Ambire Security Review

Pashov Audit Group

Conducted by: pashov

September 25th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Ambire** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Ambire

Copied from the first security review

Ambire is a smart wallet protocol. Users have wallets (accounts) which are controlled by them or other addresses that have "privileges" to do so. A user can do an off-chain signature of a bundle of transactions and anyone can execute it on-chain. Different signature schemes are allowed, for example EIP712, Schnorr, Multisig and others. The protocol works in a counterfactual manner, meaning a user wallet gets deployed only on its first transaction. The actual deployment is an EIP1167 minimal proxy for the wallet smart contract.

Continued

The `Ambire` protocol extended its signature validator options by adding an "external signature validator" option. One such option is the `DKIMRecoverySigValidator`, which is basically a way to recover access to your smart wallet by using your email. In the case that you have access & control over your secondary key and your email but you lost your primary key, you can instantly recover access to your account. If you have lost access/control over either of them you can still queue a recovery but you'd have to wait for a timelock to pass.

Observations

The data that the `DKIMRecoverySigValidator` is using is canonized by the front end service.

Part of the DKIM validating logic (DNSSEC) is forked from the ENS application on Ethereum mainnet.

`DKIMRecoverySigValidator` is deployed as a singleton and `Ambire` controls the values of `authorizedToSubmit` and `authorizedToRevoke`.

For email providers that do not support DNSSEC a "bridge" functionality is added, which basically means that Ambire supports a "DNSSEC bridge" domain that verifies authenticity of the DNS records. This is a centralization tradeoff based on the fact that for example Gmail does not support DNSSEC.

Privileged Roles & Actors

- External Signature Validator - a smart contract that has a special approach to verifying signatures (for example DKIM)
- DNSSEC Oracle - verifies that an array of signed DNS records represents a chain of trust
- `authorizedToSubmit` - address that can add new DKIM keys used for recovery
- `authorizedToRevoke` - address that can remove DKIM keys used for recovery

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **- 8521c4e68b147c0002b2e2c337178e4f5d23d594**

fixes review commit hash **- 20aae8ec666d1341dc1462962ea2ac50bf2edd6f**

Scope

The following smart contracts were in scope of the audit:

- `AmbireAccount`
- `DKIMRecoverySigValidator`

7. Executive Summary

Over the course of the security review, pashov engaged with Ambire to review Ambire. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Ambire
Date	September 25th, 2023

Findings Count

Severity	**Amount**
Critical	3
High	1
Medium	1
Low	2
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Recoveries of type SigMode.OnlyDKIM can be reverted by anyone	Critical	Resolved
[C-02]	Recoveries can be blocked by front-running	Critical	Resolved
[C-03]	Anyone can block single signature DKIM recoveries	Critical	Resolved
[H-01]	Recoveries of type SigMode.OnlySecond can be endlessly replayed	High	Resolved
[M-01]	Insufficient input validation	Medium	Resolved
[L-01]	The dateRemoved field of a removed DKIM key can be reset	Low	Resolved
[L-02]	Validate the to email as you do for the from one	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Recoveries of type `SigMode.OnlyDKIM` can be reverted by anyone

Severity

Impact: High, as the recovery will be reverted and the user still won't be able to gain access to his smart wallet

Likelihood: High, as anyone can execute this attack when `SigMode.OnlyDKIM` is used for recovery

Description

The way in which the recoveries in `DKIMRecoverySigValidator::validateSig` work are that when a recovery is executed, the `sigMeta.newAddressToSet` address will be granted `sigMeta.newPrivilegeValue` privileges. In `AmbireAccount` when an address has non-zero privileges he is allowed to execute anything in the smart wallet, making him an owner of it. The problem with this is that the `sigMeta.newPrivilegeValue` is not validated.

If a user uses a recovery of type `SigMode.OnlyDKIM` and sends a recovery transaction to set address "X" to have privilege of "1", now a timelock will be set, after which the user can send the same transaction again and the "X" address will have a privilege of "1". Now it is possible that a malicious attacker uses the same transaction payload that the user used but just by changing the `data` of the first `Transaction` in the `calls` argument in `DKIMRecoverySigValidator::validateSig` and setting the `sigMeta.newPrivilegeValue` to 0, now a new recovery will be started which will set the "X" address so that it doesn't have privileges anymore. This will work as the `identifier` for the recovery will be different, since `sigMeta` is part of the `identifier` calculation and is now different.

There are variations of this attack - the malicious user can back-run the normal user's `SigMode.OnlyDKIM` transaction, so he will be able to execute his timelock just after the user does.

Here is a Github gist link to a Proof of Concept unit test to see the attack in action (you can put it in the `'DKIM sigMode OnlyDKIM'` suite in `DKIMTest.ts` to run it).

Recommendations

For recoveries only allow `sigMeta.newPrivilegeValue == 1` or remove `sigMeta.newPrivilegeValue` from the `identifier` construction.

[C-02] Recoveries can be blocked by front-running

Severity

Impact: High, as recovery functionality won't be usable

Likelihood: High, as there are no preconditions to executing such an attack on all recoveries

Description

Each call from `AmbireAccount` to `DKIMRecoverySigValidator::validateSig` can be front-runned by calling the method with the same arguments, which will set the value in the `recoveries` mapping for the recovery payload to `true`, making the actual transaction revert due to this check:

```
require(!recoveries[identifier], 'recovery already done');
```

Here is a Github gist link to a Proof of Concept unit test to see the attack in action (you can put it in the `'DKIM sigMode Both'` suite in `DKIMTest.ts` to run it).

Recommendations

In `validateSig` just use `msg.sender` instead of the `accountAddress` parameter. This will prevent anyone from sending the `accountAddress` to be

the `AmbireAccount` one.

[C-03] Anyone can block single signature DKIM recoveries

Severity

Impact: High, as the user won't be able to recover access to his wallet

Likelihood: High, as anyone can execute this attack by front-running a recovery

Description

In `DKIMRecoverySigValidator` there is a mechanism for single signature recoveries. The catch is that if you use it, you will have to have a timelock on your recovery, which is a predefined number (should be set in `AccInfo.onlyOneSigTimelock`). Until a timelock expires the wallet access can't be recovered. The problem here is that the `checkTimelock` method that actually sets a lock for an `identifier` (which is built by hashing some recovery data) is `public`, so anyone can call it. This means that anyone can front-run a `validateSig` call with a `checkTimelock` one, setting a huge `time` until unlock of the recovery, basically blocking it.

Here is a Github gist link to a Proof of Concept unit test to see the attack in action (you can put it in the `'DKIM sigMode OnlySecond with a timelock of 2 minutes'` suite in `DKIMTest.ts` to run it).

Recommendations

Change the `checkTimelock` method from `public` to `private` so it can only be called internally by `validateSig`, or just inline the method.

8.2. High Findings

[H-01] Recoveries of type `SigMode.OnlySecond` can be endlessly replayed

Severity

Impact: High, as it can result in a loss of wallet access

Likelihood: Medium, as it requires specific conditions but they can happen even after years of using the wallet

Description

The `recoveries` mapping in `DKIMRecoverySigValidator` is serving as a way to check if a `recovery` has been executed already, so it is a way to block recovery replays. The problem with this is that the key in that mapping is calculated from fields in a recovery payload that can be slightly changed (for example by adding a random character to a `string` or `bytes` typed field, examples are `SignatureMeta.canonizedHeaders` or `SignatureMeta.key.pubKeyModulus`) so you can execute a recovery payload where all other values are the same.

The concrete problem is here:

```
bytes32 identifier = keccak256(abi.encode(accountAddr, data, sigMeta));
```

The `identifier` which is used as a key in the `recoveries` mapping, is using `SignatureMeta sigMeta` which has the `string canonizedHeaders` field. When `SigMode.OnlySecond` is used, the `canonizedHeaders` has no validation whatsoever, so it can be slightly changed and the `identifier` would be different, while all other values are the same and still valid. This means that all `SigMode.OnlySecond` recoveries can be replayed endlessly - as many times as desired, forever.

The problem is deflated by the fact that the `SignatureMeta.newKeyToSet` value can't be changed (it is validated when `SigMode.onlySecond` is used), so all of the mentioned replays will end up granting access to the same `newKeyToSet` address every time they are executed. Still, let's look at the following scenario:

1. Alice lost access to her `AmbireAccount` so she executes a recovery using `DKIMRecoverySigValidator` with `SigMode.OnlySecond`, which gives control over her `AmbireAccount` to the address she set in `SignatureMeta.newKeyToSet`
2. A year passes by, Alice rotates her keys and now throws away the old ones
3. Bob replays her previous recovery by slightly changing the `canonizedHeaders` field which results in that same address (which Alice now doesn't control) to control her `AmbireAccount`
4. Now Alice has lost control over her wallet and can't recover it, since the recovery is configured to work only for the previously set `SignatureMeta.newKeyToSet`

Recommendations

When crafting the `identifier` key for the `recoveries` mapping, possibly use only the fields from `SignatureMeta` that are validated in all code paths, so that recovery replays are not possible.

8.3. Medium Findings

[M-01] Insufficient input validation

Severity

Impact: High, as it can result in forever blocked recoveries or wallet access loss

Likelihood: Low, as it requires user error when configuring their recovery

Description

The `DKIMRecoverySigValidator` contract uses the `AccInfo` struct that has fields which values are not validated. Here are the fields that are problematic:

- `secondaryKey` can be the zero address, which can result in a loss of access to the `AmbireAccount` wallet
- `waitUntilAcceptAdded` is a time value and can be too big, resulting in inability to add DKIM keys
- `waitUntilAcceptRemoved` is a time value and can be too big, resulting in inability to remove DKIM keys
- `onlyOneSigTimelock` is a time value and can be too big, resulting in forever locked recovery

Recommendations

Add sensible upper boundaries for the time values and a check that `secondaryKey != address(0)`.

8.4. Low Findings

[L-01] The `dateRemoved` field of a removed DKIM key can be reset

The `removeDKIMKey` method in `DKIMRecoverySigValidator` does not check if the given key has already been removed. The `authorizedToRevoke` account can call the method for the same key as many times as he wants, which will reset the `dateRemoved` property to `block.timestamp` each time.

[L-02] Validate the `to` email as you do for the `from` one

In `DKIMRecoverySigValidator::_verifyHeaders` the `from` mail is validated in a way that protects from the email that looks like "fake@gmail.com@gmail.com". The check is not done for the `to` mail and can be added by slicing the string up until the special character that is expected to be after the email, and comparing this to `toHeader`.