Clave

Security Review

Cantina Managed review by:
Riley Holterhus, Lead Security Researcher
Blockdev, Security Researcher

March 20, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Clave is an easy-to-use non-custodial smart wallet powered by Account Abstraction and the Hardware Elements (e.g Secure Enclave, Android Trustzone etc...), offering a unique onboarding process.

From Feb 21st to Feb 27th the Cantina team conducted a review of clave-contracts on commit hash 4bb00991. The team identified a total of **14** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 5
- Gas Optimizations: 2
- Informational: 6

3 Findings

3.1 Medium Risk

3.1.1 Consider restricting users' gas sponsorship usage

Severity: Medium Risk

Context: GaslessPaymaster.sol#L71-L73

Description: In the `GaslessPaymaster` contract, accounts that are registered in the `claveRegistry` will receive free gas sponsorships on `userLimit` number of transactions. Each of these sponsored transactions can have arbitrary `gasLimit` and `maxFeePerGas` values, so users can potentially spend a large amount of the paymaster's ETH in each of their sponsored transactions. With enough unreasonably expensive transactions, the paymaster's balance could be depleted quickly, which would ruin the experience for honest users.

Recommendation: Consider altering the paymaster to allocate users a certain amount of gas instead of a certain amount of transactions. Alternatively, consider adding upper bounds on the `gasLimit` and `maxFeePerGas` values, so that users can only use their sponsored transactions reasonably.

Clave: Fixed in commit c84a3e5f.

Cantina Managed: Verified. A `MAX_SPONSORED_ETH` constant was added to address the issue, and each sponsored transaction must now use less than this amount.

3.2 Low Risk

3.2.1 Passkey validation missing verification steps recommended by WebAuthn standard

Severity: Low Risk

Context: PasskeyValidatorConstant.sol#L42-L74, PasskeyValidator.sol#L51-L84

Description: The WebAuthn standard specifies several steps for verifying an authentication assertion. Some of these steps are optional or irrelevant to the `PasskeyValidator` contract. However, some steps are important to fully capitalize on the enhanced security offered by secure authenticator enclaves, but are currently skipped by the `PasskeyValidator`. Specifically, the following two steps from the spec could be added to the `_validateFatSignature()` logic:

> 14. Verify that the User Present bit of the flags in authData is set.

The User Present (UP) flag indicates the authenticator is asserting the user is physically present for the authentication process, which is an appropriate assertion to verify.

> 17. If user verification is required for this assertion, verify that the User Verified bit of the flags in authData is set.

The User Verification (UV) flag signifies the authenticator has conducted a verification process (e.g., fingerprint scan, facial recognition) to confirm the user's identity. The authenticator may or may not verify the user, depending on three options in the request: "required", "preferred", or "discouraged". Since Clave currently has this option set to "required" (meaning the authenticator *should* perform verification), it would be appropriate to validate this in the `PasskeyValidator`.

Recommendation: In the `_validateFatSignature()` function, check the value of these two flag bits in the `authenticatorData` provided by the user. Note that this is already done implicitly in the `_validateSignature()` function, since the hardcoded `AUTHENTICATOR_DATA` has these flags set.

Clave: Fixed in commit 8128b154.

Cantina Managed: Verified. The `_validateFatSignature()` function now uses the `AUTH_DATA_MASK` to ensure that the UP and UV flags are both set.

3.2.2 PasskeyValidator vulnerable to signature malleability

Severity: Low Risk

Context: PasskeyValidator.sol#L76, PasskeyValidator.sol#L99, PasskeyValidatorConstant.sol#L67, PasskeyValidatorConstant.sol#L90

Description: Since PasskeyValidator verifies an ECDSA signature, `N-s` (`N` being the curve order) is also valid as a signature. Ethereum rejects signature for the higher `s` for secp256k1 in `ecrecover`. Accepting the higher `s` shouldn't pose a security risk if you are using a nonce in the message for protection against signature replay. Even then, you may want to be consistent with the general practice followed by blockchains (Bitcoin and Ethereum).

Recommendation: Assert that `s` is not greater than `N/2`, and on the mobile app, make sure the signature produced doesn't include the higher `s`. If it does, flip it to `N-s`.

Clave: Fixed in commit ad43511c.

Cantina Managed: The `s` max threshold is set for secp256k1. It needs to be set to `0x7fffffff800000007fffffffffffffffde737d56d38bcf4279dce5617e3192a8` for secp256r1 curve.

Clave: Fixed in commit 28a4e5ae.

Cantina Managed: Verified.

3.2.3 `hookData` can be arbitrarily extended

Severity: Low Risk

Context: ClaveImplementation.sol#L185-L187, ClaveImplementation.sol#L252-L256

Description: This issue was raised by the Clave team before the audit. Adding here for tracking purposes.

In the `executeTransactionFromOutside()` and `validateTransaction()` control flows, the `hookData` array in the transaction's signature is decoded and passed to the `runValidationHooks()` function. This function calls each of the account's validation hooks with the corresponding `hookData` element so the hook can validate the transaction.

With this logic, nothing is preventing a signature from providing *too much* `hookData`, which will ultimately go unused during execution. This implies that a griefer can arbitrarily extend the `hookData` component of an honest user's signature, which will increase the transaction's gas costs while keeping everything else the same. This would force transactions to use a higher percentage of their specified `gasLimit`, which can be an inconvenience and waste of funds.

Recommendation: Prevent this griefing possibility by ensuring that the `hookData` length matches the number of validation hooks in the account. This can be accomplished with the following change in `runValidationHooks()`:

```
  function runValidationHooks(
      bytes32 signedHash,
      Transaction calldata transaction,
      bytes[] memory hookData
  ) internal returns (bool) {
      mapping(address => address) storage validationHooks = _validationHooksLinkedList();

      address cursor = validationHooks[AddressLinkedList.SENTINEL_ADDRESS];
      uint256 idx = 0;
      // Iterate through hooks
      while (cursor > AddressLinkedList.SENTINEL_ADDRESS) {
          // Call it with corresponding hookData
          bool success = _call(
              cursor,
              abi.encodeWithSelector(
                  IValidationHook.validationHook.selector,
                  signedHash,
                  transaction,
                  hookData[idx++]
              )
          );

          if (!success) {
              return false;
          }

          cursor = validationHooks[cursor];
      }

+     if (hookData.length != idx) return false;

      return true;
  }
```

Clave: Fixed in commit 150c9cce.

Cantina Managed: Verified.

3.2.4 `Base64Url` encoding can be incorrect on inputs with length not divisible by 3

Severity: Low Risk

Context: Base64Url.sol#L62-L91

Description: In the `Base64Url` contract, the `encode()` function has the following `for` loop:

```
for {
    let dataPtr := data
    let endPtr := add(data, dataLen)
} lt(dataPtr, endPtr) {

} {
    // Advance 3 bytes
    dataPtr := add(dataPtr, 3)
    let input := mload(dataPtr)

    mstore8(resultPtr, mload(add(tablePtr, and(shr(18, input), 0x3F))))
    resultPtr := add(resultPtr, 1) // Advance

    mstore8(resultPtr, mload(add(tablePtr, and(shr(12, input), 0x3F))))
    resultPtr := add(resultPtr, 1) // Advance

    mstore8(resultPtr, mload(add(tablePtr, and(shr(6, input), 0x3F))))
    resultPtr := add(resultPtr, 1) // Advance

    mstore8(resultPtr, mload(add(tablePtr, and(input, 0x3F))))
    resultPtr := add(resultPtr, 1) // Advance
}
```

Each iteration of this `for` loop will read a chunk of 3 bytes and will write exactly 4 bytes of output to the `result`. There is an issue with this logic when the input length is not a multiple of 3. In this case, the final iteration of the loop will still treat the next input as a full 3 bytes, but in reality, it is only 1 or 2 bytes. This means this function can read ***past*** the memory of the input data.

For example, if the input length is 32 bytes, then the last iteration has 2 bytes (16 bits) of actual input and has 1 byte (8 bits) of random unrelated memory for the remaining data. If `x` represents one bit of input data and `y` represents one bit of the following unrelated memory, the last iteration of the loop will interpret the memory like this:

```
// xxxxxx xxxxxx xxxxyy yyyyyy
// char1  char2  char3  char4
```

This is incorrect, since the third character is using unrelated memory when it should be using zero-padding bits, and the fourth character shouldn't even exist. In fact, this is a problem in the OpenZeppelin library this code was forked from, which can be demonstrated with the following proof of concept:

```
pragma solidity 0.8.18;

import "openzeppelin-contracts/contracts/utils/Base64.sol";

contract POC {
    event LogString(string);

    struct ExampleStruct {
        bytes32 h;
        uint256 a;
    }

    function run() external {
        bytes memory inputData1 = abi.encodePacked(keccak256("OpenZeppelin Proof of Concept"));
        bytes memory inputData2 = abi.encodePacked(keccak256("OpenZeppelin Proof of Concept"));
        ExampleStruct memory example = ExampleStruct(keccak256("ExampleHash"), 12345);
        emit LogString(Base64.encode(inputData1)); // emits
↪  LogString("8cAFkAAC3Mt3P/TN95g8+Gddy5c8Lp8XOGlFfzJ1/FA=")
        emit LogString(Base64.encode(inputData2)); // emits
↪  LogString("8cAFkAAC3Mt3P/TN95g8+Gddy5c8Lp8XOGlFfzJ1/FD=")
    }
}
```

Fortunately, in the current Clave codebase, it doesn't seem possible for this problem to happen, due to the specifics of when each memory value is allocated. However, consider fixing this problem for future usage of the `Base64Url` library.

Recommendation: Change the `for` loop logic to account for a final iteration that may not include 3 input bytes:

```
for {
    let dataPtr := data
    let endPtr := add(data, dataLen)
} lt(dataPtr, endPtr) {

} {
    let remaining := sub(endPtr, dataPtr) // How many bytes are still remaining in the input
    dataPtr := add(dataPtr, 3) // Advance 3 bytes
    let input := mload(dataPtr)
    let mask := 0x3f // Mask to apply to the group of 6 input bits

    mstore8(resultPtr, mload(add(tablePtr, and(shr(18, input), mask))))
    resultPtr := add(resultPtr, 1) // Advance

    if eq(remaining, 1) { mask := 0x30 } // If there was 1 byte remaining, use 0b110000 here

    mstore8(resultPtr, mload(add(tablePtr, and(shr(12, input), mask))))
    resultPtr := add(resultPtr, 1) // Advance

    if eq(remaining, 1) { break } // Done now if 1 byte was remaining
    if eq(remaining, 2) { mask := 0x3c } // If there was 2 bytes remaining, use 0b111100 here

    mstore8(resultPtr, mload(add(tablePtr, and(shr(6, input), mask))))
    resultPtr := add(resultPtr, 1) // Advance

    if eq(remaining, 2) { break } // Done now if 2 bytes were remaining

    mstore8(resultPtr, mload(add(tablePtr, and(input, mask))))
    resultPtr := add(resultPtr, 1) // Advance
}
```

Alternatively, consider waiting for OpenZeppelin to address this issue in their codebase, and update to their new contract when ready.

Clave: Fixed in commit 856b439d and commit 7b5a66f2.

Cantina Managed: Verified. The most recent version of OpenZeppelin's code has been added. The OpenZeppelin team changed their implementation to cache and temporarily zero out the relevant dirty bytes, which indeed fixes the issue. One final technical detail worth noting is that the last iteration of the `for` loop may still write past the memory allocated for the `result` string. This is acceptable behavior, as the `result` string is the last value to be allocated, so this would simply write past the current free memory pointer.

3.2.5 Incorrect magic value check

Severity: Low Risk

Context: ClaveImplementation.sol#L79-L81

Description: The `initialize()` function in the `ClaveImplementation` contract has added two new arguments: `initCall` and `signature`. These arguments allow the user to combine their account deployment with an immediate execution, so long as the `signature` is successfully validated:

```
bytes32 signedHash = keccak256(abi.encode(initCall));
bytes memory signatureAndValidator = abi.encode(signature, initialR1Validator);
bytes4 magicValue = isValidSignature(signedHash, signatureAndValidator);

if (magicValue == ACCOUNT_VALIDATION_SUCCESS_MAGIC) {
    // ...
}
```

However, notice that the `magicValue` returned from `isValidSignature()` is being checked against the `ACCOUNT_VALIDATION_SUCCESS_MAGIC` variable. This is incorrect since the `isValidSignature()` function returns the magic value `_ERC1271_MAGIC`, which is a value distinct from `ACCOUNT_VALIDATION_SUCCESS_MAGIC`.

As a result, the new `initCode` functionality can never be reached, since the validation will always be considered failing.

Recommendation: Verify the `magicValue` in `initialize()` against the `_ERC1271_MAGIC` value:

```
- if (magicValue == ACCOUNT_VALIDATION_SUCCESS_MAGIC) {
+ if (magicValue == _ERC1271_MAGIC) {
```

Clave: Solved in commit 3f3a7721 with the removal of signature validation in `initialize()`.

Cantina Managed: Verified. Signature verification has been removed from `initialize()`, which resolves this issue.

3.3 Gas Optimization

3.3.1 Remove `if` condition in favor of early return

Severity: Gas Optimization

Context: VerifierCaller.sol#L38

Description: This `if` condition can be removed in favor of early return, as when this condition fails, the function always returns `false`:

```
if (returnValue == 1) return true;
```

Recommendation: Benchmark and update as follows:

```
-if (returnValue == 1) return true;
+return returnValue == 1;
```

Clave: Fixed with commit 2f810b76.

Cantina Managed: Verified.

3.3.2 Unnecessary variable assignment

Severity: Gas Optimization

Context: GaslessPaymaster.sol#L85-L86

Description: The `GaslessPaymaster` contract does not make use of the `context` variable that can be returned in `validateAndPayForPaymasterTransaction()` to be later used in `postTransaction()`. This is explicitly documented in a section of the code that also sets the `context` to empty bytes:

```
// No context needed
context = bytes('');
```

Note that `context` is a named return variable that is otherwise never assigned, so this code snippet is unnecessary.

Recommendation: Consider removing the unnecessary assignment:

```
+ // No context needed (already empty bytes as a named return value)
- // No context needed
- context = bytes('');
```

Clave: Fixed with commit 864eb763.

Cantina Managed: Verified.

3.4 Informational

3.4.1 Consider the `AccountFactory` trust assumptions

Severity: Informational

Context: AccountFactory.sol

Description: In the `AccountFactory`, only the privileged `_deployer` address can deploy accounts. Also, this address decides the `initializer` bytes that are used, meaning they are trusted to choose appropriate `initialR1Owner` and `initialR1Validator` values during account creation.

On the other hand, in the new `initialize()` logic, the `initCall` data is verified with a `signature` before proceeding with execution. Since the `_deployer` address is already trusted to choose the `initialR1Owner`, this signature may not be necessary.

Recommendation: Consider the trust assumptions of the `_deployer` role in the `AccountFactory`. Since the `_deployer` is already trusted to an extent, it might be possible to remove the signature verification for the `initCall`. Alternatively, consider the possibility of reducing the trust assumptions of the `_deployer`, perhaps in a future iteration of the code.

Clave: Fixed in commits 2208be7e and d18cccd2 by removing the signature and refactoring to use `_executeCall()`.

Cantina Managed: Verified.

3.4.2 Missing NatSpec

Severity: Informational

Context: ClaveImplementation.sol#L53-L58

Description: In the newest version of the `ClaveImplementation` contract, two new arguments have been added to the `initialize()` function. These two arguments have not yet been added to the function's NatSpec, which would help with further documentation.

Recommendation: Consider adding the two new arguments into the `initialize()` function's NatSpec:

```
  /**
   * @notice Initializer function for the account contract
   * @param initialR1Owner bytes calldata - The initial r1 owner of the account
   * @param initialR1Validator address    - The initial r1 validator of the account
   * @param modules bytes[] calldata      - The list of modules to enable for the account
+  * @param initCall Call calldata        - The initial call to make during account deployment
+  * @param signature bytes calldata      - The signature required to validate the initial call information
   */
  function initialize(
      bytes calldata initialR1Owner,
      address initialR1Validator,
      bytes[] calldata modules,
      Call calldata initCall,
      bytes calldata signature
  ) external initializer {
```

Clave: Fixed in commit dcab330a.

Cantina Managed: Verified. Note that the `signature` was removed in an unrelated change, so this commit only needs to add the `initCall` to the NatSpec.

3.4.3 Consider checking `type(IERC777Receiver).interfaceId` in `supportsInterface()`

Severity: Informational

Context: TokenCallbackHandler.sol#L55-L60

Description: The `TokenCallbackHandler` contract implements the ERC777 function `tokensReceived()` and also inherits from the `IERC777Recipient` interface. However, the `TokenCallbackHandler` does not have a check for the `IERC777Recipient` interface in its EIP165 `supportsInterface()` function.

Recommendation: Consider adding a check for the `type(IERC777Recipient).interfaceId` in the `supportsInterface()` function in the `TokenCallbackHandler`:

```
  function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
      return
          interfaceId == type(IERC721Receiver).interfaceId ||
          interfaceId == type(IERC1155Receiver).interfaceId ||
+         interfaceId == type(IERC777Recipient).interfaceId ||
          interfaceId == type(IERC165).interfaceId;
  }
```

Clave: Removed ERC-777 interface in commits 43f24c88, 054936e2 and d6294970.

Cantina Managed: Verified.

3.4.4 Indicate the reason for two `PasskeyValidator` files

Severity: Informational

Context: PasskeyValidator.sol, PasskeyValidatorConstant.sol

Description: The above two files are duplicates except one uses an immutable variable for `P256_VERIFIER` and the other uses a hardcoded value. As per Clave team, this is due to a bug in zksync that prevents using immutable variables in account validation. Immutable version is to be used in tests and constant one is for being deployment.

Recommendation: Indicate this difference in comments. Consider deleting `PasskeyValidator.sol` and just test with `PasskeyValidatorConstant.sol`. If you are having issues with the P256 verifier at the specified constant address, consider using `vm.etch` foundry cheatcode to set the verifier bytecode at the required address.

Clave: Fixed in commit dcb1dc26.

Cantina Managed: Verified. The relevant files have been moved to the `test/` directory.

3.4.5 Assignments within ternary operands

Severity: Informational

Context: GaslessPaymaster.sol#L112

Description: You can rewrite the following in a more canonical way:

```
userLimit > sponsored ? limit = (userLimit - sponsored) : limit = 0;
```

Recommendation: Consider updating this to:

```
-userLimit > sponsored ? limit = (userLimit - sponsored) : limit = 0;
+limit = userLimit > sponsored ? (userLimit - sponsored) : 0;
```

Clave: Fixed with commit a402e8e3.

Cantina Managed: Verified.

3.4.6 Incorrect require statements

Severity: Informational

Context: GaslessPaymaster.sol#L150, GaslessPaymaster.sol#L165

Description: In the `GaslessPaymaster` contract, the `addLimitlessAddresses()` and `removeLimitlessAddresses()` functions have the following require statements:

```
// In addLimitlessAddresses():
require(addr != address(0) || !limitlessAddresses[addr]);

// In removeLimitlessAddresses():
require(addr != address(0) || limitlessAddresses[addr]);
```

These require statements intend to enforce that the `addr` in question is non-zero and has the appropriate `limitlessAddresses` boolean. However, the statement incorrectly uses `||` instead of `&&`, meaning this sanity check can succeed in scenarios where it shouldn't. Fortunately, this is a non-critical check, especially because these functions are behind the `onlyOwner` modifier.

Recommendation: Change `||` into `&&` in the above require statements.

Clave: Fixed with commit b9b07f48.

Cantina Managed: Verified.