SOLIDIFIED

Audit Report for Argent. November 26, 2018.

Summary

Audit report prepared by Solidified for Argent covering their modular smart contract wallet and its auxiliary code.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the contracts below. Two severe issues were disclosed by Solidified, prior to the issuance of this report. They are denoted with an asterisk. The debrief took place on November 22, 2018 and the final results are presented here.

AMENDED [08-01-2019]
5 rounds of fix verification were conducted subsequently, the issues found and their fixes are documented below in amendments or denoted with two asterisks (**).

Audited Files

The following files were covered during the audit:

```
contracts
 ├──── base
 │     ├──── Managed.sol
 │     └──── Owned.sol
 ├──── ens
 │     ├──── ArgentENSManager.sol
 │     ├──── ArgentENSResolver.sol
 │     ├──── ENS.sol
 │     └──── ENSConsumer.sol
 ├──── exchange
 │     ├──── ERC20.sol
 │     ├──── KyberNetwork.sol
 │     └──── TokenPriceProvider.sol
 ├──── interfaces
 │     ├──── DBC.sol
 │     └──── Owned.sol
 ├──── modules
```

SOLIDIFIED

Audit Report for Argent. November 26, 2018.

```
│   ├── common
│   │   ├── BaseModule.sol
│   │   ├── OnlyOwnerModule.sol
│   │   └── RelayerModule.sol
│   ├── ApprovedTransfer.sol
│   ├── DappManager.sol
│   ├── GuardianManager.sol
│   ├── LockManager.sol
│   ├── ModuleManager.sol
│   ├── RecoveryManager.sol
│   ├── TokenExchanger.sol
│   └── TokenTransfer.sol
├── storage
│   ├── DappStorage.sol
│   ├── GuardianStorage.sol
│   ├── Storage.sol
│   └── TranferStorage.sol
├── upgrade
│   ├── ModuleRegistry.sol
│   └── SimpleUpgrader.sol
├── utils
│   ├── GuardianUtils.sol
│   ├── SafeMath.sol
│   └── strings.sol
└── wallet
    ├── BaseWallet.sol
    ├── Proxy.sol
    └── WalletFactory.sol
```

Intended Behavior

The purpose of these contracts is to facilitate the creation and management of Argent's modular wallets. A full specification is available here. Of particular note is their security model which consists of "guardians" who are appointed by wallet owners to assist in verifying transactions which don't conform to their configured security checks, as well as locking and recovering the wallet in case of loss or theft.

The audit was based on commit `57a78e104abdb81872ee4fec7a72bc2d196ca9e3`.

AMENDED [08-01-2019]
Final verification was performed on commit `6eafb7a385eb6be4d4950501544192336434733O`.

Issues Found

Critical

*1. Insufficient validation in function `execute` of contract `RelayerModule`

Description
`RelayerModule` contains function of the following signature `execute(BaseWallet _wallet, bytes _data, uint256 _nonce, bytes _signatures, uint256 _gasPrice)` that allows a "relayer" to execute signed transactions on the behalf of a wallet owner. Proper validation is done on the `_wallet` argument regarding signatures; however, there is no validation that the `_wallet` argument and the intended base wallet address encoded in the `_data` argument are equivalent. In short, this means an attacker can call functions on any wallet through this method by encoding the target wallet in the `_data` argument: allowing them to, for example, transfer the assets of said target wallet to themselves.

Recommendation
Validate that the BaseWallet address encoded in `_data` argument is equivalent to the `_wallet` argument. Alternatively, restructure the code to extract the wallet address from the `_data` argument.

Argent's Response:
This issue was addressed prior to commit [`57a78e104abdb81872ee4fec7a72bc2d196ca9e3`] by adding a `verifyData()` method that checks that the input wallet in `execute()` is the same as the first argument encoded in the data (given our modular architecture all methods of modules that change the state of the blockchain have the target wallet as first parameter).

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

Major

*2. ERC20 limits can be bypassed by ETH transfer

Description
ERC20 limits can be bypassed by doing transfers of ETH with data (potentially for 0 ETH) in `TokenTransfer`, that actually calls an ERC20 contract and transfers said tokens directly.

Recommendation
There should be a way to forbid all data containing ETH transactions: many valuable assets are managed in non-standard ways (i.e. not through ERC20 functions). Rather than filtering for `transfer` and `approve`, ETH transfers that intend to invoke functions should likely be handled in `DappManager`; and disallowed completely in the `TokenTransfer` module.

Argent's Response:
The internal method `transferETH()` of `TokenTransfer` has been updated to disable transfers with data. The `TokenTransfer` module can now only instruct the `BaseWallet` to transfer ETH and ERC20 tokens and not call other contracts.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

3. Be wary of ransom by guardian

Description
Guardians (especially in cases where there is only 1 guardian), can continually deny access to an owner's wallet by repeatedly locking it. The guardian could then ask for a (potentially trustless) ransom.

Recommendation
Users should be made aware of this griefing vector. Additionally, consider setting a minimum for amount of guardians.

Argent's Response:
We have addressed that issue by enabling the owner to call the `revokeGuardian()` and `confirmRevokeGuardian()` methods while the wallet is locked. With this modification a compromised guardian denying the owner access to its wallet by locking it repeatedly will be removed by the owner therefore limiting the denial to a maximum of 24 hours. This modification

does not affect the security model of the wallet.

We note that the addition of guardians must be disabled while the wallet is locked and this feature is essential for the security of the recovery procedure of the wallet. To enforce that feature we have slightly updated the locking mechanism to ensure that the wallet cannot be unlocked while a recovery has been started.

The specification has been updated accordingly.

Our Response:
One guardian can be counted as multiple guardians, by changing guardian wallet ownership in between `isGuardian()` calls. I.e. guardian is set to be a contract, the contract returns different value each time `owner()` is called, owner of the contract can now generate arbitrary number of valid signatures, bypassing the limit in `RecoveryManager` etc

Argent's Response:
We updated the logic to make sure that once a signature matches a guardian that guardian is removed from the list of guardians used in the validation of the following signatures.

AMENDED [08-01-2019]
This issue has been addressed to our satisfaction.

4. Dapps can do arbitrary data transactions on behalf of the wallet

---

Description
`DappManager`, allows arbitrary data transactions on users' behalf. Without validation/restricting the possible data transactions, each dapp poses a large potential threat (for the same reasons outlined in recommendation of issue #2).

Recommendation
Dapps need stronger controls on what they can call, else this is a likely vector for bypassing other modules (as demonstrated in issue #2).

Argent's Response:
`DappManager` has been completely refactored to address that issue:
- It is no longer an additional layer on top of `TokenTransfer`. `DappManager` is now completely independant and can call the wallet directly.
- We have introduced a whitelist of contracts and methods per dapp key, i.e. the owner must

authorise a specific key to call a specific method of a specific contract. All calls are disabled by default.
- To simplify the contract while protecting it from abusive dapps draining its ETH we have introduced a single global dapp limit that limits the amount of ETH dapps can spend collectively in a 24 hours period. Dapps no longer have individual limits.
- Since the logic to manage that global limit in `DappManager` and in `TokenTransfer` is identical we have extracted it in a `LimitManager` that both modules inherit.

Our Response:
Compromised owner can bypass daily `TokenTransfer` ERC20 limit by authorizing his malicious dapp contract to transfer ERC20 tokens in `DappManager`. `setAuthorizeCall` should have the same speed bump as everything else.
A potential alternative to the fix proposed for `DappManager` could be turning `TransferManager` into something like `TransactionManager`: a whitelisting functionality similar to that in `DappManager` should be added and `DappManager` would do its calls through the `TransactionManager`. That way, you can get rid of the speedbumps in `DappManager` and leave the protection against owner to the `TransactionManager`.

Argent's Response:
We added an `AuthorisedContractRegistry` that contains contracts and methods that we (Argent) have whitelisted. When the owner calls `setAuthoriseCall()` on the `DappManager` it will take 24 hours by default unless the target contract and methods are whitelisted in the registry. We will encourage dapp developpers to register their dapp with us to unable instant login with our Universal Login SDK.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

**5. Wallet owner can bypass the securityPeriod for limit change

---

Description
`init` in `LimitManager` can be called more than once, allowing the wallet owner to bypass the `securityPeriod` for limit change.

Argent's Response:
We added a check to make sure that the limit (current, pending and changeAfter) is 0 before setting it to the default limit.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

**6. Owner can completely control RecoveryManager when there is only one guardian

---

Description
When there's only one guardian, module can be fully controlled by just owner.

Recommendation
`RecoveryManager` should probably only consider guardian signatures as valid and require half of guardian signatures rounded up.

Argent's Response:
This one is a limitation of our security model and we are ok with it. We are (and will continue) communicating that one guardian only protects the wallet against loss of the owner key and not against theft.

AMENDED [08-01-2019]
This issue has been addressed to our satisfaction.

**7. Daily ETH transfer limit can be bypassed through metatransaction refunds

---

Description
The daily ETH transfer limit can be bypassed through metatransaction refunds, assuming a malicious miner. There should be a gaslimit for metatransactions. Guardians can easily inflate gas consumption, because `owner()` call to their contract is also metered; in combination with gas refunds, they can drain the wallet. In some cases only one guardian's signature is needed, so if the guardian is also a miner, they can drain the funds. These are essentially two separate problems, #1 is bypassing the daily limits and #2 is that guardians can trigger overpriced refunds.

Recommendation
A daily limit on gas refunds mitigates the issues by putting a cap on the potential damage, but does not solve the second. There is a solution in the form of the following trade-off: transactions without owner's approval or guardian majority would not be eligible for refunds, cost of other transactions would have to be covered by the submitter. One added consideration, for this trade-off is that one rogue guardian could DoS relayed transaction by maxing out the daily limit, so they could only be reliably stopped by a non-relayed transaction.

Argent's Response:
We've updated the refund mechanism to disable refund for calls requiring only 0 or 1 signatures, except for `TokenTransfer` and `DappManager` where the refund is counted in the daily limit.
We've also made `GuardianUtils` specify the gas (5000) when calling `BaseWallet.owner()`

Our Response:
Owner should be made to pass gas limit as a parameter for relayed functions, so there's no way for an abuser to increase the gas usage.
An attack could look like this:

1) Owner and/or Guardians sign the transaction.
2) Someone in control of some code that gets called inside the transaction (not necessarily the guardians) flips a "gas consuming switch" that swaps in unnecessary computation: for example, minting gas token for profit.
3) Transaction is now submitted, the signers unaware that a disproportionate amount of gas will be spent by the transaction and refunded.

Each of these attacks individually would be practically limited by the block gas limit, but each transaction is potentially vulnerable to this attack. Put another way: all relayed transactions are effectively submitted with an implicit gas limit equal to the block gas limit (not taking into account daily limits).

Argent's Response:
We've added a gasLimit parameter to the RelayerModule:execute method and propagation of this new parameter to the relaying logic (computation of the signHash and refund).

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

**8. Wallet can be made unrecoverable by compromised Owner

Description
Removing modules can make the wallet unrecoverable in case of malicious owner, because it can remove all modules except, for example, token transfer and then there's no way to recover the ownership of the wallet.

Argent's Response:
We've removed the ModuleManager:removeModule method (and ModuleManager:addModule as it is already present in BaseModule) so that the only way to remove a module is through the ModuleManager:upgrade method. This should eliminate the attack since Upgraders must be registered with Argent.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

Minor

9. No validation that byte arrays are long enough for the reads from memory

---

Description
`mload(p)` reads memory from position p to position p+32 (a word or 32 bytes). This is larger than the expected data (4 bytes for function signature) in the following instances:
`TokenTransfer.sol:L387`
`TokenTransfer.sol:L398`
`RelayerModule.sol:L209`
`RelayerModule.sol:L233`

There are no checks if the byte arrays are long enough for the reads, so the `mload` can read something that is stored after the array in memory, which could also be manipulated by attacker. This might allow an attacker to convince the contract that the data array contains a function signature it does not.

Recommendation
Though it's unclear how an attacker would use this in an exploit, the behavior is unpredictable and therefore undesirable. Ensure that memory isn't read out of bounds.

Argent's Response:
We have added checks to ensure that a byte array contains at least N bytes before we read it with the `mload()` method and store it in a variable of N bytes. Since `mload()` reads words of 32 bytes this does not guarantee that it will not load extra data when e.g. N=4, but the additional bytes are ignored by the logic of the contract. We believe that this is sufficient to mitigate the issue.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

10. Function `setLimit` in `DappManager` can be frontrun (similar to ERC20 approve attack)

Description
Though unlikely, a dapp can front-run calls to this function, and transfer the old value before the limit is changed. When the new limit is set, the dapp can then make another transaction with the new value: total transfer of old value + new value.

Recommendation
The `increaseAllowance`/`decreaseAllowance` pattern found in the OpenZeppelin ERC20 implementation could be adapted here.

Argent's Response:
Following the refactoring of DappManager this issue no longer applies.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

11. Adding executed pending transactions to the whitelist is counterintuitive

---

Location
`TokenTransfer.sol:L272`

Description
When a pending transaction is executed it automatically adds the `to` argument to the transfer whitelist. If the transaction is pending due to the daily limit, it seems to be an unexpected side effect. Additionally, there is no convenient way to cancel this future whitelisting.

Argent's Response:
We have removed the auto-whitelist of recipient when a pending transfer is executed.

AMENDED [08-01-2019]
This issue is no longer present in commit `6eafb7a385eb6be4d4950501544192336434733O`.

Notes, Improvements, & Optimizations

12. Consider adding two-step ownership transfers

Description
A two-step transfer (in which the current owner calls a function that sets a new owner, but that ownership is not transferred until the new owner calls a function to claim ownership) can guarantee that the new owner is capable of interacting with the contract and drastically diminish the possibility of wrong transfers.

Argent's Response:
We believe this issue is naturally mitigated by our model since ownership of a wallet can only be changed through a recovery. Should the new owner be incompatible (e.g. because of a typo in the recovery address) the guardians can launch a new recovery with a new and valid owner.

13. Consider adding methods to recover tokens mistakenly sent to modules

Description
It could prove useful to add a recovery method for tokens mistakenly sent to the various wallet modules.

Argent's Response:
We have added a `recoverToken()` method to the `BaseModule` contract. To avoid introducing some form of ownership to the modules, this method can be called by anyone and will transfer all tokens to the module registry. The same method has been added to `ModuleRegistry` and will transfer all tokens to the registry owner.

14. Misc.

A. In `TokenTransfer`, function `changeLimit` deletes current pending change even if it hasn't been fulfilled. It is unclear whether this is intended behavior

Argent's Response:
This is the intended behaviour.

B. In `DappManager`, the invariant captured by the require on L131 can be broken in `TokenTransfer`

Argent's Response:
No longer applies.

SOLIDIFIED

Audit Report for Argent. November 26, 2018.

Bug Bounty results [12-02-2019]

Minor

1. Failed transactions replay (7 ETH)

Description

In `RelayerModule.execute(...)` there are a few checks which depending on wallet's balance or a number of guardians (L77, L78, L83, L88) might fail or succeed. In case of the failure, the signed transaction data remains in the blockchain and can be replayed at any time later (especially when the amount of required signatures is > 1, because in this case the `nonce` is not checked and there is no time limit for the validity the signature). The signers of such transaction would have no way to cancel it.
For example, the following failed transactions could be replayed at some time later:

- me and my Guardians sign a Token or Ether Transfer transaction. However, we specify too high `_gasLimit` and `verifyRefund(...)` in L78 fails, because the account balance is too low. We sign a new transaction with a lower `_gasLimit` and the transaction succeeds. However, the first failed transaction can be replayed later once the wallet happens to have enough Ether. And we have no way to prevent it.
- or some other signed transaction above fails, because L88 `refund(...)` reverts, because we've forgot to authorize the module in the wallet. Once the module is authorized in the Wallet, such transaction could be replayed.
- or if we submit a signed transaction with a wrong number of Guardian signatures. Once the Wallet's Guardians change (e.g. one is removed), such transaction might be replayed long time after it has been signed.

I thing, to fix these issues, the `RelayerModoule.execute(..)` should be modified to:

- the _singHash should include the number of required signatures
- The `relayer[_wallet].executedTx[_signHash] = true` should be set even if transaction cannot be executed due too low Ether or Token balance (or some other failing condition which in the future might not fail).
- Maybe there should be a function in the Relayer to explicitly invalidate the previously signed and not successfully executed `_signHash`, by setting `relayer[_wallet].executedTx[_signHash] = true`.

Reported by: GundasV

`https://web.solidified.io/contract/5c41eb1ac752390011be4bd9/bug/5c4a2d25c752390011be4c2a`

2. Certain Relayer transactions could be replayed (7 ETH)

Description

Depending on a number of signatures required, the Relayer implements two different mechanisms to guard against transactions replay:

- It validates `_nonce` if a number of signatures required is 1
- It validates `_signHash` if a number of signatures required is > 1

When a number of Wallet Guardians changes, some Modules (e.g. `ApproveTransfer.sol` and `RecoveryManager.sol`) also change the number of signatures required. If number of signatures required changes from 0 to 1 or from 1 to 0, some previously executed transactions could be replayed.
For example `ApproveTransfer.sol` module:

- If a Wallet has single Gaurdian and that Guardian is removed, some previously signed and executed transactions (depending on what `Nonce` has been used) could be replayed by anyone.
- If a Wallet has no Guardians and a Guardian is added, the newly added Guardian can replay all previously executed transfers.

Reported by: GundasV

`https://web.solidified.io/contract/5c41eb1ac752390011be4bd9/bug/5c49a0afc752390011be4c23`

3. Gas stipend for owner() (7 ETH)

Description

When adding a new Guardian with `addGuardian()`, you check for the existence of the owner()-function, while forwarding all gas. I think it's better to apply a gas stipend, e.g. the 5000 gas you used in GuardianUtils. Otherwise, calling `owner()` could potentially cost a lot.

More importantly, however, you allow adding a Guardian that implements any `owner()` function. In contrast, `GuardianUtils.isGuardian()` only qualifies an address as Guardian when `owner()`

returns an address with 5000gas stipend. This breaks the Guardian functionality in case a Guardian's `owner()`-function costs >5000 gas. Because the Guardian addition works, but `isGuardian` would fail.

Reported by: Mo

`https://web.solidified.io/contract/5c41eb1ac752390011be4bd9/bug/5c446ad0c752390011be4bde`

4. A guardian can add (authorize) a module (4 ETH)

Description

Maybe not a direct security threat, but kind of unexpected functionality:

- a single Guardian can add (authorize) a module to his Wallet by calling `addModule(..)` on a `LockManager` contract.

I would suggest moving the `addModule(..)` function from `BaseModule` to the `ModuleManager` contract.

Reported by: GundasV

`https://web.solidified.io/contract/5c50371bc752390011be4c37/bug/5c5175dec752390011be4c39`

Suggestion For Tips

1. TokenTransfer.sol - gas savings suggestion

Description

I would suggest implementing `TokenTransfer.getRequiredSignatures(...)` L373 like it is done in other modules - they check method signature against a pre-computed bytes4 constant, rather than calling the costly `keccak256` on a string constant every time the function is executed.

Reported by: GundasV

`https://web.solidified.io/contract/5c41eb1ac752390011be4bd9/bug/5c498fc6c752390011be4c1f`

2. type revokation->revocation

Description

As the title says, I think it should be `recovation` instead of `revokation`

Reported by: Mo

`https://web.solidified.io/contract/5c41eb1ac752390011be4bd9/bug/5c47785cc752390011be4c0e`

3. Function is not consistent with comments

Description

The init function does nothing. Comments say that it should log an event and also that it should only be called by the wallet itself but there are no access restrictions.

Reported by: EtherPaul

`https://web.solidified.io/contract/5c41eb1ac752390011be4bd9/bug/5c42350fc752390011be4bdb`

4. A few Top-100 ERC20s do not implement decimals()

Description

The init function does nothing. Comments say that it should log an event and also that it should only be called by the wallet itself but there are no access restrictions.

Reported by: GundasV

`https://web.solidified.io/contract/5c50371bc752390011be4c37/bug/5c518b99c752390011be4c3f`

Audit Report for Argent. November 26, 2018.

Closing Summary - AMENDED [08-01-2019]

Argent has addressed all discovered issues and suggestions, including bug bounty .

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Argent's products. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.