DetectBox
Powered by UNSNARL

#DECENTRALISED PROACTIVE AUDIT WAR-ROOMS

Perpetual Derivatives Platform Security Assessment

ZENITH

JUNE 24TH, 2023

TABLE OF CONTENTS

SUMMARY

This report has been prepared for Zenith by Payper Finance to discover issues and vulnerabilities in the source code of the perpetual derivatives project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live

CERTIFICATION

This is to Certify that the **Zenith Security Assessment Report** was prepared using gold standard web3 security standards with all standard and advanced checks deployed.

The **Main Auditor** Involved in the Audit was -
Caron Gogwim(@cg0×01)

The **Detect-Warden** involved in the Audit was -
Rohan Jha(@rohan16___)

The Auditor mentioned hereby was chosen after self due-dilligence from the project's end.

All reported issues were either acknowledged or fixed by the project.

DetectBox's audit mitigation & residual vulnerability check did not find any major vulnerability to report within the Scope and requirements of the audit.

From
Team DetectBox

Zenith
Security Assessment Report

ABOUT DETECTBOX

DetectBox by **UNSNARL** brings to you world's first Decentralised Audit War-Rooms. An intense collaborative Environment where chosen auditors/audit team, project's decision makers, Projects developers and Detect-Wardens join hands to conduct rigorous security audits with absolute transparency followed by a **Proof of Audit (POA).**

We at DetectBox not only find bugs in smart-contracts but also take a deep look into the project's **tokenomics**, **white-paper**, **business logic**, **functional flows** and overall development health.

At DetectBox, you get to choose from a pool of Independent Security researchers which is curated by strong due-diligence. List your project requirements, verify the auditor's past work from their profiles and choose the auditor that rightly fits your project's needs and your budget.

Our auditors have successfully completed **200+ audits**, found over **2200+ vulnerabilities** and secured over **$102M+ worth of TVL.**

To know more about us visit - detectbox.io
To see our docs - https://unsnarl.gitbook.io/detectbox/

Yours Securely
UNSNARL

OVERVIEW

Project Summary

Zenith is the first perpetual future and options exchange on the Tezos blockchain. It provides users to go long or short on perpetual of their favorite assets with up to 10x leverage in all market conditions.

Official Documentation : https://docs.payperfi.com/

Audit Summary

A time-boxed independent security assessment of the Zenith protocol was done by Caron Gogwim (@CG0×01) and Team DetectBox, with a focus on the security aspects of the application's implementation.

We performed the security assessment based on the agreed scope, following our approach and methodology. Based on our scope and our performed activities, our security assessment revealed 1 Medium severity and 4 Low severity security issues. Additionally, different informational and gas optimization suggestions were also made which, if resolved appropriately, may improve the quality of the Project's Smart contract.

Audit Timeline : 16th June'23 - 20th June'23

Code Repository :
https://github.com/Zenith-FNO/Zenith-Smart-Contracts

Review commit hash :
d9f97d4852ddee6fc47e4381ce1cfbdfa8c72ff4

OVERVIEW

Audit Methodology

During our security assessments, we uphold a rigorous approach to maintain high-quality standards. Our methodology encompasses thorough functional testing and meticulous manual code reviews. To ensure comprehensive issue coverage, we employ checklists derived from industry best practices and widely recognized concerns, specifically tailored to Tezos smart contract assessment.

Throughout the smart contract audit process, we prioritize the following aspects to uphold excellence:

1. Code Quality: We diligently evaluate the overall quality of the code, aiming to identify any potential vulnerabilities or weaknesses.
2. Best Practices: Our assessments emphasize adherence to established best practices, ensuring that the smart contract follows industry-accepted guidelines and standards.
3. Documentation and Comments: We meticulously review code documentation and comments to ensure they accurately reflect the underlying logic and expected behaviour of the contract.

Auditing smart contracts involves a comprehensive analysis of the code to identify potential vulnerabilities and security risks. To achieve comprehensive coverage, we employ a series of security checklist tables, each addressing specific areas of concern. These include:

- System / Platform
- Access Control
- Storage

OVERVIEW

- Gas Issues and Efficiency
- Code Issues
- Error Handling and Exception Handling:
- Transaction Handling
- Entrypoint Validation
- Administration and Operator Functions
- Additional Topics and Test Cases

Vulnerability Summary

Severity classification

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

Findings Summary

High	0 issues
Medium	1 issue
Low	4 issues
Informational	3 issues
Gas Optimisations	0 issues

OVERVIEW

Audit Scope

The code under review is composed of multiple smart contracts written in the SmartPy language and includes 1311 SLOC- source lines of code (only source-code lines).

Sl. No.	Lines	SLOC
contracts/vmm.py	717	638
contracts/usdt.py	23	18
contracts/utils/helpers.py	104	88
contracts/utils/fa2.py	823	535
contracts/utils/address.py	22	20
contracts/utils/errors.py	23	12

Total SLOC : 1311

Auditors Involved :

Main Auditor -

Caron Gogwim - https://app.detectbox.io/profile/CGAuditor

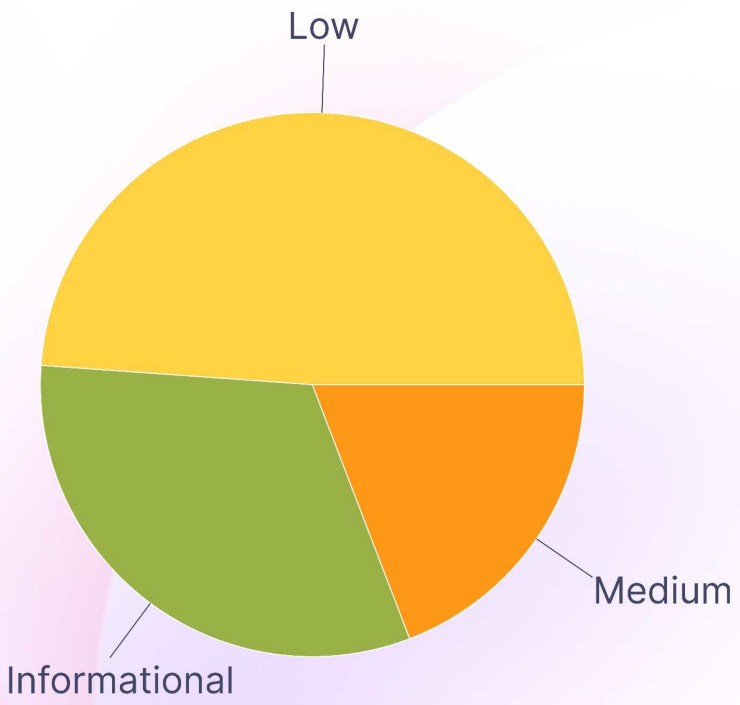
Detect Warden -

Rohan Jha - https://app.detectbox.io/profile/Rohan16

FINDINGS

Detailed Summary of Findings

Sl. No.	Name	Severity
M-01	No Function to toggle the pause and unpause	Medium
L-01	Use 2-step transfer of Administrator	Low
L-02	Use the latest version of SmartPy	Low
L-03	Lack of User's Input validation for direction in increasePosition entry_point	Low
L-04	Validity of information retrieved from other contracts	Low
I-01	Documenting compiler version	Informational
I-02	No Unit Testing	Informational
I-03	Documentation is missing in certain entry_points	Informational



FINDINGS

Static Analysis

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Manual Review

High Severity Issues

No High Severity issues were found.

Medium Severity Issues

M-01. No Function to toggle the pause and unpause

Description:

In the **VMM.py** contract's **__init__** function, the **paused** variable is initially set to **False**. However, it is worth noting that there is no explicit function within the contract to set the **paused** variable to **True** and pause the contract's functionality. While the contract contains several checks to ensure that the **paused** variable is not **True**, it lacks a specific function to toggle the **paused** state and enforce the pausing of the contract's operations.

FINDINGS

Code Snippets:
vmm.py

```
class Vmm_Contract(Helpers.Helpers):
    def __init__(self, admin, operators, usd_contract_address, oracle_address):
        self.init(
            administrator=admin,
            operators=operators,
            vmm=sp.record(token_amount=0, usd_amount=0, invariant=0),
            positions=sp.map(
                tkey=sp.TAddress,
                tvalue=sp.TRecord(
                    position=sp.TNat,
                    entry_price=sp.TNat,
                    funding_amount=sp.TInt,
                    position_value=sp.TNat,
                    collateral_amount=sp.TNat,
                    usd_amount=sp.TNat,
                ),
            ),fees_collected
            usd_contract_address=usd_contract_address,
            oracle_address=oracle_address,
            funding_period=sp.nat(3600),
            previous_funding_time=sp.now,
            upcoming_funding_time=(sp.now).add_seconds(3600),
            short_funding_rate=sp.record(value=sp.nat(0), direction="NA"),
            long_funding_rate=sp.record(value=sp.nat(0), direction="NA"),
            total_long=sp.nat(0),
            total_short=sp.nat(0),
            current_index_price=sp.nat(0),
            current_mark_price=sp.nat(0),
            paused=False, //@audit No entry_point to toggle pause.
            transaction_fees=sp.nat(2),
            fees_collected=sp.nat(0),
            decimal=10**6,
        )
```

FINDINGS

helpers.py

```
def _notPaused(self):
        sp.verify(self.data.paused == False, Errors.Paused)
```

Recommendations :

Add entry_point to toggle pause and unpause.

Low Severity Issues

L-01. Use 2-step transfer of Administrator

Description:

The current implementation of the **updateAdmin** entry-point in the **VMM.py** contract allows for directly setting a new address for the administrator role without any additional verification or confirmation steps. This approach poses a potential risk as it allows for the possibility of mistakenly assigning the administrator role to an incorrect or non-functional address. In such cases, there is a risk of losing control over the contract and potentially locking funds within it.

Code Snippets:
vmm.py: 80-86

FINDINGS

```
@sp.entry_point
    def updateAdmin(self, admin):
        self._onlyAdmin()
        self._notPaused() //@audit Use 2-step transfer of ownership.
        sp.set_type(admin, sp.TAddress)
        self.update_index_price()
        self.data.administrator = admin
```

Recommendations :

To mitigate the potential risks associated with changing critical privileged addresses, it is recommended to implement a two-step procedure. This approach adds an extra layer of security by requiring confirmation from both the current privileged address and the newly proposed address.

Here's a recommended approach:

1. **Step 1: Proposal:** The current privileged address initiates the process by proposing a new address for the administrator role.

2. **Step 2: Confirmation:** The newly proposed address acknowledges and accepts the proposal. This confirmation step can be implemented as a separate entry-point that accepts the proposed address.

3. **Step 3: Update the privileged address:** After successful confirmation from the newly proposed address, the contract can update the administrator address with the proposed address. This step should only be executed if both steps 1 and 2 have been successfully completed.

FINDINGS

L-02. Use the latest version of SmartPy

Description:

The contracts were written using a legacy version of SmartPy. While this version does not receive any new features or updates, it is highly recommended to transition to the latest version of SmartPy.

The latest version introduces new features and functionalities that enhance the development experience, including improved DSL syntax, additional libraries, enhanced testing frameworks, and better integration with Tezos-specific functionalities.

Recommendations:

It's recommended to use the latest version of SmartPy instead of relying on legacy versions.

L-03. Lack of User's Input validation for direction in increasePosition entry_point

Description:
The **increasePosition** entry point is responsible for increasing a user's position in the trading platform. It takes a direction parameter, where 1 represents a long position and 2 represents a short position. Currently, the code includes if conditions to check if the direction is 1 or 2. However, there is no explicit validation to ensure that the direction value is not greater than 2.
Although the current implementation has if conditions for checking if the value is 1 or 2, it is considered a best practice to include a verification step to ensure the value is within the expected range.

FINDINGS

Code Snippets:

```
@sp.entry_point
    def increasePosition(self, direction, usd_amount, leverage_multiple):
        self._notPaused()
        sp.set_type(leverage_multiple, sp.TNat)
        sp.set_type(usd_amount, sp.TNat)
        self.update_index_price()
        sp.verify(usd_amount > 0, message="POSITION CAN'T BE ZERO")
        sp.verify(leverage_multiple > 0, message="LEVERAGE CAN'T BE ZERO")
        sp.verify(direction =< 2, message="DIRECTION CAN'T BE MORE THAN TWO")

        self.transfer_usd(from_=sp.sender, to_=sp.self_address, amount_=usd_amou
nt)
        usd_amount1 = sp.local(
            "usd_amount1",
            sp.as_nat(usd_amount - (usd_amount * self.data.transaction_fees) / 1
00),
        )

        with sp.if_(direction == 1):...
            pos_value = sp.local(


        with sp.if_(direction == 2):...
            pos_value = sp.local(
```

Recommendations:

It is advisable to include a verification statement to validate the direction value. This can be achieved by adding a verify statement to check if the direction value is less than or equal to 2.

```
sp.verify(direction =< 2, message="DIRECTION CAN'T BE MORE THAN TWO")
```

FINDINGS

L-04. Validity of information retrieved from other contracts

Description:

The smart contract “helpers.py” retrieves price information from the external Oracle but does not check the age of the data. To ensure the validity and reliability of the retrieved data, it is recommended to check the age of the data and determine its staleness.

Code Snippets:

```
def update_index_price(self):
        """
        The function updates the current index price by retrieving data from an
oracle.
        """
        oracle_data = sp.view(
            "getlastCompletedData",
            sp.address("KT1C1sYNxacr8LPZimA512gAfWajdGah75nq"),
            sp.unit,
            t=sp.TRecord(
                round=sp.TNat,
                epoch=sp.TNat,
                data=sp.TNat,
                percentOracleResponse=sp.TNat,
                decimals=sp.TNat,
                lastUpdatedAt=sp.TTimestamp,
            ).layout(
                (
                    "round",
                    (
                        "epoch",
                        (
                            "data",
                            ("percentOracleResponse", ("decimals", "lastUpdatedA
t")),
                        ),
                    ),
                )
            ),
        ).open_some()
        self.data.current_index_price = oracle_data.data
```

FINDINGS

Recommendations:

By implementing these recommendations, the contract will validate the age of the data and ensure that only up-to-date information is used, enhancing the reliability of the contract's functionality. This can be achieved by comparing the last update timestamp of the oracle data with the current time. Here's a suggested implementation to address this issue:

```
current_time = sp.now
        last_updated_time = oracle_data.lastUpdatedAt

        # Calculate the time difference in seconds
        time_difference = current_time - last_updated_time

        # Define the staleness threshold (e.g., 5 minutes)
        staleness_threshold = sp.minutes(5)

        # Verify if the data is stale based on the staleness threshold
        sp.verify(
            time_difference <= staleness_threshold,
            message="Oracle data is stale",
        )

        self.data.current_index_price = oracle_data.data
```

FINDINGS

Informational Issues

I-01. Documenting compiler version

Description:

The documentation for the smart contracts lacks information about the specific compiler and its version used during the compilation process. Documentation should list the compiler and its version used. This helps identify and follow compiler issues in the future.

Recommendations:

We recommend adding information about which SmartPy compiler version has been used to compile the code.

I-02. No Unit Testing

Description:

The smart contracts lack comprehensive test coverage, as there are no existing test cases to validate their functionality. This poses a risk as it becomes difficult to ensure the contracts behave as intended and handle various scenarios correctly. Additionally, there are no on-chain test cases available for the contracts in scope.

We also noted that for all of the contracts in scope, there are no on-chain test cases available.

FINDINGS

Recommendations:

We recommend defining and executing test cases in SmartPy appropriately covering all entry points. In addition to the testing in SmartPy, we also advise defining appropriate on-chain testing.

I-03. Documentation is missing in certain entry_points

Description:

The smart contracts lack proper documentation at certain entry points, specifically in the following functions:

Helpers.py:
- *_onlyAdmin*
- *_notPaused*
- *transfer_usd*

vmm.py:
- *updateFundingPeriod*
- *updateAdmin*

These functions lack a docstring, which is a string enclosed in triple quotes, that describes the purpose and behavior of the functions. Documentation plays a vital role in understanding and maintaining the contracts, and its absence hinders comprehension and collaboration efforts.

FINDINGS

Recommendations:

To address this issue, we recommend the following improvements to the documentation. Include a docstring for each function that currently lacks one. The docstring should provide comprehensive information about the entry point, including:

- A clear description of the function's purpose and functionality.
- Details about the input parameters, including their meanings, types, and any restrictions or requirements.
- Pre-conditions specifying who can call the entry point and any necessary conditions for its execution.
- Post-conditions defining the expected outcomes or effects of executing the function.
- Mention any operations that may be performed, such as transfers or contract origination, and their impact.

POST AUDIT CONCLUSION

Fixing the Findings

Sl. No.	Name	Status
M-01	No Function to toggle the pause and unpause	Fixed
L-01	Use 2-step transfer of Administrator	Fixed
L-02	Use the latest version of SmartPy	Acknowledged
L-03	Lack of User's Input validation for direction in increasePosition entry_point	Fixed
L-04	Validity of information retrieved from other contracts	Fixed
I-01	Documenting compiler version	Fixed
I-02	No Unit Testing	Fixed
I-03	Documentation is missing in certain entry_points	Fixed

All issues were fixed/acknowledged by the project's developers at the **final commit hash** - 61cda9367a0112a9f077640a00d30dcc33321fd2

APPENDIX

Finding Categories

Finding Categories

Centralization/Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

DISCLAIMER

DetectBox (by UNSNARL) has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. UNSNARL and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

DISCLAIMER

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. UNSNARL is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. UNSNARL's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

UNSNARL in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will UNSNARL, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

DISCLAIMER

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

UNSNARL will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

UNSNARL will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.

GLORY OF AUDITORS AT DETECTBOX

DetectBox has created a pool of best auditors across the globe with significant experience in the web3 security industry auditing multiple protocols across multiple chains. Making audits better through **Proof of Talent**.

217+

Projects Audited

$100M+

Amount Secured

2200+

Vulnerabilities Detected

Follow Our Journey on

@unsnarl_secure

UNSNARL

founders@unsnarl.io

www.detectbox.io