



Pantos

**Pantos on-chain components
for Ethereum and compatible blockchains
SMART CONTRACT AUDIT**

21.10.2024

Made in Germany by Softstack.io



Table of contents

1. Disclaimer.....	4
2. About the Project and Company	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology	8
5. Metrics	9
5.1 Tested Contract Files	9
5.2 CallGraph.....	11
5.3 Inheritance Graph.....	13
5.4 Source Lines & Risk.....	14
5.5 Capabilities	15
5.6 Dependencies / External imports	16
5.7 Source Unites in Scope	18
6. Scope of Work.....	21
6.1 Findings Overview	22
6.2 Manual and Automated Vulnerability Test.....	23
6.2.1 Absence of Slashing Mechanism in Staking System	23
6.2.2 Unsafe ERC Operation in PantosRegistryFacet.sol	25
6.2.3 Potential Division by Zero and Precision Loss in _transferFee Function.....	26
6.2.4 Potential Denial of Service Due to Unbounded Loops in Token and Service Node Unregistration.....	27
6.2.5 Lack of Minimum Value Checks in Critical Parameter Setting Functions	29



6.2.6 Immediate Parameter Changes Pose Risk of System Instability	31
6.2.7 Decimals Function Not Part of ERC-20 Standard.....	33
6.2.8 Symbol Function Not Part of ERC-20 Standard.....	35
6.2.9 Use of Single-Step Ownership Transfer Instead of Safer Two-Step Process.....	37
6.2.10 Unnecessary Use of pragma abicoder v2 in PantosRegistryFacet.sol	38
6.3 SWC Attacks	39
7. Executive Summary.....	44
8. About the Auditor	45



1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Pantos GmbH. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (18.06.2024)	Layout
0.4 (26.06.2024)	Automated Security Testing Manual Security Testing
0.5 (27.06.2024)	Verify Claims and Test Deployment
0.6 (02.06.2024)	Testing SWC Checks
0.9 (03.06.2024)	Summary and Recommendation
1.0 (10.07.2024)	Final document
1.1 (21.10.2024)	Re-check https://github.com/pantos-io/ethereum-contracts/releases/tag/2.0.1

2. About the Project and Company

Company address:

Pantos GmbH
Stella-Klein-Löw Weg 17
1020 Vienna | Austria



Website: <https://pantos.io>

LinkedIn: https://at.linkedin.com/company/pantos_io

Twitter (X): <https://twitter.com/PantosIO>

Discord: <https://discord.gg/bitpanda>

Telegram: https://t.me/PantosIO_EN

Medium: <https://medium.com/@PantosIO>

Youtube: <https://www.youtube.com/channel/UCs8FmLFt5PmF4fp5PjUAW6A>

Facebook: <https://www.facebook.com/PantosIO>

2.1 Project Overview

Pantos is a pioneering multi-blockchain token system developed to facilitate seamless and secure asset transfers across various blockchain networks. Originating as a research project by Bitpanda, a prominent digital investment platform, Pantos aims to overcome the interoperability challenges prevalent in the blockchain space. The primary objective of Pantos is to create a decentralized, open-source protocol that enables token interoperability between different blockchain platforms. By achieving this, Pantos seeks to enhance liquidity, foster innovation, and reduce fragmentation in the blockchain ecosystem.

Pantos is designed to support multiple blockchain networks, allowing for the smooth transfer of tokens and assets between them. This interoperability is achieved through the use of advanced technologies and protocols, such as atomic swaps and smart contracts. The Pantos protocol is built on a decentralized framework to ensure transparency, security, and trustlessness. Decentralized governance mechanisms are employed to manage protocol updates and decision-making processes. Pantos is engineered to handle a high volume of transactions efficiently, making it suitable for large-scale applications. Scalability solutions are integrated to accommodate growing user demand and network activity. Robust security measures, including cryptographic techniques and consensus algorithms, are implemented to protect user assets and data. Regular security audits and updates are conducted to mitigate potential vulnerabilities. Pantos offers an intuitive interface for users, developers, and businesses to interact with the protocol and utilize its features. Comprehensive documentation and developer tools are provided to facilitate seamless integration and application development.

Pantos enables decentralized finance (DeFi) applications to operate across multiple blockchain networks, enhancing liquidity and user reach. Users can leverage Pantos for cross-chain lending, borrowing, and trading activities. The protocol supports the transfer of various digital assets between different blockchain platforms without the need for intermediaries. Developers can utilize Pantos to build interoperable applications that require interaction with multiple blockchains. This opens up new possibilities for innovative blockchain solutions and services.

Continuous research efforts are undertaken to explore new technologies and improve the Pantos protocol. Expansion of supported blockchain networks to include more platforms is planned, increasing the versatility and reach of Pantos. Building a strong community of users, developers, and stakeholders is essential to drive the adoption and development of Pantos. Initiatives to support and incentivize developers to create applications on Pantos are also a key focus.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i. Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
 - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.
2. Testing and automated analysis that includes the following:
 - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

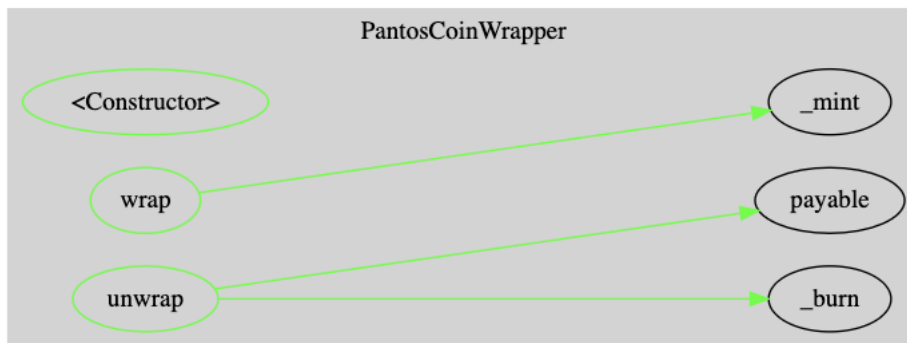
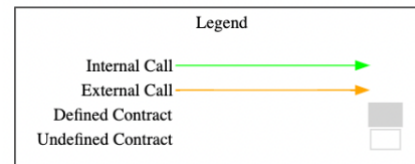
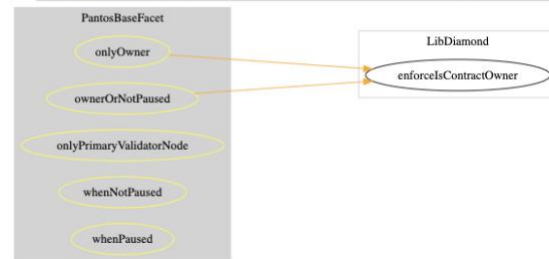
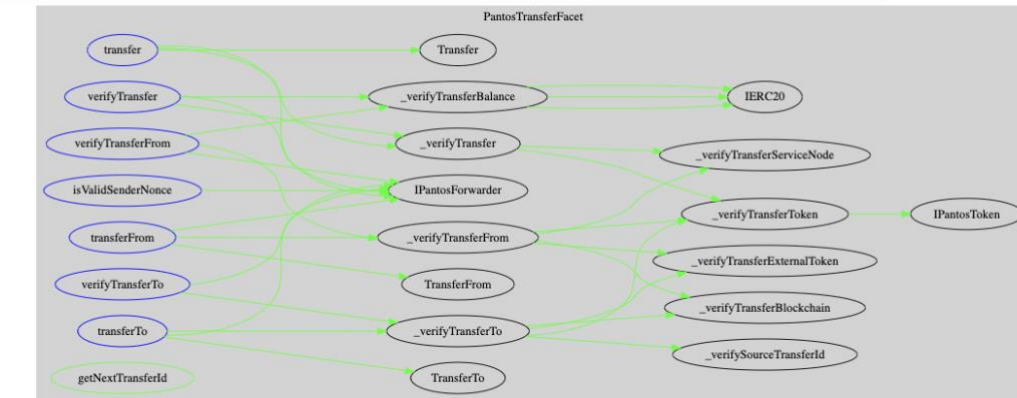
5.1 Tested Contract Files

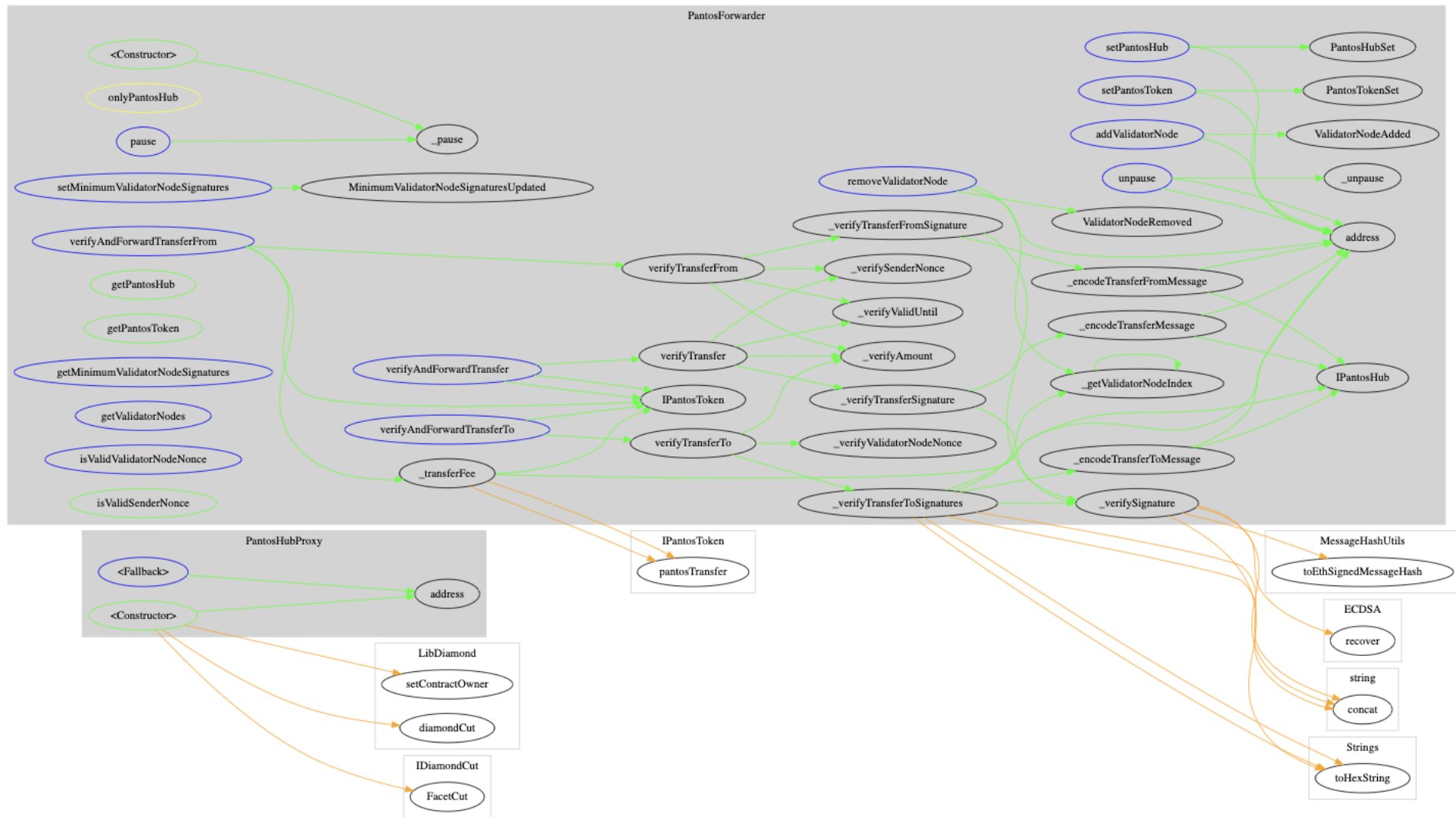
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

File	Fingerprint (MD5)
./contracts/src/PantosWrapper.sol	bdc484a3819ea15ac37d136da84519b9
./contracts/src/wrappers/PantosMaticWrapper.sol	13c708f4fe445654a813304cbc60c399
./contracts/src/wrappers/PantosCronosWrapper.sol	3216c3b8b534c9de82ffc1c8a86158b
./contracts/src/wrappers/PantosFantomWrapper.sol	f12a367d082a5a3df65a830133e3711c
./contracts/src/wrappers/PantosBnbWrapper.sol	34f1a6e6647adbdac81d32bc28bd214d
./contracts/src/wrappers/PantosAvaxWrapper.sol	20b9b1c2a3bb15e6b5460fdc10a577e1
./contracts/src/wrappers/PantosEtherWrapper.sol	82c86acffae9e1fbb107996482c8b072
./contracts/src/wrappers/PantosCeloWrapper.sol	33d38c3705888b0a99e1712cc717aa67
./contracts/src/PantosSimpleToken.sol	e8cd73dc89c50be46c0d4df456282f9d
./contracts/src/migrations/MigrationTokenBurnable.sol	3034354dcb4b52bf2eef60fcacd057f3
./contracts/src/migrations/MigrationToken.sol	18408e3d80b2ed5828ee1ee761e551ff
./contracts/src/migrations/MigrationTokenPausable.sol	7b14958d014e972f1cf05263251290eb
./contracts/src/migrations/MigrationTokenBurnablePausable.sol	9145d6c8468914a42cd2f88b49089941
./contracts/src/PantosCoinWrapper.sol	4732485a980d104ac0c20aa3aa5ea825
./contracts/src/facets/PantosRegistryFacet.sol	8b92268e19f62a92067b87d73e05df7f
./contracts/src/facets/PantosTransferFacet.sol	9422104de8f3f8f1916ae885b5715799
./contracts/src/facets/PantosBaseFacet.sol	47a8cfd493d63b743094de55c412be24

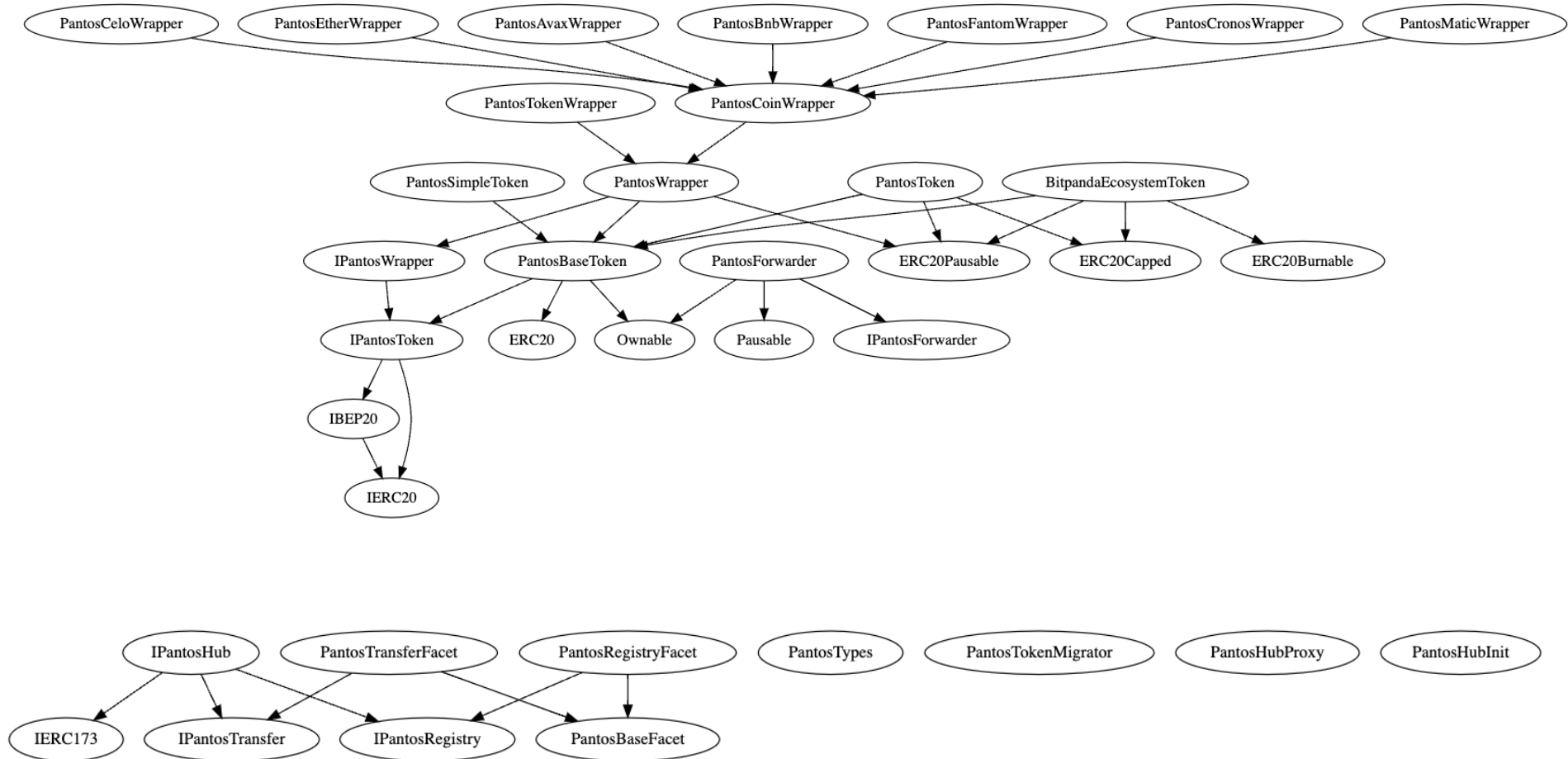
./contracts/src/upgradeInitializers/PantosHubInit.sol	cd0872c46f2cff9959e07b3a8f35ee1f
./contracts/src/PantosBaseToken.sol	dc80b0fab59f415fd512e39b7b83656c
./contracts/src/PantosForwarder.sol	7526596edf58d4b72b9b24afdbacbf80
./contracts/src/PantosHubStorage.sol	f92cc95bdf583b199ba74862b41e7248
./contracts/src/PantosHubProxy.sol	1cc48d5185352ebefacef4c14e90b9f7
./contracts/src/PantosToken.sol	7ee2a847de31fe6cc14d8bdc3887894f
./contracts/src/PantosTokenWrapper.sol	31ac480e54dcce8fb3b58295d8208e3a
./contracts/src/PantosTokenMigrator.sol	7c610e26c4aadd2588a7c2f5cd4715e1
./contracts/src/BitpandaEcosystemToken.sol	c1b39f2068593f8863d1c51bf51084e5
./contracts/src/interfaces/IPantosRegistry.sol	3438f34bc829a3782444db8f5146cbf6
./contracts/src/interfaces/IPantosTransfer.sol	4cb8e894e251cae4a59314fb3b4f16f4
./contracts/src/interfaces/PantosTypes.sol	8404ff7fb8ae37e9113dfcb4c65f518a
./contracts/src/interfaces/IPantosWrapper.sol	5a28a9c5d802ea4194e881ba29f54e2a
./contracts/src/interfaces/IPantosHub.sol	2e8e82baa47912bcc267e30e50385920
./contracts/src/interfaces/IPantosForwarder.sol	2825e27ecf95836692a6f0ac74fc43d9
./contracts/src/interfaces/IPantosToken.sol	a81b6320f08108ffc23b40273373a3b0
./contracts/src/interfaces/IBEP20.sol	759747ca251e4a1e299b5e28f0a98fca

5.2 CallGraph

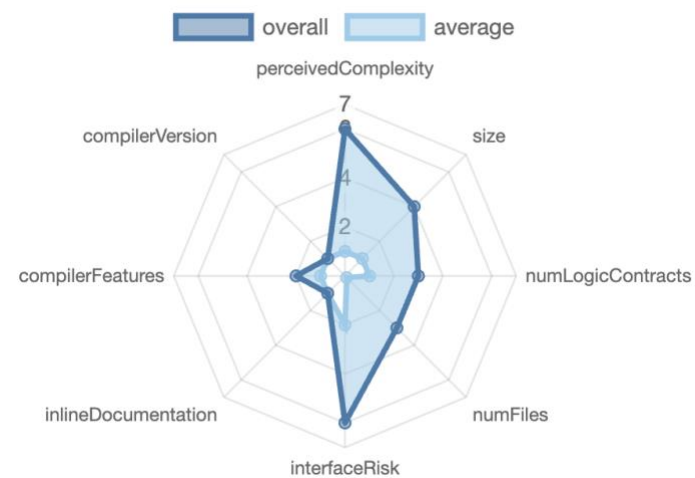
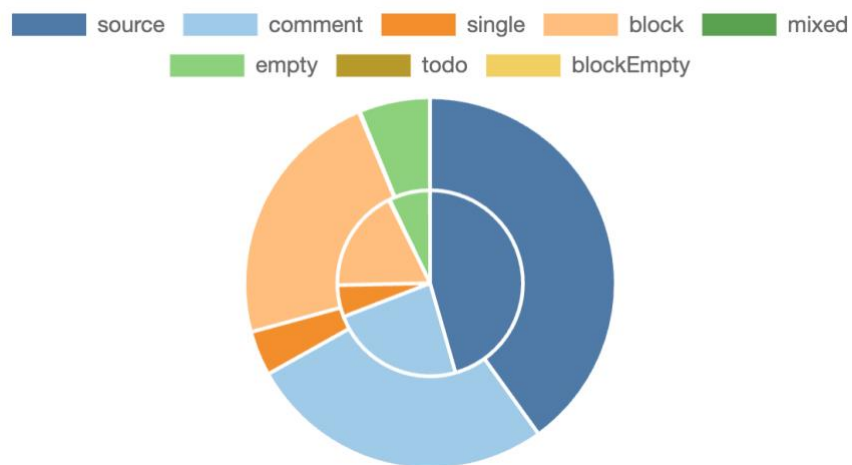








5.3 Inheritance Graph









5.4 Source Lines & Risk





5.5 Capabilities

Solidity Versions observed	 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
0.8.26		yes	yes (2 asm blocks)	

 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRecover	 New/Create/Create2
yes		yes	yes		


Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable
191	6

External	Internal	Private	Pure	View
137	122	25	2	113

StateVariables

Total	 Public
44	0

5.6 Dependencies / External imports

Dependency / Import Path	Source
@diamond/interfaces/IDiamondCut.sol	https://github.com/mudgen/diamond-1/blob/master/contracts/interfaces/IDiamondCut.sol
@diamond/interfaces/IDiamondLoupe.sol	https://github.com/mudgen/diamond-1/blob/master/contracts/interfaces/IDiamondLoupe.sol
@diamond/interfaces/IERC165.sol	https://github.com/mudgen/diamond-1/blob/master/contracts/interfaces/IERC165.sol
@diamond/interfaces/IERC173.sol	https://github.com/mudgen/diamond-1/blob/master/contracts/interfaces/IERC173.sol
@diamond/libraries/LibDiamond.sol	https://github.com/mudgen/diamond-1/blob/master/contracts/libraries/LibDiamond.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Burnable.sol

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Capped.sol
@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Pausable.sol
@openzeppelin/contracts/utils/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Pausable.sol
@openzeppelin/contracts/utils/Strings.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Strings.sol
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol
@openzeppelin/contracts/utils/cryptography/MessageHashUtils.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/MessageHashUtils.sol

5.7 Source Unites in Scope

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/src/PantosSimpleToken.sol	1		25	25	15	8	6
contracts/src/wrappers/PantosCeloWrapper.sol	1		22	22	11	6	7
contracts/src/wrappers/PantosEtherWrapper.sol	1		22	22	11	6	7
contracts/src/wrappers/PantosAvaxWrapper.sol	1		22	22	11	6	7
contracts/src/wrappers/PantosBnbWrapper.sol	1		22	22	11	6	7
contracts/src/wrappers/PantosFantomWrapper.sol	1		22	22	11	6	7
contracts/src/wrappers/PantosCronosWrapper.sol	1		22	22	11	6	7
contracts/src/wrappers/PantosMaticWrapper.sol	1		22	22	11	6	7
contracts/src/interfaces/IBEP20.sol		1	31	15	5	17	11
contracts/src/interfaces/IPantosToken.sol		1	75	34	8	51	13
contracts/src/interfaces/IPantosForwarder.sol		1	217	66	12	145	25

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/src/interfaces/IPantosHub.sol		1	23	23	6	14	7
contracts/src/interfaces/IPantosWrapper.sol		1	44	22	5	32	12
contracts/src/interfaces/PantosTypes.sol	1		75	75	62	12	1
contracts/src/interfaces/IPantosTransfer.sol		1	192	74	43	110	17
contracts/src/interfaces/IPantosRegistry.sol		1	706	156	44	498	93
contracts/src/BitpandaEcosystemToken.sol	1		130	109	61	33	50
contracts/src/PantosTokenMigrator.sol	1		139	139	78	49	43
contracts/src/PantosTokenWrapper.sol	1		73	73	44	22	32
contracts/src/PantosToken.sol	1		124	103	55	33	48
contracts/src/PantosHubProxy.sol	1		82	82	39	39	73
contracts/src/PantosHubStorage.sol			37	37	27	8	
contracts/src/PantosForwarder.sol	1		613	537	353	140	201
contracts/src/PantosBaseToken.sol	1		153	117	64	37	46
contracts/src/upgradeInitializers/PantosHubInit.sol	1		105	105	61	31	14
contracts/src/facets/PantosBaseFacet.sol	1		72	72	32	32	8
contracts/src/facets/PantosTransferFacet.sol	1		376	329	241	68	90

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/src/facets/PantosRegistryFacet.sol	1		977	867	601	215	314
contracts/src/PantosCoinWrapper.sol	1		34	34	18	12	19
contracts/src/PantosWrapper.sol	1		151	124	69	47	50
Totals	22	7	4608	3372	2020	1695	1222

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

6. Scope of Work

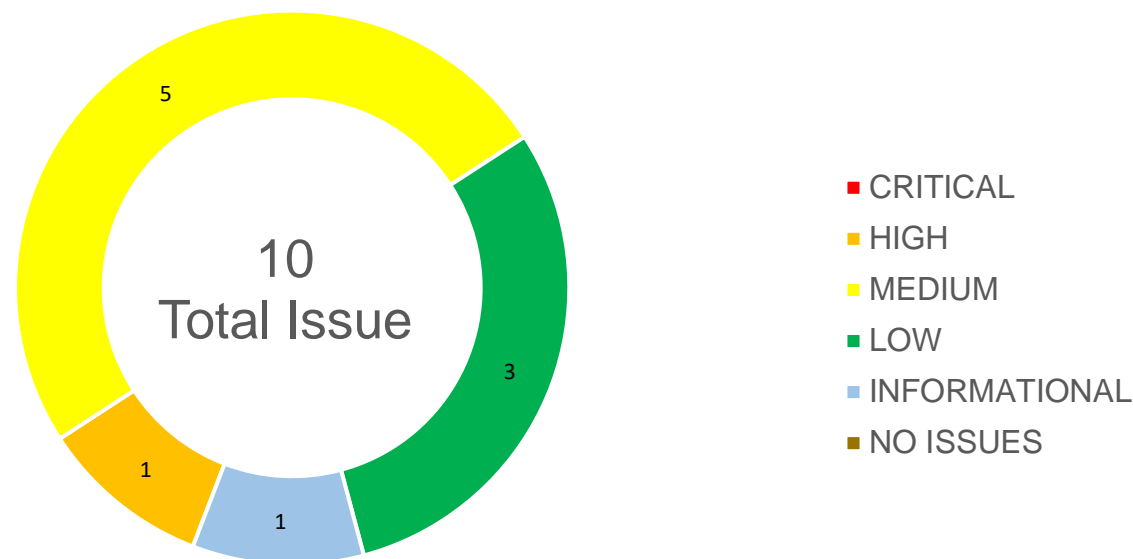
The Pantos Team provided us with the files that needs to be tested. The scope of the audit are the core contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

1. **Compliance with Best Practices:** The audit should ensure that the contracts adhere to smart contract best practices, including checking for common vulnerabilities such as reentrancy attacks and overflow/underflow issues.
2. **Proper Access Control:** The contracts should implement and enforce proper access control mechanisms to ensure that only authorized entities can execute sensitive functions.
3. **Efficient Gas Usage:** The contracts should be optimized for efficient gas usage, minimizing unnecessary computations and storage operations to reduce transaction costs.
4. **Data Integrity and Consistency:** The contracts should maintain data integrity and consistency, ensuring that all state changes are accurately reflected and that no data is inadvertently lost or corrupted.
5. **Upgradeability and Flexibility:** The contracts should be designed with upgradeability and flexibility in mind, allowing for future enhancements and modifications without disrupting the existing functionality and user experience.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Absence of Slashing Mechanism in Staking System	HIGH	FIXED
6.2.2	Unsafe ERC Operation in PantosRegistryFacet.sol	MEDIUM	ACKNOWLEDGED
6.2.3	Potential Division by Zero and Precision Loss in _transferFee Function	MEDIUM	ACKNOWLEDGED
6.2.4	Potential Denial of Service Due to Unbounded Loops in Token and Service Node Unregistration	MEDIUM	FIXED
6.2.5	Lack of Minimum Value Checks in Critical Parameter Setting Functions	MEDIUM	ACKNOWLEDGED
6.2.6	Immediate Parameter Changes Pose Risk of System Instability	MEDIUM	FIXED
6.2.7	Decimals Function Not Part of ERC-20 Standard	LOW	ACKNOWLEDGED



6.2.8	Symbol Function Not Part of ERC-20 Standard	LOW	ACKNOWLEDGED
6.2.9	Use of Single-Step Ownership Transfer Instead of Safer Two-Step Process	LOW	ACKNOWLEDGED
6.2.10	Unnecessary Use of pragma abicoder v2 in PantosRegistryFacet.sol	INFORMATIONAL	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, softstack's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, softstack's experts found **1 High issue** in the code of the smart contract.

6.2.1 Absence of Slashing Mechanism in Staking System

Severity: HIGH

Status: FIXED

File(s) affected: PantosRegistryFacet.sol

Update: <https://github.com/pantos-io/ethereum-contracts/releases/tag/2.0.1>

Attack / Description	The PantosRegistryFacet contract implements a staking system for both tokens and service nodes. However, it lacks a crucial component of many staking systems: a slashing mechanism. Slashing is typically used to penalize malicious or negligent behavior by reducing or confiscating the staked tokens. The absence of such a mechanism could potentially lead to reduced security and accountability within the system.
-----------------------------	---

Code	NA
Result/Recommendation	<p>Implement a slashing mechanism:</p> <ul style="list-style-type: none"> • Add a function to report misbehavior, accessible only to authorized parties (e.g., validators or a governance system). • Create a function to slash a portion of the stake based on the severity of the misbehavior. Modify the stake withdrawal process. • Before allowing withdrawal, check if there are any pending slash requests or if any portion of the stake has been slashed. Add a "slashed" state to the ServiceNodeRecord struct: <pre> struct ServiceNodeRecord { // ... existing fields uint256 slashedAmount; bool isSlashed; } </pre> <p>Implement a slashing function:</p> <pre> function slashServiceNode(address serviceNodeAddress, uint256 slashAmount, string calldata reason) external onlyAuthorized { ServiceNodeRecord storage record = s.serviceNodeRecords[serviceNodeAddress]; require(record.active, "Service node not active"); require(slashAmount <= record.freeStake, "Slash amount exceeds stake"); record.freeStake -= slashAmount; record.slashedAmount += slashAmount; record.isSlashed = true; emit ServiceNodeSlashed(serviceNodeAddress, slashAmount, reason); } </pre>


```
}
```

Modify the withdrawServiceNodeStake function:

```
function withdrawServiceNodeStake(address serviceNodeAddress) external override {  
    // ... existing checks  
    uint256 stake = serviceNodeRecord.freeStake;  
    uint256 slashedAmount = serviceNodeRecord.slashedAmount;  
  
    // Update the service node record  
    serviceNodeRecord.unregisterTime = 0;  
    serviceNodeRecord.freeStake = 0;  
    serviceNodeRecord.slashedAmount = 0;  
  
    // Refund the non-slashed portion of the stake  
    if (stake > slashedAmount) {  
        require(  
            IPantosToken(s.pantosToken).transfer(  
                serviceNodeRecord.unstakingAddress,  
                stake - slashedAmount  
            ),  
            "PantosHub: refund of service node stake failed"  
        );  
    }  
}
```

MEDIUM ISSUES

During the audit, softstack's experts found **5 Medium issues** in the code of the smart contract.

6.2.2 Unsafe ERC Operation in PantosRegistryFacet.sol

Severity: MEDIUM



Status: ACKNOWLEDGED

File(s) affected: PantosRegistryFacet.sol

Update: This applies to the Pantos token which does not follow ERC20.

Attack / Description	The PantosRegistryFacet.sol contract contains multiple instances of unsafe ERC20 token transfer operations. These operations do not check the return value of the transfer and transferFrom functions, which can lead to unexpected behavior if the token transfer fails.
Code	src/facets/PantosRegistryFacet.sol#257 src/facets/PantosRegistryFacet.sol#310 src/facets/PantosRegistryFacet.sol#342 src/facets/PantosRegistryFacet.sol#374 src/facets/PantosRegistryFacet.sol#506 src/facets/PantosRegistryFacet.sol#590 src/facets/PantosRegistryFacet.sol#656 src/facets/PantosRegistryFacet.sol#694
Result/Recommendation	<p>To ensure the safety and reliability of the ERC20 operations, it is recommended to use the OpenZeppelin's SafeERC20 library. This library provides wrappers around the ERC20 operations that throw on failure (when the token contract returns false). Tokens that return no value (and instead revert or throw on failure) are also supported, non-reverting calls are assumed to be successful.</p> <p>Replace unsafe operations with SafeERC20 functions:</p> <pre>IPantosToken(s.pantosToken).safeTransferFrom(...); IPantosToken(s.pantosToken).safeTransfer(msg.sender, stake);</pre>

6.2.3 Potential Division by Zero and Precision Loss in _transferFee Function

Severity: MEDIUM

Status: ACKNOWLEDGED

File(s) affected: PantosForwarder.sol

Update: Requiring a minimum amount on the numerator basically means requiring a minimum fee for the bid. The actual problem here is not the precision loss, but the fact that the validator could work for free.

Attack / Description	<p>The <code>_transferFee</code> function in the <code>PantosForwarder</code> contract performs divisions that can lead to potential issues:</p> <p>Division by Zero: The function does not check for zero values in the input parameters, which can cause the function to revert if zero is passed.</p> <p>Precision Loss: Division by large numbers may result in the result being zero due to Solidity not supporting fractions. This can lead to incorrect fee calculations.</p>
Code	<code>src/PantosForwarder.sol#536</code>
Result/Recommendation	<p>To address these issues, consider the following recommendations:</p> <ul style="list-style-type: none">• Zero-Value Checks: Add checks to ensure that the input parameters are not zero before performing the division.• Minimum Amount Requirement: Require a minimum amount for the numerator to ensure that it is always larger than the denominator.

6.2.4 Potential Denial of Service Due to Unbounded Loops in Token and Service Node Unregistration

Severity: MEDIUM

Status: FIXED

File(s) affected: `PantosRegistryFacet.sol`

Update: <https://github.com/pantos-io/ethereum-contracts/releases/tag/2.0.1>

Attack / Description	<p>The contract contains unbounded loops in the <code>unregisterToken</code> and <code>unregisterServiceNode</code> functions. These loops iterate over the <code>s.tokens</code> and <code>s.serviceNodes</code> arrays respectively. As the number of tokens or service nodes increases, the gas cost of these operations will grow linearly. This could potentially lead to two significant issues:</p>
-----------------------------	--

	<ul style="list-style-type: none"> Extremely high gas costs for users trying to unregister tokens or service nodes. If the arrays grow large enough, the operations might exceed the block gas limit, making it impossible to unregister tokens or service nodes.
Code	<p>Line 297 - 305 (PantosRegistryFacet.sol):</p> <pre> uint numberTokens = s.tokens.length; for (uint i = 0; i < numberTokens; i++) { if (s.tokens[i] == token) { s.tokens[i] = s.tokens[numberTokens - 1]; s.tokens.pop(); break; } } </pre> <p>Line 541 - 548 (PantosRegistryFacet.sol):</p> <pre> uint numberServiceNodes = s.serviceNodes.length; for (uint i = 0; i < numberServiceNodes; i++) { if (s.serviceNodes[i] == serviceNodeAddress) { s.serviceNodes[i] = s.serviceNodes[numberServiceNodes - 1]; s.serviceNodes.pop(); break; } } </pre>
Result/Recommendation	<p>To mitigate this issue, consider implementing one or more of the following solutions:</p> <ul style="list-style-type: none"> Use a mapping to track the index of each token or service node in the array. This would allow for $O(1)$ removal instead of $O(n)$. <pre> mapping(address => uint256) private tokenIndex; </pre>

	<pre>mapping(address => uint256) private serviceNodeIndex;</pre> <ul style="list-style-type: none">• Implement a maximum limit on the number of tokens or service nodes that can be registered.• Instead of removing elements from the array, consider marking them as inactive in a separate mapping. This would avoid the need for array manipulation entirely. <pre>mapping(address => bool) private activeTokens; mapping(address => bool) private activeServiceNodes;</pre> <p>If removal from the array is necessary, consider implementing a batched removal process that can be executed over multiple transactions to avoid hitting the gas limit.</p>
--	---

6.2.5 Lack of Minimum Value Checks in Critical Parameter Setting Functions

Severity: MEDIUM

Status: ACKNOWLEDGED

File(s) affected: PantosRegistryFacet.sol

Update: Each of these values would be set based on the RBAC roles, by a high security role. Considering this, enforcing a minimum value at the contract level seems unnecessary. Especially because they might be 0 if we want to achieve a certain effect.

Attack / Description	<p>The PantosRegistryFacet contract contains several functions for setting critical system parameters. These functions currently lack proper checks to prevent setting values to zero or extremely low values, which could potentially disrupt the system's functionality, economic model, or security.</p> <p>While updateFeeFactor already has a check for newFactor >= 1, the other functions do not have similar safeguards.</p> <p>Proof of Concept:</p> <p>Currently, an owner could set critical parameters to zero or very low values:</p>
-----------------------------	---

	<pre> setMinimumTokenStake(0); setUnbondingPeriodServiceNodeStake(1); // 1 second setMinimumServiceNodeStake(1); // 1 wei setMinimumValidatorFeeUpdatePeriod(0); </pre> <p>These actions could potentially destabilize the system or create security vulnerabilities.</p>
Code	<pre> setMinimumTokenStake setUnbondingPeriodServiceNodeStake setMinimumServiceNodeStake setMinimumValidatorFeeUpdatePeriod </pre>
Result/Recommendation	<p>Implement minimum value checks in all critical parameter setting functions:</p> <p>setMinimumTokenStake:</p> <pre> function setMinimumTokenStake(uint256 minimumTokenStake) public override whenPaused onlyOwner { require(minimumTokenStake > 0, "PantosHub: Minimum token stake must be greater than 0"); s.minimumTokenStake = minimumTokenStake; emit MinimumTokenStakeUpdated(minimumTokenStake); } </pre> <p>setUnbondingPeriodServiceNodeStake:</p> <pre> function setUnbondingPeriodServiceNodeStake(uint256 unbondingPeriodServiceNodeStake) public override onlyOwner { require(unbondingPeriodServiceNodeStake > 0, "PantosHub: Unbonding period must be greater than 0"); s.unbondingPeriodServiceNodeStake = unbondingPeriodServiceNodeStake; emit UnbondingPeriodServiceNodeStakeUpdated(unbondingPeriodServiceNodeStake); } </pre>

	<pre>setMinimumServiceNodeStake: function setMinimumServiceNodeStake(uint256 minimumServiceNodeStake) public override whenPaused onlyOwner { require(minimumServiceNodeStake > 0, "PantosHub: Minimum service node stake must be greater than 0"); s.minimumServiceNodeStake = minimumServiceNodeStake; emit MinimumServiceNodeStakeUpdated(minimumServiceNodeStake); } setMinimumValidatorFeeUpdatePeriod: function setMinimumValidatorFeeUpdatePeriod(uint256 minimumValidatorFeeUpdatePeriod) external override onlyOwner { require(minimumValidatorFeeUpdatePeriod > 0, "PantosHub: Minimum validator fee update period must be greater than 0"); s.minimumValidatorFeeUpdatePeriod = minimumValidatorFeeUpdatePeriod; emit MinimumValidatorFeeUpdatePeriodUpdated(minimumValidatorFeeUpdatePeriod); }</pre>
--	---

6.2.6 Immediate Parameter Changes Pose Risk of System Instability

Severity: MEDIUM

Status: FIXED

File(s) affected: PantosRegistryFacet.sol

Update: <https://github.com/pantos-io/ethereum-contracts/releases/tag/2.0.1>

Attack / Description	The PantosRegistryFacet contract contains several functions that allow the owner to modify critical system parameters. These functions currently apply changes immediately upon execution, which could lead to abrupt alterations in the system's behavior. This immediate effect poses risks to system stability and user experience, as participants are not given time to adjust to new
-----------------------------	--



	parameters.
Code	<pre> setMinimumTokenStake setUnbondingPeriodServiceNodeStake setMinimumServiceNodeStake setMinimumValidatorFeeUpdatePeriod </pre>
Result/Recommendation	<p>Implement a time-delayed change mechanism for critical parameter updates. This approach allows users to prepare for upcoming changes and adjust their positions accordingly. Here's an example implementation for setMinimumTokenStake:</p> <pre> uint256 public constant PARAMETER_UPDATE_DELAY = 7 days; uint256 public pendingMinimumTokenStake; uint256 public minimumTokenStakeUpdateTime; function setMinimumTokenStake(uint256 newMinimumTokenStake) public override whenPaused onlyOwner { require(newMinimumTokenStake > 0 && newMinimumTokenStake <= MAX_TOKEN_STAKE, "Invalid minimum token stake"); pendingMinimumTokenStake = newMinimumTokenStake; minimumTokenStakeUpdateTime = block.timestamp + PARAMETER_UPDATE_DELAY; emit MinimumTokenStakeUpdateScheduled(newMinimumTokenStake, minimumTokenStakeUpdateTime); } function executeMinimumTokenStakeUpdate() public { require(block.timestamp >= minimumTokenStakeUpdateTime, "Update time not reached"); require(pendingMinimumTokenStake > 0, "No pending update"); s.minimumTokenStake = pendingMinimumTokenStake; pendingMinimumTokenStake = 0; minimumTokenStakeUpdateTime = 0; } </pre>


```
emit MinimumTokenStakeUpdated(s.minimumTokenStake);
}
```

Similar patterns should be implemented for other critical parameter update functions. This approach provides several benefits:

- Transparency: Users are informed of upcoming changes through emitted events.
- Preparation Time: The delay allows users to adjust their positions or strategies.
- Cancellation Option: If needed, the owner can cancel a scheduled update before it takes effect.

LOW ISSUES

During the audit, softstack's experts found **3 Low issues** in the code of the smart contract

6.2.7 Decimals Function Not Part of ERC-20 Standard

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: MigrationTokenBurnable.sol, MigrationTokenBurnablePausable.sol, MigrationTokenPausable.sol, BitpandaEcosystemToken.sol

Update: The Pantos token is a superset of ERC20. IPantosToken extends both IERC20 and IBEP20. The latter defines the decimals() function.

Attack / Description

The decimals() function is not part of the original ERC-20 standard. It was introduced later as an optional extension through the IERC20Metadata interface. As a result, some valid ERC-20 tokens do not support this interface. Blindly casting all tokens to this interface and calling the decimals() function can lead to unexpected behavior or failures.

Code	src/migrations/MigrationTokenBurnable.sol#40 src/migrations/MigrationTokenBurnablePausable.sol#43 src/migrations/MigrationTokenPausable.sol#41 src/BitpandaEcosystemToken.sol#93 src/BitpandaEcosystemToken.sol#105 Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/IERC20Metadata.sol
Result/Recommendation	<p>To handle tokens that do not implement the decimals() function, it is recommended to:</p> <ul style="list-style-type: none"> • Check for Interface Support: Use the ERC165 standard to check if the token supports the IERC20Metadata interface before calling the decimals() function. • Fallback Mechanism: Implement a fallback mechanism for tokens that do not support the decimals() function. For example, assume a default value or handle the absence of the function gracefully. <p>Example Code</p> <p>Here is an example of how to modify the code to check for interface support and handle tokens that do not implement the decimals() function:</p> <pre>// Import the necessary interfaces import "@openzeppelin/contracts/token/ERC20/IERC20.sol"; import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol"; import "@openzeppelin/contracts/utils/introspection/ERC165Checker.sol"; contract MigrationTokenBurnable { using ERC165Checker for address; function getTokenDecimals(address token) internal view returns (uint8) { // Check if the token supports the IERC20Metadata interface if (token.supportsInterface(type(IERC20Metadata).interfaceId)) {</pre>

	<pre> return IERC20Metadata(token).decimals(); } else { // Handle the case where the token does not support the decimals function // For example, assume a default value of 18 decimals return 18; } } function someFunction(address token) external { uint8 decimals = getTokenDecimals(token); // Use the decimals value in your logic } } </pre>
--	--

6.2.8 Symbol Function Not Part of ERC-20 Standard

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: MigrationTokenBurnable.sol, MigrationTokenBurnablePausable.sol, MigrationTokenPausable.sol, PantosBaseToken.sol

Update: Similar comments to 6.2.9. See IPantosToken, which extends both IERC20 and IBEP20. The latter defines the symbol() function.

Attack / Description	The symbol() function is not part of the original ERC-20 standard. It was introduced later as an optional extension through the IERC20Metadata interface. Consequently, some valid ERC-20 tokens do not support this interface. Blindly casting all tokens to this interface and calling the symbol() function can lead to unexpected behavior or failures.
Code	src/migrations/MigrationTokenBurnable.sol#52 src/migrations/MigrationTokenBurnablePausable.sol#55 src/migrations/MigrationTokenPausable.sol#53

	src/PantosBaseToken.sol#101
Result/Recommendation	<p>To handle tokens that do not implement the symbol() function, it is recommended to:</p> <ul style="list-style-type: none"> • Check for Interface Support: Use the ERC165 standard to check if the token supports the IERC20Metadata interface before calling the symbol() function. • Fallback Mechanism: Implement a fallback mechanism for tokens that do not support the symbol() function. For example, handle the absence of the function gracefully or provide a default value. <p>Example Code Here is an example of how to modify the code to check for interface support and handle tokens that do not implement the symbol() function:</p> <pre>// Import the necessary interfaces import "@openzeppelin/contracts/token/ERC20/IERC20.sol"; import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol"; import "@openzeppelin/contracts/utils/introspection/ERC165Checker.sol"; contract MigrationTokenBurnable { using ERC165Checker for address; function getTokenSymbol(address token) internal view returns (string memory) { // Check if the token supports the IERC20Metadata interface if (token.supportsInterface(type(IERC20Metadata).interfaceId)) { return IERC20Metadata(token).symbol(); } else { // Handle the case where the token does not support the symbol function // For example, return a default value or handle it gracefully return "UNKNOWN"; } } }</pre>

	<pre>function someFunction(address token) external { string memory symbol = getTokenSymbol(token); // Use the symbol value in your logic }</pre>
--	--

6.2.9 Use of Single-Step Ownership Transfer Instead of Safer Two-Step Process

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: PantosBaseToken.sol, PantosForwarder.sol

Attack / Description	The PantosBaseToken contract currently uses the Ownable contract from OpenZeppelin for ownership management. This implementation allows for a single-step ownership transfer, which can be less secure. A more secure approach is to use the Ownable2Step contract, which implements a two-step ownership transfer process. This process requires the new owner to accept the ownership transfer, ensuring that the new owner address is valid and active.
Code	src/PantosBaseToken.sol#6 src/PantosForwarder.sol#6
Result/Recommendation	To enhance the security of the ownership transfer process, it is recommended to use the Ownable2Step contract from OpenZeppelin. This contract provides a two-step ownership transfer process, which is more secure. <ul style="list-style-type: none">• Import Ownable2Step: Replace the import of Ownable with Ownable2Step.• Use Ownable2Step Functions: Utilize the transferOwnership and acceptOwnership functions provided by Ownable2Step.

INFORMATIONAL ISSUES

During the audit, softstack’s experts found 1 **Informational issue** in the code of the smart contract.



6.2.10 Unnecessary Use of pragma abicoder v2 in PantosRegistryFacet.sol

Severity: INFORMATIONAL

Status: FIXED

File(s) affected: PantosRegistryFacet.sol

Update: <https://github.com/pantos-io/ethereum-contracts/releases/tag/2.0.1>

Attack / Description	The PantosRegistryFacet.sol file includes the directive pragma abicoder v2. However, as of Solidity 0.6.0, the ABI coder v2 is considered non-experimental, and starting with Solidity 0.8.0, it is enabled by default. Therefore, explicitly including pragma abicoder v2 is redundant and unnecessary. This can lead to confusion and clutter in the codebase without providing any functional benefit.
Code	Line 4 (PantosRegistryFacet.sol): pragma abicoder v2;
Result/Recommendation	<p>Remove the pragma abicoder v2 directive from the PantosRegistryFacet.sol file to clean up the code and avoid redundancy. The updated section of the file should look like this:</p> <pre>// SPDX-License-Identifier: GPL-3.0 // slither-disable-next-line solc-version pragma solidity 0.8.26; import {PantosTypes} from "../interfaces/PantosTypes.sol"; import {IPantosForwarder} from "../interfaces/IPantosForwarder.sol"; import {IPantosToken} from "../interfaces/IPantosToken.sol"; import {IPantosRegistry} from "../interfaces/IPantosRegistry.sol"; import {PantosBaseFacet} from "../PantosBaseFacet.sol"; import {PantosHubStorage} from "../PantosHubStorage.sol";</pre> <p>By removing the unnecessary pragma abicoder v2 directive, the code will be cleaner and easier to maintain, without any loss of functionality.</p>

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓


ID	Title	Relationships	Test Result
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✓

ID	Title	Relationships	Test Result
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓




6.4 Verify Claims


6.4.1 Compliance with Best Practices: The audit should ensure that the contracts adhere to smart contract best practices, including checking for common vulnerabilities such as reentrancy attacks and overflow/underflow issues.

Status: tested and verified 


6.4.2 Proper Access Control: The contracts should implement and enforce proper access control mechanisms to ensure that only authorized entities can execute sensitive functions.

Status: tested and verified 


6.4.3 Efficient Gas Usage: The contracts should be optimized for efficient gas usage, minimizing unnecessary computations and storage operations to reduce transaction costs.

Status: tested and verified 

6.4.4 Data Integrity and Consistency: The contracts should maintain data integrity and consistency, ensuring that all state changes are accurately reflected and that no data is inadvertently lost or corrupted.

Status: tested and verified 

6.4.5 Upgradeability and Flexibility: The contracts should be designed with upgradeability and flexibility in mind, allowing for future enhancements and modifications without disrupting the existing functionality and user experience.

Status: tested and verified 

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the smart contract codebase provided by the Pantos team. The main objective of the audit was to verify the security and functionality claims of the smart contract. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of one issue, classified as follows:

- No critical issues were found.
- 1 high severity issues were found.
- 5 medium severity issues were found.
- 3 low severity issues were discovered
- 1 informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, CWE classifications, and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes. Based on the nature of the finding and adherence to the business logic, we recommend that the Pantos team review the suggestions.

8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over \$100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: <https://softstack.io>

How We Work



1 -----

PREPARATION

Supply our team with audit ready code and additional materials



2 -----

COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



3 -----

AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



4 -----

FIXES

Your development team applies fixes while consulting with our auditors on their safety.



5 -----

REPORT

We check the applied fixes and deliver a full report on all steps done.

