Waterusdc and Vaultka Solana Programs

Vaultka

HALBORN

Waterusdc and Vaultka Solana Programs - Vaultka

Prepared by: HALBORN

Last Updated 08/26/2024

Date of Engagement by: July 22nd, 2024 - August 19th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
9	1	1	2	3	2

TABLE OF CONTENTS

1. Introduction

Vaultka engaged Halborn to conduct a security assessment on their **Waterusdc and Vaultka Solana programs** beginning on July 22, 2024, and ending on August 19, 2024. The security assessment was scoped to the Solana Programs provided in vaultkarust GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

The Waterusdc program is a single side lending pool program that allows whitelisted users or other programs to borrow USDC tokens from the lending pool. The Vaultkausdc program is a strategy program allowing users to borrow USDC tokens via the Waterusdc program and gain exposure to the Jupiter's JLP token. The program uses the Pyth oracle in order to fetch current JLP and USDC prices, and also allows users to use leverage up to certain limits to increase market exposure.

2. Assessment Summary

Halborn was provided 4 weeks for the engagement and assigned one full-time security engineer to review the security of the Solana Programs in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the Solana Programs.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security concerns that were addressed by Vaultka team. The main ones were the following:

- Withdraw fee is transferred to the user instead of the fee vault
- Inefficient slippage control
- Incorrect token price conversion prevent withdrawal
- Incorrect accounts mutability

Only informational issues were currently only acknowledged and not addressed.

3. Test Approach And Methodology

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.

- Manual program source code review to identify business logic issues.

- Mapping out possible attack vectors

- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.

- Scanning dependencies for known vulnerabilities (ċargo audit).

- Local anchor testing (ànchor test)

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: vaultkarust

(b) Assessed Commit ID: 6e81ea1

(c) Items in scope:

- /vaultkarust/blob/fixes-tests/Anchor.toml
- /vaultkarust/blob/fixes-tests/Cargo.toml
- /vaultkarust/blob/fixes-tests/programs/vaultkausdc/Cargo.toml
- /vaultkarust/blob/fixes-tests/programs/vaultkausdc/Xargo.toml
- /vaultkarust/blob/fixes-tests/programs/vaultkausdc/src/lib.rs
- /vaultkarust/blob/fixes-tests/programs/waterusdc/Cargo.toml
- /vaultkarust/blob/fixes-tests/programs/waterusdc/Xargo.toml
- /vaultkarust/blob/fixes-tests/programs/waterusdc/src/lib.rs

Out-of-Scope: /vaultkarust/blob/fixes-tests/programs/watersol/Cargo.toml, /vaultkarust/blob/fixes-tests/programs/watersol/Xargo.toml, /vaultkarust/blob/fixes-tests/programs/watersol/src/lib.rs, /vaultkarust/blob/fixes-tests/migrations/deployUSDCStrategy.js, /vaultkarust/blob/fixes-tests/migrations/deployWaterUSDC.js, /vaultkarust/blob/fixes-tests/programs/vaultkasol/Cargo.toml, /vaultkarust/blob/fixes-tests/programs/vaultkasol/Xargo.toml, /vaultkarust/blob/fixes-tests/programs/vaultkasol/src/lib.rs, /vaultkarust/blob/fixes-tests/package.json, /vaultkarust/blob/fixes-tests/tsconfig.json

REMEDIATION COMMIT ID:

- 3d3ea42
- c089107
- 6481b3d
- fc34d4a

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

1 1 2 3 2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
WITHDRAW FEE IS TRANSFERRED TO THE USER INSTEAD OF THE FEE VAULT	CRITICAL	SOLVED - 08/21/2024
INEFFICIENT SLIPPAGE CONTROL	HIGH	SOLVED - 08/26/2024
INCORRECT ACCOUNTS MUTABILITY	MEDIUM	SOLVED - 08/23/2024
INCORRECT TOKEN PRICE CONVERSION PREVENTS WITHDRAWAL	MEDIUM	SOLVED - 08/21/2024
RISK OF OUTDATED PRICE FEED	LOW	SOLVED - 08/21/2024
SET_JLP_PRICE INSTRUCTION WILL ALWAYS FAIL	LOW	SOLVED - 08/21/2024
UNBOUNDED FEES CALCULATION	LOW	SOLVED - 08/21/2024
INABILITY TO CLOSE UNNEEDED ACCOUNTS	INFORMATIONAL	ACKNOWLEDGED
FORMAL ISSUES AND RECOMMENDATIONS	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 WITHDRAW FEE IS TRANSFERRED TO THE USER INSTEAD OF THE FEE VAULT

// CRITICAL

Description

The `Withdraw` instruction of the `Waterusdc` program allows users to withdraw liquidity from the lending program for a predefined fee.
However, instead of transferring the fee to the fee vault as intended, the instruction mistakenly returns the fee to the user.
As a result, the protocol is unable to collect fees from users.

waterusdc/src/lib.rs

```
640 let fee_transfer_accounts = TransferChecked {
641             from: ctx.accounts.usdc_token_account.to_account_info(),
642             to: ctx.accounts.user_ata.to_account_info(),
643             authority: ctx.accounts.program_authority.to_account_info(),
644             mint: ctx.accounts.usdc_mint.to_account_info(),
645         };
646 let fee_amount_context = CpiContext::new_with_signer(
647             ctx.accounts.token_program.to_account_info(),
648             fee_transfer_accounts,
649             signer_seeds,
650         );
651 anchor_spl::token::transfer_checked(fee_amount_context, w_fee, 6)?;
```

Proof of Concept

1. Initialize waterusdc program
2. Deposit USDC tokens
3. Withdraw USDC tokens

```ts
it("Withdraw", async () => {
  const withdrawAmount = 500_000;

  const userInfos = PublicKey.findProgramAddressSync([Buffer.from("USER_INFOS"), user.publicKey.toBuffer()], program.programId)[0]

  const feeBalanceBefore = (await provider.connection.getTokenAccountBalance(withdraw_fee_receiver_ata)).value.uiAmount;

  await program.methods
  .withdraw(new anchor.BN(withdrawAmount))
  .accountsStrict({
    user: user.publicKey,
    userInfos,
    lendingAccount: deployer_lending_account,
    systemProgram: SystemProgram.programId,
    withdrawFeeReceiver: withdraw_fee_receiver.publicKey,
    withdrawFeeReceiverAta: withdraw_fee_receiver_ata,
    usdcTokenAccount: vaultUsdc,
    userAta: userAta,
    tokenProgram: TOKEN_PROGRAM_ID,
    programAuthority: programAuthority,
    usdcMint: usdcMint,
  })
  .signers([user])
  .rpc();

  let vaultBalance = await provider.connection.getTokenAccountBalance(vaultUsdc);
  const feeBalanceAfter = (await provider.connection.getTokenAccountBalance(withdraw_fee_receiver_ata)).value.uiAmount;

  const deployer_lending_account_data = await program.account.lending.fetch(
    deployer_lending_account
  );

  assert.ok(deployer_lending_account_data.vaultUsdcBalance.eq(new anchor.BN(0)));
  expect(vaultBalance.value.amount).to.eq("0");
  expect(feeBalanceAfter).to.greaterThan(feeBalanceBefore, "Fee balance on fee account did not increase!");

});
```

```
Transaction executed in slot 16:
  Signature: 3W7cYjsp115kq3u1UrEFKgmextsYhYBy4tAahceJcTUkNuykcUjywGjbu29qDEo275DNug1Dhzr2ht9onrsfr62R
  Status: Ok
  Log Messages:
    Program 2PtJ5jHaqGJ9b7ekk3LgBkEizfGtystsgUTgpRumFxgM invoke [1]
    Program log: Instruction: Withdraw
    Program log: Max withdraw: 500000
    Program log: Lending balance: 500000
    Program log: User shares estimated: 500000
    Program log: Withdraw fee: 25000
    Program log: Withdraw amount with fee: 475000
    Program log: Program authority bump seed: 253
    Program log: Program authority: 4QDQnX4GBH6DqUsqQohaezsUgWoyKoHxXxWt8WnDoshe
    Program log: Authority seeds: [[97, 117, 116, 104, 111, 114, 105, 116, 121], [253]]
    Program log: Signer seeds: [[[97, 117, 116, 104, 111, 114, 105, 116, 121], [253]]]
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 148649 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 139741 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program data: FgmFGqAsR8BFAVn+8Bex46WtIFVFE0073JjUrKn5IjsQ4L8z+xOBvROzAVVp23R6gGfNGwaIcPPdVIEMn0rMwIA+0fbjhyOdIKEHAAAAAAAAAAAAAAAAAAM2gvGYAAAAA
    Program 2PtJ5jHaqGJ9b7ekk3LgBkEizfGtystsgUTgpRumFxgM consumed 68403 of 200000 compute units
    Program 2PtJ5jHaqGJ9b7ekk3LgBkEizfGtystsgUTgpRumFxgM success
```

```
1) lending_usdc
     Withdraw:

    Fee balance on fee account did not increase!
    + expected - actual


    at /Users/adam/Work/Audits/Vaultka/vaultkarust-test/tests/lending_usdc.ts:226:32
    at Generator.next (<anonymous>)
    at fulfilled (tests/lending_usdc.ts:28:58)
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:C/R:N/S:U (10.0)

Recommendation

To address this issue, it is recommended to correct the destination address so that the fee is sent to the fee vault account.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by modifying the destination address and setting it to the correct fee vault account address.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/3d3ea42c829469c87f9af464b4604337054916d0

7.2 INEFFICIENT SLIPPAGE CONTROL

// HIGH

Description

Both the `ExecuteDeposit` and `ExecuteWithdraw` instructions ensure that the received amount falls within the slippage range defined by the expected amount and slippage percentage, verifying that there hasn't been a significant exchange rate change between the request and execution.
However, the `RequestDeposit` and `RequestWithdraw` instructions do not allow users to set slippage limits. In volatile markets, outdated Pyth oracle prices, or incorrect manual JLP price settings, users might request a deposit or withdrawal that results in an unexpected output amount.
Depending on the slippage control settings in the `ExecuteDeposit` or `ExecuteWithdraw` instructions, the execution might fail, causing funds to become stuck, or it might succeed, with the user receiving less than expected.
In addition to the issue above, the amount in the `ExecuteDeposit` or `ExecuteWithdraw` instructions is verified only using the slippage control. However, the slippage control parameter can be adjusted by the admin user without any bounds. Invoking the `ExecuteDeposit` instruction with an incorrect amount and large slippage range can result in loses either for the user or the protocol.

Proof of Concept

1. Initialize the vaultkausdc and waterusdc programs
2. Set strategy parameters
3. Manually set the JLP token price to twice of the current price to simulate incorrect/outdated/volatile price
4. Request deposit
5. Set back the JLP token price to the current price
6. Execute deposit
7. Manually set the JLP token price to twice of the current price to simulate incorrect/outdated/volatile price
8. Set back the JLP token price to the current price
9. Execute deposit

Deposit request with incorrect price:

```
Transaction executed in slot 33:
  Signature: 2X5svHY4Z68P6cFCgzhZvFZmo4HbfFZuTx8fpAJBwNh8YfbgWRvfMxdA1oqnmE7KcdRuu6pr2V5xziyEEwQNdGtL
  Status: Ok
  Log Messages:
    Program ComputeBudget111111111111111111111111111111 invoke [1]
    Program ComputeBudget111111111111111111111111111111 success
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: RequestDeposit
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program log: Depositing 2000000 lamports to strategy
    Program log: strategy.leverage_limit: 5
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 437146 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: Mapping Before: 0
    Program log: The price is (99995001 ± 95151) * 10^-8
    Program log: The price is (313070122 ± 4471181) * 10^-8
    Program log: Manual JLP price: 650000000
    Program log: Deposited 2000000 lamports to strategy
    Program log: JLP price: 6500000000
    Program log: Borrow amount: 4000000
    Program log: JLP price in USDC: 6500324
    Program log: Expected amount out JLP: 923030
    Program log: CPI program: 2PtJ5jHaqGJ9b7ekk3LgBkEizfGtystsgUTgpRumFxgM
    Program log: Program authority bump seed: 255
    Program log: Program authority: 3SGdMmfrJkfbwXbWHmc29FoRA3pKrdbmTS7vn34vG8WV
    Program log: Authority seeds: [[97, 117, 116, 104, 111, 114, 105, 116, 121], [255]]
    Program log: Signer seeds: [[[97, 117, 116, 104, 111, 114, 105, 116, 121], [255]]]
    Program 2PtJ5jHaqGJ9b7ekk3LgBkEizfGtystsgUTgpRumFxgM invoke [2]
    Program log: Instruction: Borrow
    Program log: Borrow amount: 4000000
    Program log: Borrowed amount: 0
    Program log: Total assets: 5000000
    Program log: Utilization rate: 0
    Program log: Lending balance: 5000000
    Program log: Borrow amount: 4000000
    Program log: Program authority bump seed: 253
    Program log: Program authority: 4QDQnX4GBH6DqUsqQohaezsUgWoyKoHxXxWt8WnDoshe
    Program log: Authority seeds: [[97, 117, 116, 104, 111, 114, 105, 116, 121], [253]]
    Program log: Signer seeds: [[[97, 117, 116, 104, 111, 114, 105, 116, 121], [253]]]
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [3]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 335090 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
```

Execute deposit with correct price fails and with incorrect price completes successfully:

```
Transaction executed in slot 36:
  Signature: YrTeDobgkN4J2KsbRhdG2UBKcF8CuozMjh7Go7z7n2qynvZ5WjyVxQV5K7kKZVuRmE1gszDvrdXbpFXuZa68xrq
  Status: Error processing Instruction 0: custom program error: 0x10
  Log Messages:
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: ExecuteDeposit
    Program log: Executing deposit for position ID: 0
    Program log: Fetching balance before swap
    Program log: JLP token account balance before swap: 0
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 169617 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: JLP token account balance after swap: 1846046
    Program log: Received amount: 1846046
    Program log: Expected amount: 923030
    Program log: Slippage amount: 92303
    Program log: Min amount: 830727
    Program log: Max amount: 1015333
    Program log: ProgramError occurred. Error Code: Custom(16). Error Number: 16. Error Message: Custom program error: 0x10.
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 consumed 43084 of 200000 compute units
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 failed: custom program error: 0x10
Transaction executed in slot 38:
  Signature: 5KP9dDx8mvDyEUjYdGJ9S2Xrr9Zi1QhFdd1eDD17oNywtAkvHLGMutESH1ZUiEzRQ8JFx5ojJuVttf6JiNksy747
  Status: Ok
  Log Messages:
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: ExecuteDeposit
    Program log: Executing deposit for position ID: 0
    Program log: Fetching balance before swap
    Program log: JLP token account balance before swap: 0
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 169617 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: JLP token account balance after swap: 923023
    Program log: Received amount: 923023
    Program log: Expected amount: 923030
    Program log: Slippage amount: 92303
    Program log: Min amount: 830727
    Program log: Max amount: 1015333
    Program data: NNAvwMZXANXZ5qzLJ2rniIeNL3qHpfTrhc6l034DWEGoQVPj15F5wAAAAAAAAAAAjxUOAAAAAAAc2r1mAAAAAA==
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 consumed 44454 of 200000 compute units
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 success
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:M/D:N/Y:H/R:N/S:U (8.8)

Recommendation

To address this issue, it is recommended to implement slippage control also for the `RequestDeposit` and `RequestWithdraw` instructions and limit the allowed range of slippage.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by adding a new input parameter to the `RequestDeposit`, `RequestIncreaseCollateral` and `RequestWithdraw` instructions so that the user is able to set the minimal output amount that he or she can accept. If the calculated output amount is less than the expected minimum, the instruction fails and the user has to repeat the request.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/c0891071a0e7ef95a78f1855fafde2189d264af0

7.3 INCORRECT ACCOUNTS MUTABILITY

// MEDIUM

Description

The `vaultkausdc` program implements manually the `AccountsMeta` structures, that are used during CPI (cross-program invocation) of `Borrow` and `Repay` instructions of the `waterusdc` program.
However, these `AccountsMeta` structs are implemented incorrectly and the `token_program` and `usdc_mint` accounts are marked as mutable instead of read-only.
This prevents the `RequestDeposit` and the `ExecuteWithdraw` instructions from being invoked due to cross-program invocation error.
vaultkausdc/src/lib.rs

```
109 impl<'info> ToAccountMetas for Borrow<'info> {
110     fn to_account_metas(&self, _is_signer: Option<bool>) -> Vec<AccountMeta>
111         vec![
112             AccountMeta::new(*self.borrower.key, true),
113             AccountMeta::new(*self.vault.key, false),
114             AccountMeta::new(*self.whitelisted_borrower.key, false),
115             AccountMeta::new(*self.program_authority.key, false),
116             AccountMeta::new(*self.receiver_ata.key, false),
117             AccountMeta::new(*self.usdc_token_account.key, false),
118             AccountMeta::new(*self.token_program.key, false),
119             AccountMeta::new(*self.usdc_mint.key, false),
120         ]
121     }
122 }
```

```
151 impl<'info> ToAccountMetas for Repay<'info> {
152     fn to_account_metas(&self, _is_signer: Option<bool>) -> Vec<AccountMeta>
153         vec![
154             AccountMeta::new(*self.borrower.key, true),
155             AccountMeta::new(*self.vault.key, false),
156             AccountMeta::new(*self.whitelisted_borrower.key, false),
157             AccountMeta::new(*self.program_authority_lending.key, false),
158             AccountMeta::new(*self.usdc_token_account.key, false),
159             AccountMeta::new(*self.borrower_ata.key, false),
160             AccountMeta::new(*self.token_program.key, false),
161             AccountMeta::new(*self.usdc_mint.key, false)
162         ]
163     }
164 }
```

In addition to this issue, the context structures `Deposit`, `ExecuteDeposit`, `RequestIncreaseCollateral`, `RequestWithdraw` and `ExecuteWithdraw` of the `vaultkausdc` program do not set correctly the mutability attribute of their accounts (some accounts are missing the `mut` attribute and are considered as read-only instead of mutable). It does not create a vulnerability in the program, however it does not allow the Anchor Typescript client to correctly generate the `AccountMetas` for the corresponding instruction. Therefore it is impossible to invoke these instructions using the generated Anchor Typescript client.

Proof of Concept

1. Initialize the vaultkausdc and waterusdc programs
2. Set strategy parameters
3. Request deposit
4. Execute deposit
5. Request increase collateral
6. Execute deposit
7. Request withdrawal
8. Execute withdrawal

```
Transaction executed in slot 31:
  Signature: 22nF27EbjQumXLZNKqZcnCXCweTz23xtuQdB3XPB2wjefDeH5G4uLa4qRvD8z5ASZa6P65x7puyFpvwrX52Z47Eg
  Status: Error processing Instruction 1: Cross-program invocation with unauthorized signer or writable account
  Log Messages:
    Program ComputeBudget111111111111111111111111111111 invoke [1]
    Program ComputeBudget111111111111111111111111111111 success
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: RequestDeposit
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program log: Depositing 2000000 lamports to strategy
    Program log: strategy.leverage_limit: 15
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 437134 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: Mapping Before: 0
    Program log: The price is (99993129 ± 98143) * 10^-8
    Program log: The price is (317623616 ± 630022) * 10^-8
    Program log: Manual JLP price: 325000000
    Program log: Deposited 2000000 lamports to strategy
    Program log: JLP price: 3250000000
    Program log: Borrow amount: 4000000
    Program log: JLP price in USDC: 3250223
    Program log: Expected amount out JLP: 1846027
    Program log: CPI program: 2PtJ5jHaqGJ9b7ekk3LgBkEizfGtystsgUTgpRumFxgM
    Program log: Program authority bump seed: 255
    Program log: Program authority: 3SGdMmfrJkfbwXbWHmc29FoRA3pKrdbmTS7vn34vG8WV
    Program log: Authority seeds: [[97, 117, 116, 104, 111, 114, 105, 116, 121], [255]]
    Program log: Signer seeds: [[[97, 117, 116, 104, 111, 114, 105, 116, 121], [255]]]
    TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA's writable privilege escalated
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 consumed 94967 of 482350 compute units
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 failed: Cross-program invocation with unauthorized signer or writable account
```

```
Transaction executed in slot 40:
  Signature: 4GU8rMqxq7FBdq7JMUd6BUqvtFwQkFTmfZ63E7jgA8y3m8nFpbMh3L1Rwpf1Whx8J92yKnq7M5d95Z2u4o7bNg9
  Status: Error processing Instruction 0: Cross-program invocation with unauthorized signer or writable account
  Log Messages:
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: ExecuteWithdraw
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 157019 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: Slippage amount: 8500757
    Program log: Min amount: 76506813
    Program log: Max amount: 93508327
    Program log: fixed_fee_split: 250
    Program log: leverage: 3
    Program log: Fixed split leverage: 750
    Program log: Fixed split adjusted: 2500
    Program log: Split: 3250
    Program log: To water: 25026435
    Program log: M fee: 11550662
    Program log: To user: 40427319
    TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA's writable privilege escalated
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 consumed 61590 of 200000 compute units
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 failed: Cross-program invocation with unauthorized signer or writable account
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:H/R:P/S:U (5.9)

Recommendation

To address this issue, it is recommended to set the accounts `token_program` and `usdc_mint` as read-only. Also review the mutability attribute for each account with the `Deposit`, `ExecuteDeposit`, `RequestIncreaseCollateral`, `RequestWithdraw` and `ExecuteWithdraw` instruction context structs.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by correctly setting the `token_program` and `usdc_mint` accounts as read-only and by correctly setting the mutability attribute to the corresponding accounts.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/6481b3d7efd15ebf36c7ff4b537f69ab237d717a

7.4 INCORRECT TOKEN PRICE CONVERSION PREVENTS WITHDRAWAL

// MEDIUM

Description

The `RequestWithdraw` instruction allows users to initiate withdrawal and close their positions. The instruction calculates, among other calculations also the current price of JLP tokens in USDC tokens based on the price of JLP and USDC tokens in USD.
However, the price of JLP tokens is incorrectly scaled and is 10 times greater than it is supposed to be (it is multiplied by 10 twice instead of only once). This results in the JLP/USDC price being 10 times greater, and consequently also the resulting expected amount of USDC tokens is 10 times greater. This prevents the invocation of the `ExecuteWithdraw` instruction, because the received amount of USDC tokens will not be within the accepted range of allowed slippage.
vaultkausdc/src/lib.rs

```
1327  let usdc_price = get_pyth_price_helper(pyth_price_account_usdc, strategy.usdc
1328  let mut jlp_price = 0;
1329  if price_control.is_manual_price_update{
1330      jlp_price = price_control.jlp_price * 10; // here is the JLP price multip
1331      msg!("Manual JLP price: {}", jlp_price);
1332  } else {
1333      jlp_price = get_pyth_price_helper(pyth_price_account_jlp, strategy.jlp_so
1334      msg!("Pyth JLP price: {}", jlp_price);
1335  }
```

```
1396  let mut jlp_p: u128 = jlp_price as u128;
1397  jlp_p = jlp_p * 10; // here is the JLP price multiplied by 10 for the second
1398  msg!("JLP price: {}", jlp_p);
1399  let mut usdc_p: u128 = usdc_price as u128;
1400  usdc_p = usdc_p * 10;
1401  msg!("USDC price: {}", usdc_p);
1402  let jlp_price_usdc: u128 = jlp_p.checked_mul(10u128.pow(6)).ok_or(ErrorCode::
1403                      .checked_div(usdc_p).ok_or(ErrorCode::ReceivedAmountOpera
1404  msg!("JLP price in USDC: {}", jlp_price_usdc);
1405
1406  pos_amount = pos_amount.checked_mul(10u64.pow(3)).ok_or(ErrorCode::ReceivedAm
1407  let pa_a = pos_amount as u128;
1408  let expected_amount_out_usdc: u128 = pa_a.checked_mul(jlp_price_usdc).ok_or(E
1409      .checked_div(10u128.pow(9)).ok_or(ErrorCode::ReceivedAmountOperationError
```

Proof of Concept

1. Initialize the vaultkausdc and waterusdc programs

2. Set strategy parameters
3. Request deposit
4. Execute deposit
5. Request withdrawal
6. Manually execute withdrawal

```
Transaction executed in slot 38:
  Signature: jufxP1YCGgXsGYzVr5FpvKc5tyoS88hBG32WraAaN8Jhw4SgM6YnW63jMyGH1tgBragfmE9woeywxa3TKya8rea
  Status: Ok
  Log Messages:
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: RequestWithdraw
    Program 11111111111111111111111111111111 invoke [2]
    Program 11111111111111111111111111111111 success
    Program log: The price is (99992713 ± 83746) * 10^-8
    Program log: Manual JLP price: 3250000000
    Program log: Position JLP amount: 2615338000
    Program log: Position borrowed amount: 4000000
    Program log: Position JLP value: 8499848500
    Program log: Debt-to-Value ratio: 470596000
    Program log: Debt-to-Value ratio: 470596000
    Program log: Mapping Before: 2615338
    Program log: Program authority bump seed: 255
    Program log: Program authority: 3SGdMmfrJkfbwXbWHmc29FoRA3pKrdbmTS7vn34vG8WV
    Program log: Authority seeds: [[97, 117, 116, 104, 111, 114, 105, 116, 121], [255]]
    Program log: Signer seeds: [[[97, 117, 116, 104, 111, 114, 105, 116, 121], [255]]]
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 137228 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: JLP price: 32500000000
    Program log: USDC price: 999927130
    Program log: JLP price in USDC: 32502368
    Program log: Expected amount out USDC: 85004678
    Program data: FgmFGqAsR8DtLvHMNoaux3mLO6Jx3hRhRTBjdiroRM2lIlnxyJpmhAAAAAAAAAAAEOTimwAAAAA=
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 consumed 75123 of 200000 compute units
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 success
Transaction executed in slot 40:
  Signature: 5EgRVAWqKKj9cA9Qz2e8Si6Lry8g3QQfJNJDxsWSekUUHYqUin4uAPSvUTYtp48o5xqXX3JMRWgLJusns4egDwwC
  Status: Error processing Instruction 0: custom program error: 0x10
  Log Messages:
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 invoke [1]
    Program log: Instruction: ExecuteWithdraw
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA invoke [2]
    Program log: Instruction: TransferChecked
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA consumed 6174 of 157019 compute units
    Program TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA success
    Program log: Slippage amount: 8500467
    Program log: Min amount: 76504211
    Program log: Max amount: 93505145
    Program log: ProgramError occurred. Error Code: Custom(16). Error Number: 16. Error Message: Custom program error: 0x10.
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 consumed 54280 of 200000 compute units
    Program Gy6GvYiMCFf3KtmpQCapHTANtUYhLexwPB52uzkRixT1 failed: custom program error: 0x10
```

```
#[msg("Slippage error")]
SlippageError, //16
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:P/S:U (5.0)

Recommendation

To address this issue, it is recommended to correct the calculation of the JLP/USDC price pair.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by removing the superfluous multiplication of the the JLP token price and thus corrected the price conversion.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/3d3ea42c829469c87f9af464b4604337054916d0

7.5 RISK OF OUTDATED PRICE FEED

// LOW

Description

The program relies on the Pyth oracle to retrieve the current USD prices of JLP and USDC tokens. However, the maximum age for the price feed update is currently set to 120,000 seconds (approximately 33.3 hours). If prices become outdated, such as during a Pyth oracle outage, and market volatility is high, both the user and the protocol risk losses due to incorrect exchange rates.

waterusdc/src/lib.rs

```
1662 pub fn get_pyth_price_helper<'info>(price_account: &Account<'info, PriceUpdat
1663     let price_update = price_account;
1664     // get_price_no_older_than will fail if the price update is more than 30
1665     let maximum_age: u64 = 120000;
1666     // get_price_no_older_than will fail if the price update is for a differe
1667     // This string is the id of the WIF/USD feed (close to JLP price). See ht
1668     //let feed_id: [u8; 32] = get_feed_id_from_hex("0x4ca4beeca86f0d164160323
1669     let price = price_update.get_price_no_older_than(&Clock::get()?, maximum_
1670     // Sample output:
1671     // The price is (7160106530699 ± 5129162301) * 10^-8
1672     msg!("The price is ({} ± {}) * 10^{}", price.price, price.conf, price.exp
1673     let final_price = price.price as u64;
1674
1675     Ok(final_price)
1676 }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:M/R:P/S:U (2.5)

Recommendation

To address this issue, it is recommended to reduce the maximum age parameter.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by decreasing the maximum age parameter to 120 seconds and thus reducing the risk of outdated price feed.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/3d3ea42c829469c87f9af464b4604337054916d0

7.6 SET_JLP_PRICE INSTRUCTION WILL ALWAYS FAIL

// LOW

Description

The `SetJlpPrice` instruction is intended to manually set the USD price of the JLP token by the admin and supposedly also by the authorized keeper.
However, the access control condition is flawed, as it always evaluates to true, leading to an `IncorrectOwner` error. Specifically, the code is checking whether the `user.key` is not equal to both `admin.admin` and `strategy.keeper` at the same time. This creates a situation where the condition will always be true, even if the `user.key` matches one of the two addresses, because a single `user.key` cannot simultaneously match both `admin.admin` and `strategy.keeper`.
waterusdc/src/lib.rs

```
873 // Set the JLP Price manually
874 pub fn set_jlp_price(ctx: Context<SetJLPPrice>, jlp_price: u64) -> ProgramRes
875     let strategy = &mut ctx.accounts.strategy;
876     let admin = &ctx.accounts.admin;
877     let price_control = &mut ctx.accounts.price_control;
878 
879     if *ctx.accounts.user.key != admin.admin || *ctx.accounts.user.key != str
880         return Err(ErrorCode::IncorrectOwner.into());
881     }
882 
883     //MUST BE IN DECIMAL 8
884     //ensure the new JLP price is not 0
885     if jlp_price == 0 {
886         return Err(ErrorCode::InvalidSetterData.into());
887     }
888     let min_jlp_price = price_control.jlp_price - (price_control.jlp_price *
889     let max_jlp_price = price_control.jlp_price + (price_control.jlp_price *
890 
891     if jlp_price < min_jlp_price || jlp_price > max_jlp_price {
892         return Err(ErrorCode::SlippageError.into());
893     }
894 
895     price_control.jlp_price = jlp_price;
896 
897     Ok(())
898 }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:F/S:U (2.5)

Recommendation

To address this issue, it is recommended to correct the access control condition. Either to authorize only one defined account (admin or keeper) or change the logical operator from `OR` to `AND`.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by modifying the logical operator from `OR` to `AND` and thus authorizing the admin or keeper accounts to invoke the `setJlpPrice` instruction.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/3d3ea42c829469c87f9af464b4604337054916d0

7.7 UNBOUNDED FEES CALCULATION

// LOW

Description

During the `ExecuteWithdraw` instruction, the program deducts various fees from the user's profit during the withdrawal process. The amount of fees depends on the following parameters:

- leverage
- fixed_fee_split
- mfee_percent

Although these parameters can be set within predefined limits, the resulting fees are not limited and certain parameter combinations may result in fees equal to or exceeding 100% of the profit. If the fees are less than or equal to 100% of the profit, the `ExecuteWithdraw` instruction will succeed, but the user risks receiving very little or no profit at all. If the fees exceed 100% of the profit, the `ExecuteWithdraw` instruction will fail due to overflow by subtraction, preventing the user from withdrawing their position.
waterusdc/src/lib.rs

```
1800 fn get_profit_split(profit: u64, leverage: u64, fixed_fee_split: u64, mfee_pe
1801     msg!("fixed_fee_split: {}", fixed_fee_split);
1802     msg!("leverage: {}", leverage);
1803     let fixed_split_leverage = fixed_fee_split * leverage;
1804     msg!("Fixed split leverage: {}", fixed_split_leverage);
1805     let fixed_split_adjusted = fixed_fee_split * 10;
1806     msg!("Fixed split adjusted: {}", fixed_split_adjusted);
1807 
1808     let split = fixed_split_leverage.checked_add(fixed_split_adjusted).ok_or(
1809     msg!("Split: {}", split);
1810     let to_water = profit.checked_mul(split).ok_or(ErrorCode::ReceivedAmountO
1811                         .checked_div(10000).ok_or(ErrorCode::ReceivedAmountOp
1812     msg!("To water: {}", to_water);
1813     let m_fee = profit.checked_mul(mfee_percent).ok_or(ErrorCode::ReceivedAmo
1814                     .checked_div(10000).ok_or(ErrorCode::ReceivedAmountOperat
1815     msg!("M fee: {}", m_fee);
1816     let to_user = profit.checked_sub(to_water.checked_add(m_fee).ok_or(ErrorC
1817     msg!("To user: {}", to_user);
1818 
1819     Ok((to_water, m_fee, to_user))
1820 }
```

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:H/R:N/S:U (2.4)

Recommendation

To address this issue, it is recommended to cap the fees to ensure they do not exceed a maximum percentage of the profit.

Remediation Progress

SOLVED: The **Vaultka team** solved the issue by limiting the maximal percentage of total fees and thus ensuring that the fees will always be less than 100 percent of the profit. This guarantees that a user always receives part of the profit and that no arithmetic issues can occur.

Remediation Hash

https://github.com/Vaultka-Project/vaultkarust/commit/fc34d4ac68097cb70b67058587c73dfdda547f69

7.8 INABILITY TO CLOSE UNNEEDED ACCOUNTS

// INFORMATIONAL

Description

The program lacks functionality to close unnecessary `PositionInfo` and `UserInfo` accounts, resulting in rent lamports being permanently locked.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:F/S:U (0.6)

Recommendation

To address this issue, it is recommended to close the unneeded accounts and sent the rent back to the initializer.

Remediation Progress

ACKNOWLEDGED: The client acknowledged the finding.

7.9 FORMAL ISSUES AND RECOMMENDATIONS

// INFORMATIONAL

Description

The project is written in Rust and utilizes the Anchor framework to improve code clarity and security. While the program generally makes effective use of Rust and Anchor, some areas could be enhanced to better align with common Solana development conventions and best software engineering practices. Additionally, numerous formal issues, such as incorrect naming, unused variables, and inaccurate comments, need to be addressed.

BVSS

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:F/S:U (0.1)

Recommendation

To address this issue, it is recommended to implement following actions:

- Do not redefine accounts and instruction data structs (such as lending::cpi::accounts::Repay) for CPI calls.
  - Use automatically generated code from Anchor. Follow the corresponding Anchor CPI documentation.
- Avoid recalculating PDAs and corresponding bumps when not necessary.
  - Use `ctx.bumps.program_authority` instead of `Pubkey::find_program_address(...)` .
- To reload accounts after CPI completion, use the `reload()` method.
  - For example: `ctx.accounts.jlp_token_account.reload()?;`
- Avoid using `msg!()` macro if not necessary.
- Avoid truncating downcast.

```
// instead of truncating downcast
let variable_u64 = variable_u128 as u64;
// prefer using
let variable_u64 = u64::try_from(variable_u128).map_err(...)?;
```

- Add programs as last accounts in the context struct.
- Preferably perform the access control checks in the context struct to separate it from the business logic.

```
// For example in set_vault_keeper, instead of
if ctx.accounts.user.key() != vault_data.owner {}
// use rather the has_one attribute
#[account(mut, has_one = owner, seeds = [b"LENDING".as_ref()], bump)]
```

```
pub lending_account: Account<'info, Lending>,
#[account(mut)]
pub owner: Signer<'info>, // rename user to owner

// Instead of passing unchecked AccountInfo
pub lending_program: AccountInfo<'info>,
// prefer using dedicated Anchor types such as
pub lending_program: Program<‘info, LendingProgram>
```

- Avoid separate variables declaration and initialization.
- Define seeds as constants and reuse them across accounts.
- Modularize the program and split it in multiple files.
- Do not use magic numbers.
- Avoid passing unnecessary (unused) accounts to instructions.
  - The System program in `SetVaultKeeper`, `SetFeeParams` and `SetMaxUtilRate` instructions is not needed.
  - The strategy account in `ManualPriceUpdate`, `SetSlippageControl`, `SetJLPPriceSlippage` instructions is not needed.
- Remove the invalid `#[instruction(to_whitelist: Pubkey)]` macro annotation from `SetVaultKeeper` struct.
- Use more descriptive names for accounts.
  - For example the account `user` in SetFeeParams, SetWhitelisted, SetMaxUtilRate, SetVaultKeeper or DisableWhitelisted instruction should rather be `owner` .
- Return correct errors:
  - After `checked_div` operations, only division by zero error can happen and not overflow or underflow.
  - Review all errors returned.
  - *waterusdc/src/lib.rs*

```
740 pub fn repay(ctx: Context<Repay>, repay_amount: u64, debt_value: u64) -> Re
741         let ra:u128;
742         ra = repay_amount as u128;
743         let dv:u128;
744         dv = debt_value as u128;
745
746         if ra <= 0 {
747             return err!(ErrorCode::InvalidRepayAmount);
748         }
749
750         if dv <= 0 {
751             return err!(ErrorCode::InvalidRepayAmount); // FIX incorrect er
752         }
753
754         let whitelisted = &ctx.accounts.whitelisted_borrower;
755         if !whitelisted.whitelisted {
```

```
            return err!(ErrorCode::InvalidBorrowAmount); // FIX incorrect e
        }
// ...
}
```

- Rename incorrect variable names:
  - *waterusdc/src/lib.rs*

```
pub fn set_strategy_params(
        ctx: Context<SetStrategyParams>,
        dtv_limit: u64,
        keeper: Pubkey,
        keeper_ata: Pubkey,
        leverage_limit: u64,
        fixed_fee_split: u64,
        mfee_percent: u64,
        sol_jlp_feed: [u8; 32], // FIX incorrect name
        jlp_sol_feed: [u8; 32], // FIX incorrect name
        jlp_mint: Pubkey,
        keeper_fees: u64,
        usdc_mint: Pubkey,
        maturity_time: i64) -> ProgramResult {...}
```

- Use `cargo clippy` and resolve all warnings.
-

Remediation Progress

ACKNOWLEDGED: The client acknowledged the finding.

8. AUTOMATED TESTING

STATIC ANALYSIS REPORT

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	CRATE	DESCRIPTION
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`'s `Scalar29::sub`/`Scalar52::sub`
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.