Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

panther.

Panther Protocol

Veridise Inc.

September 18, 2025

► **Prepared For:**

Panther
https://www.pantherprotocol.io/

► **Prepared By:**

Tyler Diamond
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Sep. 18, 2025 V1

Contents

Contents iii

1 Executive Summary 1

2 Project Dashboard 3

3 Security Assessment Goals and Scope 4
- 3.1 Security Assessment Goals . . . 4
- 3.2 Security Assessment Methodology . . . 4
- 3.3 Scope . . . 4
- 3.4 Classification of Vulnerabilities . . . 4

4 Vulnerability Report 6
- 4.1 Detailed Description of Issues . . . 7
  - 4.1.1 V-PAN-VUL-001: Bypassing internal transfer limits via swaps . . . 7
  - 4.1.2 V-PAN-VUL-002: Denial of Service from oversized UTXO weighted amount 8
  - 4.1.3 V-PAN-VUL-003: Maintainability issues . . . 10

Glossary 11

1 Executive Summary

From Sep. 12, 2025 to Sep. 15, 2025, Panther engaged Veridise to conduct a security assessment of an enhancement in the Panther Protocol. The security assessment evaluated the enforcement logic of transfer limits within the protocol's primary zero-knowledge funds transfer circuit.

Compared to the previous version, which Veridise has reviewed previously*, the new version enables users to create self-destined unspent transaction output (UTXO) without being constrained by the internal transaction limit.

Veridise conducted the assessment over 4 person-days, with 2 security analysts reviewing the project over 2 days on commit `06a8186`. The review strategy combined tool-assisted analysis of the program's source code with an in-depth manual code review.

Project Summary. Panther Protocol is a privacy-preserving, compliance-oriented multi-chain digital asset management system that supports ERC-20, ERC-721, and ERC-1155 tokens. The protocol relies heavily on advanced cryptographic techniques and Zero-Knowledge Proof technology to implement its core features.

Users of the protocol are able to perform the following functions in a privacy-preserving manner:

- Deposit tokens into the protocol
- Withdraw tokens from the protocol
- Transfer tokens from one internal account to another
- Swap deposited tokens on selected decentralized exchanges
- Provide evidence of asset movements to a designated third party upon request
- Stake their assets
- Earn rewards in the form of PRP points through participation in protocol-wide activities
- Convert PRP points into ZKP tokens
- Seamlessly transfer funds between different supported networks

Panther Protocol also includes the following compliance features:

- Users of the protocol are assigned to Zones. Each zone has its own policies regarding transaction limits, potential destinations, KYC/KYC requirements, and more.
- Almost all transactions require a KYC/KYT certificate tied to the specific transaction being processed and issued by an external trusted service provider.

The security assessment evaluated the enforcement of transfer limits on internal transfers to other users, while ensuring that self-directed transfers remain exempt from these restrictions.

Code Assessment. The Panther Protocol developers submitted a Pull Request (PR) with proposed changes for review. The PR included a detailed description of its objectives and the specific implementation approach. Compared to the previous security review, no modifications were introduced beyond the adjustments to the internal transfer limit logic.

* The previous audit report, if it is publicly available, can be found on Veridise's website at `https://veridise.com/audits/`

Summary of Issues Detected. The security assessment uncovered 3 issues, 1 of which was assessed to be of high or critical severity by the Veridise analysts. Specifically, V-PAN-VUL-001 details that swaps do not enforce transfer limits. The Veridise analysts also identified 1 low-severity issue, in which extremely large values can lead to an unsatisfiable circuit. Additionally, 1 warning issue was found. The Panther Protocol developers have fixed the high-severity and warning issues and have indicated they will not address the low-severity issue.

Recommendations. After conducting the assessment of the proposed changes, the security analysts had a few suggestions to improve the Panther Protocol.

Improve comments. Given Veridise's prior experience with the Panther protocol, it was feasible to conduct the review within a constrained time-frame despite the significant contextual knowledge required to assess these changes. Nevertheless, the review would have benefited from more detailed documentation of the input and output signals of the main protocol circuit, zSwap. Hence, our primary recommendation is to improve the source code documentation by adding clear explanations of the input and output values.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Panther Protocol	`06a8186`	Circom	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Sep. 12–Sep. 15, 2025	Manual & Tools	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	1	1	1
Medium-Severity Issues	0	0	0
Low-Severity Issues	1	0	0
Warning-Severity Issues	1	1	1
Informational-Severity Issues	0	0	0
TOTAL	3	2	2

Table 2.4: Category Breakdown.

Name	Number
Logic Error	1
Data Validation	1
Maintainability	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of Panther Protocol's source code surrounding transaction limits related to internal transfers. During the assessment, the security analysts aimed to answer questions such as:

- Can users bypass the transfer limit by creating multiple UTXOs?
- Does swapping affect the enforcement of transfer limits?
- Are time-period limits enforced correctly?
- Is the enforcement logic of `zZoneInternalMaxAmount` consistent with other critical limits, such as `zZoneDepositMaxAmount` and `zZoneWithdrawMaxAmount`?
- Are there any Circom-specific implementation-level vulnerabilities, such as field arithmetic overflows and underflows?

3.2 Security Assessment Methodology

Security Assessment Methodology. To address the questions above, the security assessment involved a combination of human experts and automated program analysis & testing tools. In particular, the security assessment was conducted with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, security analysts leveraged Veridise's custom circom analysis tool ZK Vanguard. This tool is designed to find instances of common circuit vulnerabilities, such as under-constrained signals and non-deterministic witnesses.

3.3 Scope

The scope of this security assessment was limited to the changes to a single file:

- `circuits/circuits/zSwapV1.circom`

This is a ZK circuit implementing the core swapping and transfer logic of the Panther Protocol.

3.4 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

The likelihood of a vulnerability is evaluated according to the Table 3.2.

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-PAN-VUL-001	Bypassing internal transfer limits via swaps	High	Fixed
V-PAN-VUL-002	Denial of Service from oversized UTXO . . .	Low	Won't Fix
V-PAN-VUL-003	Maintainability issues	Warning	Fixed

4.1 Detailed Description of Issues

4.1.1 V-PAN-VUL-001: Bypassing internal transfer limits via swaps

Severity	High	**Commit**	N/A
Type	Logic Error	**Status**	Fixed
Location(s)	`circuits/circuits/zSwapV1.circom:630-631`		
Confirmed Fix At	`https://github.com/pantherfoundation/panther-core/pull/1/`, 39b770d		

Description To enforce internal transfer limits, the zSwap circuit computes a utxoOutWeightedAmount value, which represents the transferred token amount normalized into a universal weighted unit:

```
utxoOutWeightedAmount[i] <== utxoOutAmount[i] * zAssetWeight[swapToken];
```

This value is then compared against `zZoneInternalMaxAmount` to ensure the transfer remains within the configured limit. However, the circuit implements special handling for swap UTXOs. In such cases:

1. Users are expected to set the UTXO out amount to `0`, since the actual transferred value is appended later at the smart contract level.
2. The circuit also permits swap UTXOs to be issued to recipients different from the sender.

As a result, the weighted amount for swap UTXOs is always computed as `0`, bypassing the intended internal transfer limit checks.

Impact Because swap UTXOs always evaluate to zero, they automatically satisfy any internal transfer limit. This creates a bypass vulnerability, allowing users to exceed transfer restrictions by:

1. Using a swap instead of a direct transfer.
2. Assigning the newly created swap UTXO to another user.

In practice, this means internal transfer caps can be circumvented, undermining the purpose of the restriction.

Recommendation Swaps must only be permitted when the sender and recipient are the same user.

Developer Response The developers now enforce that the UTXO of an `isSwapUtxo` must be destined for the sending `zAccount`.

4.1.2 V-PAN-VUL-002: Denial of Service from oversized UTXO weighted amount

Severity	Low	**Commit**	N/A
Type	Data Validation	**Status**	Won't Fix
Location(s)	`circuits/circuits/zSwapV1.circom:622-623`		
Confirmed Fix At	N/A		

To enforce internal transfer limits, the **zSwap circuit** computes a `utxoOutWeightedAmount` value, representing the transferred token amount normalized into a universal weighted unit:

```
utxoOutWeightedAmount[i] <== utxoOutAmount[i] * zAssetWeight[swapToken];
```

This computed value is then passed into the `isLessThanEq_weightedUtxoOutAmount_zZoneInternalMaxAmount` component, which uses a `ForceLessThan(96)` comparator. The issue arises because the comparator is instantiated to handle a maximum of **96 bits**, while the multiplication above can yield results up to **112 bits** in the worst case. This mismatch creates a scenario where certain valid UTXOs exceed the comparator's range, causing the circuit to fail.

Impact When `utxoOutWeightedAmount` exceeds the 96-bit limit, the circuit cannot generate valid proofs. This results in a **denial-of-service** for affected users, as their transactions cannot be processed.

- In some cases (e.g., when the weighted value is just slightly above the 96-bit threshold, such as 97 bits), users may be able to avoid the issue by splitting their UTXOs into smaller amounts.
- However, if the multiplication produces significantly larger values (e.g., approaching the 112-bit maximum), such recovery becomes impossible, potentially leaving user funds effectively **locked and unusable**.

This situation is most likely triggered by a sudden spike in asset price, which increases the `zAssetWeight` .

Recommendation The `ForceLessThan` comparator should be instantiated to handle 112 bits.

Developer Response The developers have provided the following reasons on why they believe this is not an issue:

1. **Design Intent of Universal Weighted Units**: The mentioned "universal weighted unit" is intentionally designed to represent the value of tokens in a standardized unit approximating real-world monetary value (e.g., USD). The **minimum** targeted value for this unit is set to 1e-6 USD, representing monetary values that are granular yet practical in financial terms.
2. **Numerical Limits vs. Real-World Context**: For `ForceLessThan(96)` to fail, the weighted amount would need to exceed 96 bits of precision, i.e., represent values greater than `2^96` `7.9e+28` in the universal weighted unit. This implies a monetary value of **over 7.9e+22 USD** (given the unit precision of 1e-6 USD). To put this into perspective:
   - This value is 727 million times larger than the world's total GDP (1.09e+14 USD).
   - It is far beyond the combined financial value of all real-world assets, making this scenario infeasible within the operational scope of the protocol.

3. **Protocol Applicability**: The protocol serves to facilitate secure and private financial transactions. By definition, it does not aim to handle transactions representing values several orders of magnitude higher than the entire global economy. Thus, values exceeding the comparator's 96-bit range are irrelevant and unrealistic in this context.
4. **Smart contract enforcement:** Yet-to-be-implemented smart contracts responsible for updating configuration parameters for assets should ensure that the combination of an asset's "scale" and "weight" guarantee the maximum possible amount of a UTXO fits within the 96-bit constraint.

Update Veridise Response: The security analysts acknowledge that the scenario in which this could be exploited is extremely unlikely. However, the circuit itself allows values of this size to be used and therefore the completeness of the circuit is affected.

4.1.3 V-PAN-VUL-003: Maintainability issues

Severity	Warning	**Commit**	N/A
Type	Maintainability	**Status**	Fixed
Location(s)	`circuits/circuits/zSwapV1.circom`		
Confirmed Fix At	`https://github.com/pantherfoundation/panther-core/pull/1/`, 67e5386		

Description During the security review, the following maintainability issues were identified in the `zSwapV1.circom` circuit:

1. **Duplicate Imports** The circuit includes `zAccountNoteHasher.circom` and `zAccountNullifierHasher.circom` twice, introducing unnecessary redundancy.
2. **Incorrect Array Instantiation** The `utxoOutCommitmentProver[nUtxoIn]` array of components is instantiated using `nUtxoIn`, whereas it should be instantiated with `nUtxoOut`. This misalignment may cause logical inconsistencies in how outputs are processed.
3. **Inconsistent Swap UTXO Indexing** The expression `var isSwapUtxo = isSwap & (i == nUtxoOut - 1)` assumes the last output always corresponds to the swap UTXO. However, the circuit also defines a `SwapTokenIndex()` function that always returns `1`, independent of `nUtxoOut`.
   - Since `nUtxoOut = 2` in the current implementation, both methods resolve to index `1`.
   - If `nUtxoOut` changes in the future, the mismatch will introduce inconsistencies or functional bugs.
4. **Inconsistent Merkle Tree Selector Comments** The comments on what the value should be for the `zAccountUtxoInMerkleTreeSelector` and `utxoInMerkleTreeSelector` do not match, even though the ordering of the trees is the same in both instances.

Impact These issues do not directly compromise security in the current configuration but pose significant maintainability and correctness risks:

- **Code Complexity & Redundancy**: Duplicate imports increase cognitive overhead and risk of divergence in updates.
- **Logical Errors**: Incorrect instantiation (`nUtxoIn` vs. `nUtxoOut`) may cause subtle bugs in future circuit modifications or audits.
- **Fragile Design**: Reliance on implicit assumptions (e.g., swap UTXO always at the last index) reduces adaptability. If parameters change (e.g., `nUtxoOut > 2`), functionality may silently break. Together, these issues make the circuit harder to maintain, more error-prone, and less resilient to future changes.

Recommendation It is recommended to address the mentioned defects.

Developer Response The developers have implemented fixes for all mentioned defects.

Glossary

unspent transaction output A data structure that represents a certain amount of cryptocurrency and who has the authorization to spend it (usually via digital signature). Similar to physical notes, UTXOs are immutable and must be spent in entirety. Leftover change is sent back to the sender with an additional UTXO addressed to himself. This is distinct from account-based cryptocurrencies, such as Ethereum, that track balances by adding and subtracting from a single balance per account. . 1, 11

UTXO unspent transaction output. 1