` type if they are not an `Optin` or a `CloseOut`.** This will prevent an attacker from updating or deleting the applications. TOB-VAULT-001

⛶ **Check in the vault that `OnCompletion` in the first transaction of the group is either NoOp or `CloseOut`.** This will prevent an attacker from bypassing all the code's logic by executing the clear state program. TOB-VAULT-002

⛶ **Check in the vault that there is no `rekeyto` operation in the transaction.** This will prevent an attacker from rekeying the vault. TOB-VAULT-003, TOB-VAULT-004

⛶ **Prevent the minter from self-transferring during the burning operation.** This will prevent attackers from burning without paying back the wAlgo. TOB-VAULT-005

⛶ **Set the bytecode during the creation of the app. Alternatively Use a template to generate the base64 of the vault bytecode in `app-vault` or check if it matches the latest version.** This will ensure the `app-vault` is up to date with the vault code. TOB-VAULT-006

⛶ **Maintain one version of the vault contract and use JavaScript to parse the file.** This ensures that a single standard codebase is updated. TOB-VAULT-007

⛶ **Document all the privileged operations that can be performed by the admin** to ensure that users are aware of the underlying risks when depositing funds. TOB-VAULT-008

⛶ **Check that the transaction fee in the minter stateless contract is equal to `MinTxnFee`.** This will ensure that all the transactions from the minter account have a limited fee. TOB-VAULT-009

⛶ **Ensure the system locks enough Algo with respect to the wAlgo minted during the burn operation by either (TOB-VAULT-010):**

- Checking if the minted wAlgo is above the vault's balance at the end of the burning operation, or

- Preventing these transactions from being called by the vault, or
- Ensuring that the wAlgo burned includes the fees paid by all the transactions, if the caller is the vault itself (both the transaction fee and the burning fee).

⛶ **Document that an attacker can prevent the creation of a new vault.** This ensures that users building complex integrations are aware of this corner case and can mitigate the attack. TOB-VAULT-011

⛶ **Catch all exceptions and report them visibly from the test runner so tests do not fail silently.** TOB-VAULT-012

⛶ **Add thorough unit and integration tests for the codebase to facilitate more complicated testing scenarios.** The system uses a new technology, Aglorand, and must be thoroughly tested to ensure it works as expected. TOB-VAULT-013

⛶ **Set `ASA_ID` during the creation of the app or use a template to generate it in `app-vault`.** This will ensure the `app-vault` is up to date with the asset value. TOB-VAULT-014

Long term

⛶ **Document the components of the system as well as each accepted and rejected user interaction.** Several issues targeting the access controls might have been prevented by thorough documentation. TOB-VAULT-001, TOB-VAULT-002, TOB-VAULT-005, TOB-VAULT-009

⛶ **Increase test coverage, and include negative tests for rejected transactions.** Several issues targeting the access controls might have been prevented by thorough testing. TOB-VAULT-001, TOB-VAULT-002, TOB-VAULT-005, TOB-VAULT-009

⛶ **Always disable `rekeyto` operations in stateless contracts unless explicitly required.** Otherwise an attacker can steal the stateless contracts funds through rekeying. TOB-VAULT-003, TOB-VAULT-004

⛶ **List which hardcoded parameters must be updated, and either use template or additional checks to ensure they stay up to date.** The use of hardcoded values is error-prone and might lead to deployment of outdated contracts. TOB-VAULT-006, TOB-VAULT-014

⛶ **Minimize the amount of manual copying and pasting required in order to run the tests.** List which hardcoded parameters must be updated, and either use a template or additional checks to keep them up to date. TOB-VAULT-007

⛶ **Identify and document all the actors and their associated privileges.** Documenting the system's actors and their privileges is required to properly evaluate the system. TOB-VAULT-008

⛶ **Investigate how to integrate fuzzing or a formal method to check the system's arithmetic properties.** Automated testing and verification are powerful techniques that can help ensure the code behaves as expected. TOB-VAULT-010

⛶ **Thoroughly evaluate the impact of every mitigation added to the system. Take into account that attackers can send Algo to any account when designing mitigation based on the contract's balance.** Mitigation can add new vulnerabilities if its not properly designed. TOB-VAULT-011

⛶ **Follow standard testing practice for smart contracts to minimize the number of issues during development.** The codebase is missing testing best practices, which increases the likelihood of undetected issues. TOB-VAULT-012

⛶ **Use a development framework to facilitate complex testing scenarios and integrate it into a CI/CD pipeline.** This will make it easier to add new tests and improve the robustness of the test suite. TOB-VAULT-013

Findings Summary

#	Title	Type	Severity
1	Anyone can update or delete the `app-vault`	Access Controls	High
2	Lack of clear state program check allows any vault to be drained	Data Validation	High
3	Missing `RekeyTo` on mint operations allows vault owner to withdraw all the Algo from the vault	Data Validation	High
4	Missing `RekeyTo` on burn operations allows vault owner to withdraw all the Algo from the vault	Data Validation	High
5	Minter can be abused to avoid paying the burned wAlgo	Data Validation	High
6	Incorrect vault bytecode usage	Patching	High
7	Code does not match	Configuration	High
8	Undocumented privileged operations	Undefined Behavior	High
9	Anyone can burn all the minter's Algo	Access Controls	Medium
10	With no fee consideration for burning operations the system is undercollateralized	Data Validation	Medium
11	Attackers can prevent a user from opening a vault	Access Controls	Low
12	Bad practices for exception handling in the test suite	Error Reporting	Low
13	Insufficient testing coverage	Error Reporting	Low
14	Hardcoded `ASA_ID` value is error-prone	Patching	Informational

1. Anyone can update or delete the `app-vault`

Severity: High
Type: Access Controls
Target: `app-vault-clear-state.teal`

Difficulty: Low
Finding ID: TOB-VAULT-001

Description
Without transaction field validation on user interactions, any user can update or delete `app-vault`. This means an attacker can update the application and steal the funds from all the associated vaults.

When an application transaction is successful, it can execute additional operations based on the `OnComplete` field:

ONCOMPLETE ¶

An application transaction must indicate the action to be taken following the execution of its approvalProgram or clearStateProgram. The constants below describe the available actions.

Value	Constant name	Description
0	NoOp	Only execute the `ApprovalProgram` associated with this application ID, with no additional effects.
1	OptIn	Before executing the `ApprovalProgram`, allocate local state for this application into the sender's account data.
2	CloseOut	After executing the `ApprovalProgram`, clear any local state for this application out of the sender's account data.
3	ClearState	Don't execute the `ApprovalProgram`, and instead execute the `ClearStateProgram` (which may not reject this transaction). Additionally, clear any local state for this application out of the sender's account data as in `CloseOutOC`.
4	UpdateApplication	After executing the `ApprovalProgram`, replace the `ApprovalProgram` and `ClearStateProgram` associated with this application ID with the programs specified in this transaction.
5	DeleteApplication	After executing the `ApprovalProgram`, delete the application parameters from the account data of the application's creator.

Figure 1.1: https://developer.algorand.org/docs/reference/teal/specification/#oncomplete.

For non-admin users, the approval program only checks for the `OptIn` and `CloseOut` operations:

```
// OptIn
int OptIn
txn OnCompletion
==
bnz no_admin_opt_in

[..]

// CloseOut
int CloseOut
txn OnCompletion
==
bnz no_admin_close_out
```

Figure 1.2: `app-vault.teal#L338-L369`.

Other transaction types, including `UpdateApplication` and `DeleteApplication`, are considered valid. This allows an attacker to update or delete the application by executing any valid operations, and steal or trap the funds of all the vaults.

Exploit Scenario
Bob has $1,000,000 worth of Algo locked in its vaults. Eve calls mint Algo on the `app-vault`, but adds the update operation in the transaction. Eve then updates the `app-vault` code, rekeys all the vaults, and steals Bob's funds.

Recommendations
Short term, check that the user transactions are of `NoOp` type if they are not an `Optin` or a `CloseOut`. This will prevent an attacker from updating or deleting the applications.

Long term, document and test the components of the system as well as each accepted and rejected user interaction. Several issues targeting the access controls might have been prevented by thorough documentation and test.

2. Lack of clear state program check allows any vault to be drained

Severity: High
Type: Data Validation
Target: `vault.teal.tmpl`

Difficulty: Low
Finding ID: TOB-VAULT-002

Description
The vault authorization can be abused by executing the clear state program on the `app-vault`.

To execute a transaction from a vault, the transaction must:

- Be in a group where the transaction at index 0:
  - Is the `app-vault`.
  - Has the vault address in the accounts list.
- Not be the first transaction in the group.

```
#pragma version 2
addr TMPL_USER_ADDRESS
pop

gtxn 0 ApplicationID
int TMPL_APP_ID
==

// do not allow to call the App from the Vault
txn GroupIndex
int 0
!=
&&

gtxn 0 Accounts 1
txn Sender
==
&&
```

Figure 2.1: `vault.teal.tmpl#L5-L18`.

The code assumes that the first transaction in the group to the `app-vault` will prevent abuse due to the approval contract. This assumption can be broken if the first transaction in the group is a call to the clear state program of the `app-vault`:

```
#pragma version 2
// approve
int 1
return
```

Figure 2.2: `app-vault-clear-state.teal#L1-L4`.

As a result, an attacker can execute any transaction from any vault by calling the clear state program of the `app-vault.`

Exploit Scenario
Bob has $1,000,000 worth of Algo locked in its vaults. Eve creates a group transaction, where the first transaction executes the clear state program of `app-vault`, and the second transaction transfers all of Bob's assets to her accounts (i.e., Eve steals Bob's funds).

Recommendations
Short term, check in the vault that `OnCompletion` in the first transaction of the group is either NoOp or `CloseOut`. This will prevent an attacker from bypassing all the code's logic by executing the clear state program.

Long term, document and test the components of the system as well as each accepted and rejected user interaction. Several issues targeting the access controls might have been prevented by thorough documentation and test.

3. Missing `RekeyTo` on mint operations allows vault owner to withdraw all the Algo from the vault

Severity: High
Type: Data Validation
Target: `app-vault.teal`

Difficulty: Low
Finding ID: TOB-VAULT-003

Description
The minting operation does not validate that its associated transaction has no `RekeyTo`. As a result, an attacker can rekey its vault and withdraw the Algo without paying back the wAlgo.

The minting operations has three transactions:

User mintwALGOs

- Tx0:
  - Sender: Vault owner account
  - arg0: str:mw
  - acc0: Vault address
  - Application Call tx
- Tx1:
  - Sender: Mint account
  - AssetReceiver: any account
  - Fee: MinTxnFee
  - AssetAmount: mint amount. The total minted amount must be less or equal to the ALGO Vault balance substracted the fees and it will need to keep at least the price of an additional tx to closeOut.
  - AssetCloseTo: ZeroAddress
  - XferAsset: 2671688 (betanet)
  - AssetTransfer tx

If MintFee > 0, a third tx is needed

- Tx2:
  - Sender: any
  - Receiver: Admin
  - Amount: Tx1.AssetAmount * MintFee / 10000
  - CloseRemainderTo: ZeroAddress
  - Payment tx

Figure 3.1: `README.md#user-mintwalgos`.

The last transaction (`Tx2`) is only required if there is a minting fee. In that case, the transaction will pay the associated fee.

`app-vault` does not check that `Tx2` has not set the `rekeyto`:

```
global GroupSize
 int 3
 ==
 &&

 load 9
 int 0
 >
 &&

 gtxn 2 TypeEnum
 int pay
 ==
 &&

 // calculate Admin fees: AssetAmount * MintFee / 10000
 load 8
 load 9
 *
 int 10000
 /
 dup
 store 6

 // calculate the total spend in the third tx by the Vault. Verify if the Vault is
paying the fees first.
 gtxn 2 Sender
 load 3
 ==
 bz no_admin_mint_walgos_fee_sender_not_vault

 // balanceVaultAfter = balance - tx2.Amount - tx2.Fee
 // position 7: tx2.Amount + tx2.Fee
 load 6
 gtxn 2 Fee
 +
 store 7

 no_admin_mint_walgos_fee_sender_not_vault:

 // fees == tx2.Amount
 gtxn 2 Amount
 ==
 &&

 // tx2.Receiver == AdminAddr
 gtxn 2 Receiver
```

```
load 4
==
&&

gtxn 2 CloseRemainderTo
global ZeroAddress
==
&&

// )
```

Figure 3.2: `app-vault.teal#L616-L671`.

A vault owner can use the vault account to pay the minting fee, and rekey the vault account to its address. Then the vault owner can withdraw all the Algo from the vault without paying back the corresponding wAlgo.

Exploit Scenario
Eve creates a vault and deposits $1,000,000 worth of Algo. She mints the corresponding wAlgo, rekeys the vault, and withdraws the Algo from the vault, thereby receiving $1,000,000 worth of wAlgo for free.

Recommendations
Short term, check in the vault that there is no `rekeyto` operation in the transaction. This will prevent an attacker from rekeying the vault.

Long term, always disable `Rekeyto` operations in stateless contracts unless explicitly required.

4. Missing RekeyTo on burn operations allows vault owner to withdraw all the Algo from the vault

Severity: High
Type: Data Validation
Target: `app-vault.teal`

Difficulty: Low
Finding ID: TOB-VAULT-004

Description
The burning operation does not validate that its associated transaction has no `RekeyTo`. As a result, an attacker can rekey its vault and withdraw the Algo without paying back the wAlgo.

The burning operation has three transactions:

User burnwALGOs

- Tx0:
  - Sender: Vault owner
  - arg0: str:bw
  - Application Call tx
- Tx1:
  - Sender: any account
  - AssetReceiver: Mint account
  - AssetAmount: burn amount. The total burned amount must be less or equal to total minted. It should be equal to arg1 Tx0
  - XferAsset: 2671688 (betanet)
  - AssetTransfer tx

If BurnFee > 0, a third tx is needed

- Tx2:
  - Sender: any
  - Receiver: Admin
  - Amount: Tx1.AssetAmount * BurnFee / 10000
  - CloseRemainderTo: ZeroAddress
  - Payment tx

Figure 4.1: `README.md#user-burnwalgos`.

`app-vault` does not check that `Tx1` and `Tx2` have not set the `rekeyto`:

```
// AssetTransfer
gtxn 1 TypeEnum
```

```
int axfer
==

load 3
txn Accounts 1
==
&&

// (
global GroupSize
int 2
==

load 9
int 0
==
&&
bnz no_admin_burn_walgos_no_fee

global GroupSize
int 3
==
&&

load 9
int 0
>
&&

gtxn 2 TypeEnum
int pay
==
&&

// calculate Admin fees: AssetAmount * MintFee / 10000
gtxn 1 AssetAmount
load 9
*
int 10000
/

gtxn 2 Amount
==
&&

// tx2.Receiver == AdminAddr
gtxn 2 Receiver
load 4
==
&&

gtxn 2 CloseRemainderTo
global ZeroAddress
```

```
==
&&

// )

no_admin_burn_walgos_no_fee:

// more than 0
gtxn 1 AssetAmount
int 0
>
&&

// Minter == ASA Receiver
gtxn 1 AssetReceiver
byte "MA" // MintAccount
app_global_get
==
&&

// ASA_ID
gtxn 1 XferAsset
int 2671688
==
&&

bz failed
```

Figure 4.2: `app-vault.teal#L848-L929`.

A vault owner can use the vault account to burn the wAlgo or pay the burning fee, and rekey the vault account to its address. Then the vault owner can withdraw all the Algo from the vault without paying back the corresponding wAlgo.

Exploit Scenario

Eve creates a vault and deposits $1,000,000 worth of Algo. She mints the corresponding wAlgo and burns one token, then rekeys the vault and withdraws all the Algo from it, thereby receiving $1,000,000 worth of wAlgo for free.

Recommendations

Short term, check in the vault that there is no `rekeyto` operation in the transaction. This will prevent an attacker from rekeying the vault.

Long term, always disable `rekeyto` operations in stateless contracts unless explicitly required.

5. Minter can be abused to avoid paying the burned wAlgo

Severity: High
Type: Data Validation
Target: `app-vault.teal`

Difficulty: Low
Finding ID: TOB-VAULT-005

Description
Without access controls on the minter account, an attacker can withdraw its Algo without paying back the wAlgo.

Burning allows a user to unlock the Algo from its vault. To burn, the user must pay the corresponding wAlgo to the minter:

User burnwALGOs

- Tx0:
  - Sender: Vault owner
  - arg0: str:bw
  - Application Call tx
- Tx1:
  - Sender: any account
  - AssetReceiver: Mint account
  - AssetAmount: burn amount. The total burned amount must be less or equal to total minted. It should be equal to arg1 Tx0
  - XferAsset: 2671688 (betanet)
  - AssetTransfer tx

If BurnFee > 0, a third tx is needed

- Tx2:
  - Sender: any
  - Receiver: Admin
  - Amount: Tx1.AssetAmount * BurnFee / 10000
  - CloseRemainderTo: ZeroAddress
  - Payment tx

Figure 5.1: `README.md#user-burnwalgos`.

`Tx1`, which transfers the wAlgo to the minter account, can be sent by the minter itself:

```
#pragma version 2
// Minter Delegate Teal
// Allows App to mint wAlgos to Vault users
// TMPL_APP_ID: Application ID
// TMPL_ASA_ID: wALGOs id

gtxn 0 ApplicationID
int TMPL_APP_ID
==

gtxn 1 TypeEnum
int 4
==
&&

// ASA ID
gtxn 1 XferAsset
int TMPL_ASA_ID
==
&&

txn RekeyTo
global ZeroAddress
==
&&

txn AssetCloseTo
global ZeroAddress
==
&&

// do not allow to call the App from the Vault, only allow calls in index 1 that
are XferAsset
txn GroupIndex
int 1
==
```

Figure 5.2: `minter.teal.tmpl`.

As a result, vault owners can avoid paying back the wAlgo to unlock their Algo.

Exploit Scenario
Eve creates a vault and deposits $1,000,000 worth of Algo. She mints the corresponding wAlgo, calls burn, and uses the minter to burn the wAlgo. Eve then withdraws the Algo from the vault, receiving $1,000,000 worth of wAlgo for free.

Recommendations
Short term, prevent the minter from self-transferring during the burning operation. This will prevent attackers from burning without paying back the wAlgo.

Long term, clearly document the components of the system as well as each accepted and rejected user interaction. Additionally, increase the test coverage, and include negative tests for rejected transactions.

6. Incorrect vault bytecode usage

Severity: High
Type: Patching
Target: `app-vault.teal`

Difficulty: Low
Finding ID: TOB-VAULT-006

Description
`app-vault` requires the vault bytecode to generate the vault addresses. The vault bytecode used does not match the bytecode generated from the vault source code.

`app-vault` uses the base64 of the vault bytecode:

```
// Prefix (see README.md)
byte base64 AiAC3rekAQAmASA=
concat
```

Figure 6.1: `app-vault.teal#L450-L452`.

```
// Suffix (see README.md)
byte base64 KEgzABgiEjEWIxMQNwAcATEAEhAxIDIDEhA=
concat
```

Figure 6.2: `app-vault.teal#L457-L459`.

The base64 does not match the base64 from the `README`:

- Convert the base64 to Hex
  - 022002a0d7a30100260120e607408d6d6547c905aac86b42783b503a1b322d5007c95b39059ac7085a9732284833001822123116231310370 01c0131001210312032031210
  - 022002a0d7a30100260120b6e95ebb559a702e27c3c1240003d069e1819e8b238977148b04fe521e7f94b628483300182212311623131037001c0131001210312032031210
- Identify the part at the beginning and at the end that are exactly the same in both bytestreams:
  - First part: 022002a0d7a30100260120
  - Last part: 28483300182212311623131037001c0131001210312032031210

Figure 6.3: `README.md#change-vaultteal`.

It also does not match the base64 of the bytecode from the current `vault.teal.tmpl` (or `vault.js`; see issue TOB-VAULT-007):

Source	Prefix
app-vault	022002deb7a40100260120
Readme	022002a0d7a30100260120
vault.teal.tmpl	022002d0b8a30600260120

As a result, the `app-vault` will not provide the security guarantees of the reviewed vault and might contain additional vulnerabilities.

Exploit Scenario
wAlgo is deployed with a vault that does not contain the fixes for the issues reported in this audit. Eve exploits a vulnerability in the vaults and drains all the funds.

Recommendations
Short term, set the bytecode during the creation of the app . Alternatively use a template to generate the base64 of the vault bytecode in `app-vault` or check if matches the latest version. This will ensure the `app-vault` is up to date with the vault code.

Long term, list what hardcoded parameters must be updated, and either use template or additional checks to ensure they stay up to date.

7. Code does not match

Severity: High
Type: Configuration
Target: `vault.js`

Difficulty: High
Finding ID: TOB-VAULT-007

Description
The current development practice involves copying code from string variables in `vault.js` into the `vault.teal.tmpl` file. This practice is error-prone, as it requires manual copying and pasting of the TEAL code.

Figure 7.1 shows the string variable `vaultTEAL`, which represents the code for the vault:

```
var vaultTEAL =
`#pragma version 2
addr TMPL_USER_ADDRESS
pop

gtxn 0 ApplicationID
int TMPL_APP_ID
==

// do not allow to call the App from the Vault
txn GroupIndex
int 0
!=
&&

gtxn 0 Accounts 1
txn Sender
==
&&

txn RekeyTo
global ZeroAddress
==
&&
```

```
`
```

Figure 7.1: `vaultTEAL` variable in `vault.js#L35-L60`.

However, this code needs to be copied into the `vault.teal.tmpl` file. Failure to do so results in a mismatch between the two files, including omission of an important `Rekey` check:

```
#pragma version 2
addr TMPL_USER_ADDRESS
pop

gtxn 0 ApplicationID
int TMPL_APP_ID
==

// do not allow to call the App from the Vault
txn GroupIndex
int 0
!=
&&

gtxn 0 Accounts 1
txn Sender
==
&&
```

Figure 7.2: `vault.teal.tmpl`.

Exploit Scenario
Bob rewrites the deployment script. The new script opens `vault.teal.tmpl`. Because `vault.teal.tmpl` was not updated, the issues shown in the report are still present. Eve exploits one of the vault's flaws to drain all the funds.

Recommendations
Short term, maintain one version of the vault contract and use JavaScript to parse the file. This ensures that a single standard codebase is updated.

Long term, minimize the amount of manual copying and pasting required in order to run the tests. List which hardcoded parameters must be updated, and either use a template or additional checks to keep them up to date.

8. Undocumented privileged operations

Severity: High
Type: Undefined Behavior
Target: `app-vault.teal`

Difficulty: High
Finding ID: TOB-VAULT-008

Description
Several admin operations are not documented. This lack of transparency can mislead users regarding the system's centralization.

The privileged operations that are documented include:

Admin setGlobalStatus

The admin can enable or disable any vault at any time.

- Tx0:
  - Sender: Admin
  - arg0: integer: new status (0 or 1)
  - Application Call tx

Admin setAccountStatus

The Admin can enable or disable any vault at any time.

- Tx0:
  - Sender: Admin
  - acc0: User Address
  - arg0: integer: new status (0 or 1)
  - Application Call tx

Admin setMintFee

Set the percent of paid in ALGOs on each mintwALGOs operation

- Tx0:
  - Sender: Admin
  - arg0: integer: new fee (0 to 5000 which means 0%-50%)
  - Application Call tx

Admin setBurnFee

Set the percent of ALGOs reserved for the Admin on each burnwALGOs operation

- Tx0:
  - Sender: Admin
  - arg0: integer: new fee (0 to 5000 which means 0%-50%)
  - Application Call tx

Admin setCreationFee

Set the fee in ALGOs that is required to send to Admin to optin to the App

- Tx0:
  - Sender: Admin
  - arg0: integer: new fee in microALGOs
  - Application Call tx

Figure 8.1: `README.md#application-calls`.

However, the admin can perform many additional critical operations, including:

- Updating the `app-vault` program.
- Deleting the `app-vault` program.
- Updating the mint/admin account.

Exploit Scenario
Bob creates a vault. Bob thinks that its funds are safe, and that no one can steal its assets. Eve, a malicious admin of the `app-vault`, changes the `app-vault` approval program and steals Bob's assets.

Recommendation
Short term, document all the privileged operations that can be performed by the admin to ensure that users are aware of the underlying risks when depositing funds.

Long term, identify and document all the actors and their associated privileges. Documenting the system's actors and their privileges is required to properly evaluate the systemidentify and document all the actors, and their associated privileges.

9. Anyone can burn all the minter's Algo

Severity: Medium
Type: Access Controls
Target: `app-vault.teal, minter.teal.tmpl`

Difficulty: Low
Finding ID: TOB-VAULT-009

Description
Without a transaction fee check on the burning operation, anyone can burn all the Algo from the minter's account.

To burn wAlgo, the user sends a transaction on behalf of the minter account (`Tx1`):

User burnwALGOs

- Tx0:
  - Sender: Vault owner
  - arg0: str:bw
  - Application Call tx
- Tx1:
  - Sender: any account
  - AssetReceiver: Mint account
  - AssetAmount: burn amount. The total burned amount must be less or equal to total minted. It should be equal to arg1 Tx0
  - XferAsset: 2671688 (betanet)
  - AssetTransfer tx

If BurnFee > 0, a third tx is needed

- Tx2:
  - Sender: any
  - Receiver: Admin
  - Amount: Tx1.AssetAmount * BurnFee / 10000
  - CloseRemainderTo: ZeroAddress
  - Payment tx

Figure 9.1: `README.md#user-burnwalgos`.

There is no restriction on the fee in `Tx1`, so anyone can burn all the Algo from the minter's account and make the system unavailable.

Exploit Scenario

The system is deployed, and `StakerDao` provides the minter account with 10 Algo to allow users to call the minter. Eve creates a burn transaction, and consumes the 10 Algo in the fee. As a result, the system becomes unavailable.

Recommendations
Short term, check that the transaction fee in the minter stateless contract is equal to `MinTxnFee`. This will ensure that all the transactions from the minter account have a limited fee.

Long term, clearly document the components of the system as well as each accepted and rejected user interactions. Additionally, increase the test coverage, and include negative tests for rejected transactions.

10. With no fee consideration for burning operations the system is undercollateralized

Severity: Medium
Type: Data Validation
Target: `app-vault.teal`

Difficulty: Low
Finding ID: TOB-VAULT-010

Description
The system assumes that every vault cannot have more Algo minted than its current balance. This assumption can be broken through the fees paid by the burning operations.

When burning wAlgo:

- The second transaction (`Tx1`) can be called from the vault and has to pay the transaction fee.
- The third transaction (`Tx2`) can be called from the vault and has to pay the transaction fee and the burning fee.

User burnwALGOs

- Tx0:
  - Sender: Vault owner
  - arg0: str:bw
  - Application Call tx
- Tx1:
  - Sender: any account
  - AssetReceiver: Mint account
  - AssetAmount: burn amount. The total burned amount must be less or equal to total minted. It should be equal to arg1 Tx0
  - XferAsset: 2671688 (betanet)
  - AssetTransfer tx

If BurnFee > 0, a third tx is needed

- Tx2:
  - Sender: any
  - Receiver: Admin
  - Amount: Tx1.AssetAmount * BurnFee / 10000
  - CloseRemainderTo: ZeroAddress
  - Payment tx

Figure 10.1: `README.md#user-burnwalgos`.

The amount of wAlgo minted decreases only by the amount of wAlgo burned:

```
// Minted = Minted - tx1.AssetAmount
load 8
gtxn 1 AssetAmount
-
store 8

int 0
byte "m" // minted
load 8
app_local_put

b success
```

Figure 10.2: app-vault.teal#L940-L951.

This does not take into consideration that fees paid by the Tx1 and Tx2 can decrease the vault's balance.

If the fees paid are greater than the wAlgo burned, a vault can hold less Algo than its wAlgo debt amount. As a result, the amount of wAlgo minted can be slightly higher than the total amount of Algo locked, and the system is undercollateralized.

Exploit Scenario

- Eve creates a vault, and deposits 2,000 microAlgo.
    - The vault has 2,000 microAlgo and a debt of 0 wAlgo.
- Eve mints 2,000 wAlgo (microAlgo).
    - The vault has 2,000 microAlgo and a debt of 2,000 wAlgo.
- Eve burns one wAlgo and pays 1,000 microAlgo in fees.
    - The vault has 1,000 microAlgo and a debt of 1,999 wAlgo.

Eve uses this difference to publicly attack the system's reputation. As a result, users lose their trust in the system.

Recommendations

Short term, ensure the system locks enough Algo with respect to the wAlgo minted during the burn operation by either:

- Checking if the minted wAlgo is above the vault's balance at the end of the burning operation, or
- Preventing these transactions from being called by the vault, or
- Ensuring that the wAlgo burned includes the fees paid by all the transactions if the caller is the vault itself (both the transaction fee and the burning fee).

Long term, investigate how to integrate fuzzing or a formal method to check the system's arithmetic properties.

11. Attackers can prevent a user from opening a vault

Severity: Low
Type: Access Controls
Target: `app-vault.teal`

Difficulty: Low
Finding ID: TOB-VAULT-011

Description
`app-vault` prevents users from adding a vault if its current balance is positive. This can be abused by an attacker to prevent a user from opening a vault.

To prevent attackers from calling the clear state program and re-opening a vault without paying its outstanding debt, `app-vault` checks that the vault has no Algo when it is opened:

```
// if balance > 0 is allowed on creation, the user can call ClearState, reset the
amount minted and withdraw the Algos without
// returning the wALGOs
int 1
balance
int 0
==
bz failed
```

Figure 11.1: app-vault.teal#L439-L445.

An attacker can abuse this mitigation by sending Algo to the vault before it has opted in, thereby preventing a specific user from opening a vault.

Exploit Scenario
Bob and Alice create a mult-sig account controlled by a stateless contract. The contract allows it to interact with its associated vault, but does not allow it to withdraw the Algo directly. Bob and Alice send $100,000 worth of Algo to the contract. Eve sends 1 micro-Algo to the corresponding vault before it has opted-in. As a result, Bob's and Alice's funds are trapped in the multi-sig wallet.

Recommendations
Short term, document that an attacker can prevent the creation of a new vault. This ensures that users building complex integrations are aware of this corner case and can mitigate the attack.

Long term, thoroughly evaluate the impact of every mitigation added to the system. Take into account that attackers can send Algo to any account when designing mitigation based on the contract's balance.

12. Bad practices for exception handling in the test suite

Severity: Low
Type: Error Reporting
Target: `test.js`

Difficulty: Low
Finding ID: TOB-VAULT-012

Description
The test suite does not provide sufficient exception handling in test cases. This error-prone behavior can allow failed tests to be overlooked, or prevent the addition of new tests.

In particular, the tests fail to:

- Stop properly when running into exceptions. When testing for failed transaction calls, the tests silently catch all exceptions and log errors, so the system is unable to differentiate HTTP errors (such as Bad Request) from proper error messages.

```
setBurnFee: should fail
setBurnFee successfully failed: Bad Request
setBurnFee: should fail
setBurnFee successfully failed: Bad Request
```

Figure 12.1: Unsuccessful burn fee terminal output.

- Ensure all errors are caught properly. The test suite makes asynchronous calls to `vault.js`, which does not account for promise rejections, resulting in `UnhandledPromiseRejection` errors.

```
optIn
(node:10660) UnhandledPromiseRejectionWarning: Error: ERROR:
{"message":"TransactionPool.Remember: transaction
CLS4CNDVVOTYU7TFZENFMWDIXEKN7RFK4BDYUCUNOT3XIWUUZTHQ: transaction rejected by
ApprovalProgram"}
    at main (test.js:721:9)
    at processTicksAndRejections (internal/process/task_queues.js:93:5)
(Use `node --trace-warnings ...` to show where the warning was created)
(node:10660) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This
error originated either by throwing inside of an async function without a catch
block, or by rejecting a promise which was not handled with .catch(). To terminate
the node process on unhandled promise rejection, use the CLI flag
`--unhandled-rejections=strict` (see
https://nodejs.org/api/cli.html#cli_unhandled_rejections_mode). (rejection id: 1)
(node:10660) [DEP0018] DeprecationWarning: Unhandled promise rejections are
deprecated. In the future, promise rejections that are not handled will terminate
the Node.js process with a non-zero exit code.
```

Figure 12.2: Test output failure due to unhandled promise rejection.

Exploit Scenario

StakerDao adds new functionality to the codebase and runs the tests to ensure behavior is uniform. The test suite has trouble connecting with the betanet node and throws a `400 BAD REQUEST` error, which is printed out in the terminal logs. Because the error is not reported properly, StakerDao assumes the tests have run properly, and deploys untested code.

Recommendations
Short term, catch all exceptions and report them visibly from the test runner so tests do not fail silently.

Long term, follow standard testing practice for smart contracts to minimize the number of issues during development.

13. Insufficient testing coverage

Severity: Low
Type: Error Reporting
Target: `test.js`

Difficulty: Low
Finding ID: TOB-VAULT-013

Description
The StakerDao Vault codebase does not include sufficient test cases. Robust unit and integration tests are critical for catching the introduction of certain bugs and logic errors early in the development process. Projects should strive for thorough coverage, both against bad inputs as well as "happy path" scenarios. This will greatly increase confidence in the functionality of the code for users and developers alike.

Example test case improvements include:

- Admins and users deleting application.
- Non-admins attempting to update application.
- Non-admin attempting to update mint account address.
- Non-admin attempting to set account status.

Additionally, unit tests are a beneficial addition for the testing architecture to verify that subsets of the application work locally as expected. These tests allow developers to effectively detect early inconsistencies in code behavior prior to deployment.

Exploit Scenario
StakerDao discovers a bug in the application and writes a patch to update it. The new code causes backward incompatibility and deviates from the contract's expected behavior. Due to insufficient testing coverage, this is not caught by the testing suite.

Recommendations
Short term, add thorough unit and integration tests for the codebase to facilitate more complicated testing scenarios.

Long term, use a development framework to facilitate complex testing scenarios and integrate it into a CI/CD pipeline.

References
- Chai assertions
- Mocha testing framework
- Algorand Private Network

14. Hardcoded ASA_ID value is error-prone

Severity: Informational
Type: Patching
Target: app-vault.teal, settings.js

Difficulty: Low
Finding ID: TOB-VAULT-014

Description
app-vault requires the asset ID value to ensure correct assets transfers. This value is currently hardcoded to the betanet asset ID. The use of hardcoded value is error-prone and might allow the codebase to be deployed with an incorrect asset.

```
// ASA_ID
gtxn 1 XferAsset
int 2671688
```

Figure 14.1: app-vault.teal#L704-L706.

```
// ASA_ID
gtxn 1 XferAsset
int 2671688
```

Figure 14.2: app-vault.teal#L923-L925.

Additionally, the test setup uses a hardcoded value to identify a deployed contract on Algorand's betanet to run tests against. This practice is error-prone, as failure to update the appId after deployment results in testing against the wrong contract.

```
appId: 2694110,
```

Figure 14.3: settings.js#L9.

Exploit Scenario
wAlgo is deployed, but the ASA_ID is not updated, sothe wAlgo used is worthless and does not match the StakerDao assets.

Recommendations
Short term, set ASA_ID during the creation of the app, or use a template to generate it in app-vault. This will ensure the app-vault is up to date with the asset value.

Long term, list which hardcoded parameters must be updated, and either use template or additional checks to ensure they stay up to date.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important

Medium	Individual user information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

App-vault

- **Branch to `success` in `setMintFee` instead of returning (app-vault.teal#L272-L273).** This will improve the consistency in the codebase.

D. Algorand Checklist

General:

- Disable all the features by default, and use an opt-in approach.

Stateless contract:

- Check the transaction fee is below a threshold or equal to `MinTxnFee`.
- Check that `RekeyTo` operation is disabled unless explicitly approved.
- Check that `CloseRemainderTo` is disabled unless explicitly approved.
- Check that `AssetCloseTo` is disabled unless explicitly approved.
- Disable all the transactions types unless explicitly approved.

Stateful contract:

- Ensure that all the paths check for `ApplicationUpdate`/`ApplicationDelete`.

Transaction group:

- Check that the `GroupSize` tis equal to the expected number of transactions.
- Check that the `GroupIndex` is equal to the expected index.
- Check the sender of each external transaction.
- Consider checking the `ApplicationArgs` of stateful transactions in stateless transactions.

E. Teal Analyzer

During the audit, we developed tealer, a static analyzer for Teal code.

The analyzer parses Teal code and builds the control-flow-graph of the graph. It contains vulnerability detectors and allows the program to be shown through different visualizations.

Currently, the analyzer can:

- Print the CFG of the teal code.
- Detect paths with a missing `RekeyTo` check (group transaction) (TOB-VAULT-003, TOB-VAULT-004).
- Detect paths with a missing `GroupSize` check (group transaction).
- Print paths that can delete the application (TOB-VAULT-001).
- Print paths that can update the application (TOB-VAULT-001).

Figure E.1 shows all the paths of the app-vault that can delete the application (TOB-VAULT-001):

```
tealer app-vault.teal --print-delete
```

We recommend adding tealer to the CI and using it to review future versions of the contracts.

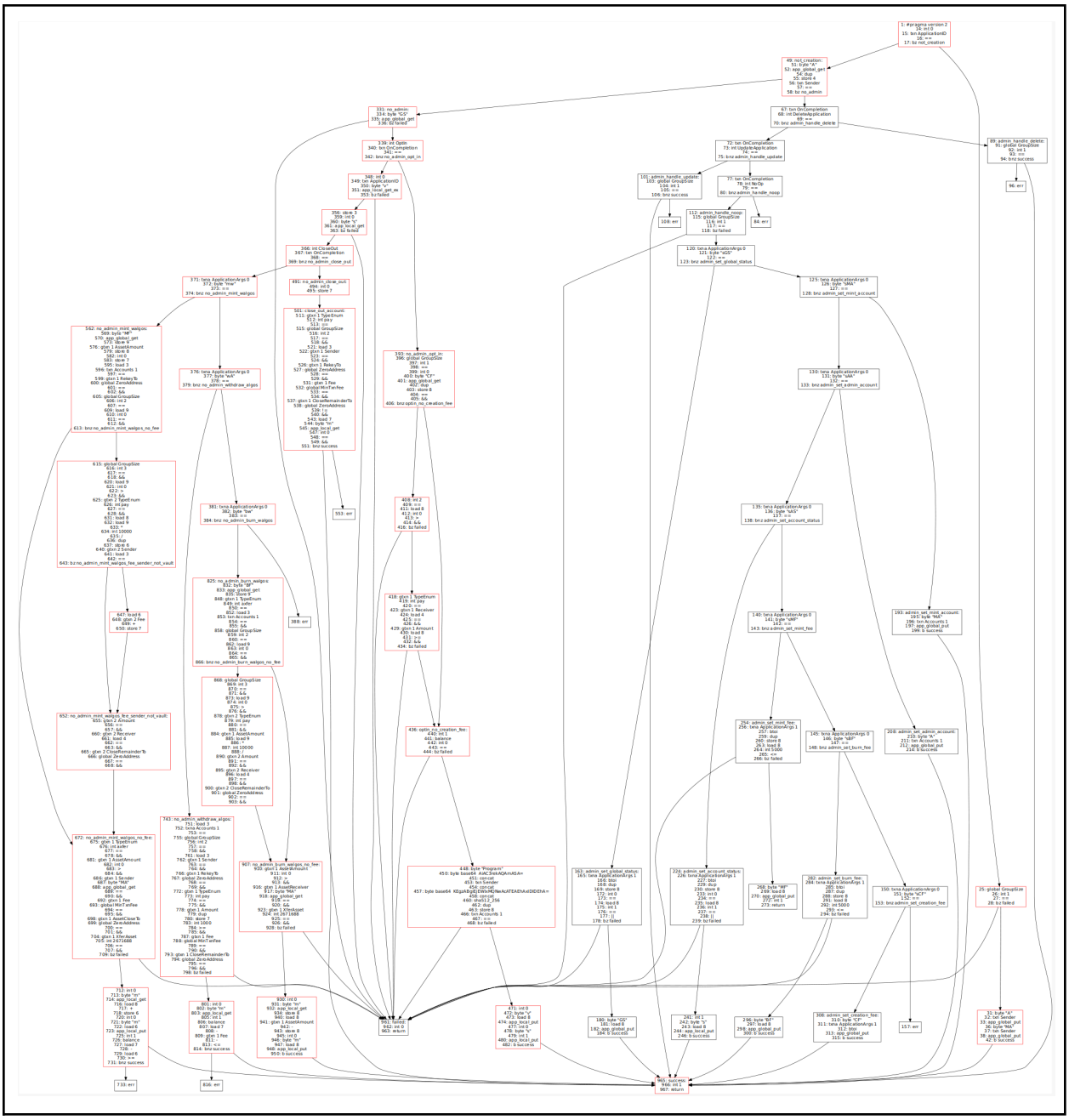

Figure E.1: All paths that can delete the application.

F. Fix Log

StakerDao addressed the issues we found in their codebase. Each of the fixes provided was checked by Trail of Bits on the week of December 7th.

#	Title	Type	Severity	Status
1	Anyone can update or delete the `app-vault`	Access Controls	High	Fixed (98bac113)
2	Lack of clear state program check allows any vault to be drained	Data Validation	High	Fixed (ada7bd86)
3	Missing `RekeyTo` on mint operations allows vault owner to withdraw all the Algo from the vault	Data Validation	High	Fixed (ada7bd86)
4	Missing `RekeyTo` on burn operations allows vault owner to withdraw all the Algo from the vault	Data Validation	High	Fixed (ada7bd86)
5	Minter can be abused to avoid paying the burned wAlgo	Data Validation	High	Fixed (2956621f, 892289bb)
6	Incorrect vault bytecode usage	Patching	High	Fixed (de5c38a)
7	Code does not match	Configuration	High	Fixed (f715c30d, de5c38a)
8	Undocumented privileged operations	Undefined Behavior	High	Fixed (c2400893, 45488fe7)
9	Anyone can burn all the minter's Algo	Access Controls	Medium	Fixed (87246f04)
10	With no fee consideration for burning operations the system is undercollateralized	Data Validation	Medium	Fixed (0ccfa3ad, 0dc93b11)
11	Attackers can prevent a user from opening a vault	Access Controls	Low	Fixed (a121c069)

12	Bad practices for exception handling in the test suite	Error Reporting	Low	Fixed (0771c24c)
13	Insufficient testing coverage	Error Reporting	Low	Fixed (*)
14	Hardcoded `ASA_ID` value is error-prone	Patching	Informational	Fixed (de5c38a)

Further details regarding issues 12 and 13:

12. Bad practices for exception handling in the test suite
The tests were refactored to use Mocha and Chai to catch all exceptions and report them visibly. A beneficial addition would be to catch individual errors instead of `expectTEALReject` to ensure individual function calls fail for the right reason.

13. Insufficient testing coverage
Several tests were added in multiple commits. StakerDao provided the following list:

- Covering each reported issue
- Admins deleting application
- Non-admins deleting application
- Non-admins deleting application after optin
- Non-admins attempting to update application
- Non-admins attempting to update application after optin
- Non-admin attempting to update mint account address
- Non-admin attempting to update mint account address after optin
- Non-admin attempting to set account status
- Non-admin attempting to set account status after optin
- Non-admins attempting to update application
- Non-admins attempting to update application after optin
- Non-admin attempting to update mint account address
- Non-admin attempting to update mint account address after optin
- Minter trying to setMintFee/setCreationFee/setBurnFee
- Added expected state after of the internal variables of the application after calling each test