Pantos

Service & Validator Node
CODEBASE AND ARCHITECTURE
SECURITY AUDIT

21.10.2024

Made in Germany by Softstack.io

Table of contents

1. Disclaimer

The audit makes no statements or warrantees 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.

Major Versions / Date	Description
0.1 (15.07.2024)	Layout
0.4 (25.07.2024)	Setup Testing Environment
0.5 (27.07.2024)	Automated Security Testing
0.6 (30.07.2024)	Manual Security Testing
0.9 (01.08.2024)	Summary and Recommendation
1.0 (05.08.2024)	Final document
1.1 (18.10.2024)	Re-check https://github.com/pantos-io/servicenode/releases/tag/1.8.2 & https://github.com/pantos-io/validatornode/releases/tag/1.8.3

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 codebase functioning in a number of scenarios, or creates a risk that the codebase may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using the codebase, or provides the opportunity to use the codebase 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 codebase 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 codebase 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 pen-testers 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 codebase.
   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 codebase 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 codebase.

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)
./servicenode-1.8.1/pantos-service-node-worker.sh	2bb0eea4a012dc4dcf2f687196965887
./servicenode-1.8.1/pantos/servicenode/database/enums.py	73810cfa7cf77e6100d13c105ac6e32b
./servicenode-1.8.1/pantos/servicenode/database/models.py	d85c801367e91eba690a384d7924c1a8
./servicenode-1.8.1/pantos/servicenode/database/access.py	751bb05498b92d40cfb777e63aca7528
./servicenode-1.8.1/pantos/servicenode/database/__init__.py	3f914a9217cba07112343e273b1aeb9f
./servicenode-1.8.1/pantos/servicenode/database/exceptions.py	3e99a8f77eae7daed4116e4f05a94245
./servicenode-1.8.1/pantos/servicenode/configuration.py	53462d42de18495b1a2729a0097e27bc
./servicenode-1.8.1/pantos/servicenode/plugins/bids.py	3e2745213c5314dc73e7af44ac71e521
./servicenode-1.8.1/pantos/servicenode/plugins/__init__.py	919c73f39c0381056fb20736ed70fc31

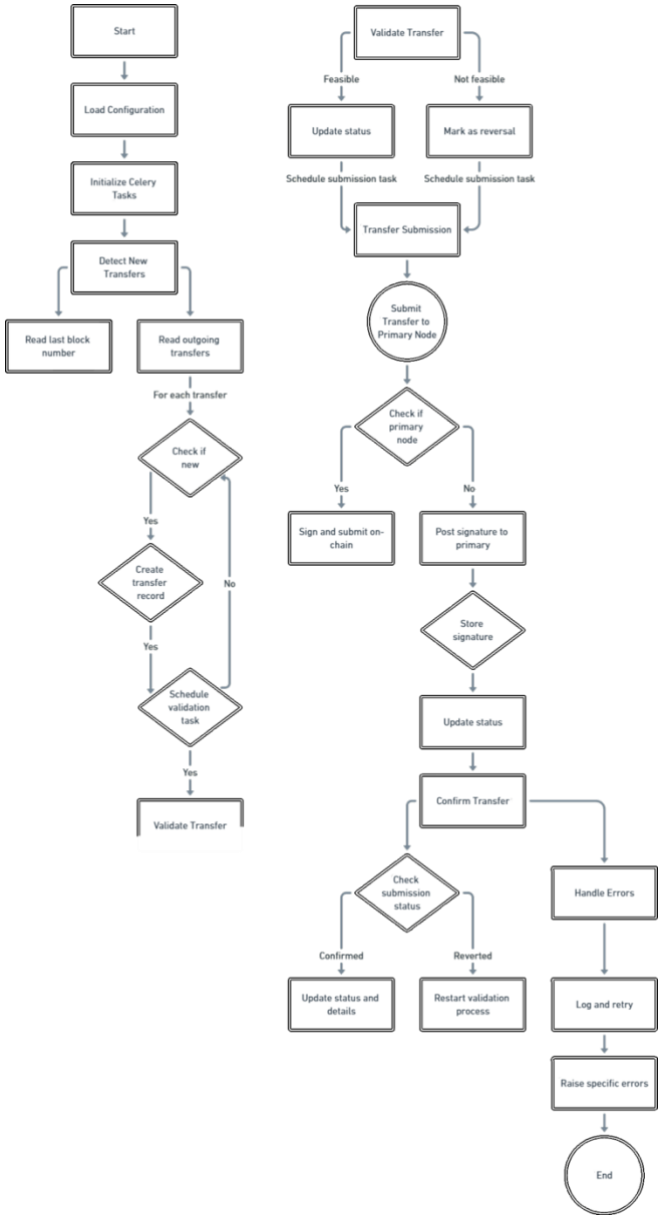
./servicenode-1.8.1/pantos/servicenode/plugins/base.py	a70da90c5d1858c44bee7f87a4606b06
./servicenode-1.8.1/pantos/servicenode/business/plugins.py	512229e14fbafcc3f09525d0a01b26b6
./servicenode-1.8.1/pantos/servicenode/business/transfers.py	f60d8e4f0aaa905107a8986c4ea56e25
./servicenode-1.8.1/pantos/servicenode/business/bids.py	fc33c116f81ca899ede5ab4ad034e05d
./servicenode-1.8.1/pantos/servicenode/business/__init__.py	b079525ef8a61f7673ca02fc5b53b346
./servicenode-1.8.1/pantos/servicenode/business/node.py	f20d08d7990818e8c0fadd17cf7eed94
./servicenode-1.8.1/pantos/servicenode/business/base.py	99ef997af97f5e943e6d40ecaabab6a
./servicenode-1.8.1/pantos/servicenode/__init__.py	149b5a61900625a580c9bc827bbab217
./servicenode-1.8.1/pantos/servicenode/application.py	97083101ee4d9851bdf375f3d9dd0736
./servicenode-1.8.1/pantos/servicenode/restapi.py	5de6ebf869c444ac0508a933e3292515
./servicenode-1.8.1/pantos/servicenode/celery.py	688156abb6a457bd206ddc60d1e533c9
./servicenode-1.8.1/pantos/servicenode/exceptions.py	4e6293af78fd37902568cafe2d28e391
./servicenode-1.8.1/pantos/servicenode/__main__.py	a2aea418d5a7a543c7c136578f1dad3a
./servicenode-1.8.1/pantos/servicenode/blockchains/avalanche.py	121b28dd0498d5b8aba96771cbcd5d08
./servicenode-1.8.1/pantos/servicenode/blockchains/solana.py	46ef522d865d49702be61819d1bbcb46
./servicenode-1.8.1/pantos/servicenode/blockchains/polygon.py	09d68898687553924e1b86add7fe75e8
./servicenode-1.8.1/pantos/servicenode/blockchains/__init__.py	1b1dd30682f5869ee21256c0d94351bd
./servicenode-1.8.1/pantos/servicenode/blockchains/factory.py	e572c822349f000be44ccfc6c1b1fb9e
./servicenode-1.8.1/pantos/servicenode/blockchains/bnbchain.py	8eb4735a658ad1b89506e0171e4aa8c0

./servicenode-1.8.1/pantos/servicenode/blockchains/cronos.py	6bb49e1b62be5332795a548d666236f1
./servicenode-1.8.1/pantos/servicenode/blockchains/celo.py	0ff4951a8761421c406de891f72e6055
./servicenode-1.8.1/pantos/servicenode/blockchains/ethereum.py	c5ff4776ea8caa6bafec5dab1b443345
./servicenode-1.8.1/pantos/servicenode/blockchains/fantom.py	bd79da24147552821764ba4c19b29dea
./servicenode-1.8.1/pantos/servicenode/blockchains/base.py	6f4adde087338023e20554e07f8d5ba5
./servicenode-1.8.1/pantos/servicenode/wsgi.py	af409fdb8eb13e51bc9a5dab4e56be67
./servicenode-1.8.1/bids.yml	6bec969a88091ed40ec46aca6009177e
./servicenode-1.8.1/service-node-config.docker.env	2cc50a03eaeaa33de1e576dd246968ac
./servicenode-1.8.1/pantos-service-node.sh	f8b12d4b2d0991eb7455e61a91da62c2
./validatornode-1.8.2/pantos-validator-node-worker.sh	ba51918ac73db4c453339927e71e941b
./validatornode-1.8.2/pantos/validatornode/database/enums.py	f784028363e674842cee5fead548c6e6
./validatornode-1.8.2/pantos/validatornode/database/models.py	49ad76942c9785444fcb322ccc2a165c
./validatornode-1.8.2/pantos/validatornode/database/access.py	5ae022ee392510e9cd9b2303e92487fc
./validatornode-1.8.2/pantos/validatornode/database/__init__.py	8712b01ca9c4db255d975222749d9f1e
./validatornode-1.8.2/pantos/validatornode/database/exceptions.py	7174db3bc8501dc5fbc5cf155622a1c5
./validatornode-1.8.2/pantos/validatornode/configuration.py	058d2a4b2c840497f939e059652d56a2
./validatornode-1.8.2/pantos/validatornode/monitor.py	ec4eabaede0b3cf721159d2106027d9e
./validatornode-1.8.2/pantos/validatornode/business/transfers.py	f56fb0e59ade28cb81c0ccce9c986094
./validatornode-1.8.2/pantos/validatornode/business/__init__.py	b079525ef8a61f7673ca02fc5b53b346

./validatornode-1.8.2/pantos/validatornode/business/signatures.py	aa84fd2d3b55c02e4fb8b77732095ba8
./validatornode-1.8.2/pantos/validatornode/business/base.py	aab54213c472f6b990f01d2356d4b508
./validatornode-1.8.2/pantos/validatornode/__init__.py	3c5c8f6b3c0ab389a54697cf78666fef
./validatornode-1.8.2/pantos/validatornode/application.py	fb1c8be7aa7bce892278791e783678ab
./validatornode-1.8.2/pantos/validatornode/restapi.py	355b5d89b812f381b1fab1f7b633a635
./validatornode-1.8.2/pantos/validatornode/celery.py	cb45b7dc3d88671fb5fd0f3683747f0c
./validatornode-1.8.2/pantos/validatornode/exceptions.py	37466b127e6ddde95400944cff029678
./validatornode-1.8.2/pantos/validatornode/entities.py	d694740b610949b5c620738644626816
./validatornode-1.8.2/pantos/validatornode/__main__.py	901299b400e42225fac167e13f4116e1
./validatornode-1.8.2/pantos/validatornode/blockchains/avalanche.py	2e6b21745dd980f2ae5952bdc9fd9fc4
./validatornode-1.8.2/pantos/validatornode/blockchains/solana.py	5597db800af775101a55b85b59646b36
./validatornode-1.8.2/pantos/validatornode/blockchains/polygon.py	a062a83decaf43397e33547d2773c833
./validatornode-1.8.2/pantos/validatornode/blockchains/__init__.py	1b1dd30682f5869ee21256c0d94351bd
./validatornode-1.8.2/pantos/validatornode/blockchains/factory.py	fed0bc565369e9aa6663a2115a24caab
./validatornode-1.8.2/pantos/validatornode/blockchains/bnbchain.py	326fe867eeedd054aa8fb7ac9e24675d
./validatornode-1.8.2/pantos/validatornode/blockchains/cronos.py	9680e0fed0f998fe0bcb8b9c17e55ef4
./validatornode-1.8.2/pantos/validatornode/blockchains/celo.py	5ed0d8be0948e8d445cef3c3acd0d1b
./validatornode-1.8.2/pantos/validatornode/blockchains/ethereum.py	317a1d25a8a9fbf569a5c06f04f2a093

./validatornode-1.8.2/pantos/validatornode/blockchains/fantom.py	3ba4b9b3751950e35d1ce701e713f8e9
./validatornode-1.8.2/pantos/validatornode/blockchains/base.py	05457a92a85a39163c90883048a58f83
./validatornode-1.8.2/pantos/validatornode/restclient.py	b4c3196166e87b1661a2efe56253b44b
./validatornode-1.8.2/pantos/validatornode/wsgi.py	ce026ff0ce91527ba989e4532bdc7d67

5.2 Flows


Start
Load Configuration
Initialize Celery Tasks
Detect New Transfers
Read last block number
Read outgoing transfers
For each transfer
Check if new
Yes
No
Create transfer record
Yes
No
Schedule validation task
Yes
Validate Transfer
Validate Transfer
Feasible
Not feasible
Update status
Mark as reversal
Schedule submission task
Schedule submission task
Transfer Submission
Submit Transfer to Primary Node
Check if primary node
Yes
No
Sign and submit on-chain
Post signature to primary
Store signature
Update status
Confirm Transfer
Check submission status
Confirmed
Reverted
Update status and details
Restart validation process
Handle Errors
Log and retry
Raise specific errors
End

5.3 Source Unites in Scope

Path: ./validatornode-1.8.2/pantos/

language	files	code	comment	blank	total
Python	39	5,075	121	853	6,049

Directories

path	files	code	comment	blank	total
validatornode	39	5,075	121	853	6,049
validatornode (Files)	11	1,089	10	173	1,272
validatornode/blockchains	11	1,362	55	275	1,692
validatornode/business	4	1,126	27	142	1,295
validatornode/database	13	1,498	29	263	1,790
validatornode/database (Files)	5	1,229	12	203	1,444
validatornode/database/migrations	8	269	17	60	346
validatornode/database/migrations (Files)	2	47	0	19	66
validatornode/database/migrations/versions	6	222	17	41	280

Path: ./servicenode-1.8.1/pantos/servicenode

language	files	code	comment	blank	total
Python	40	4,395	122	781	5,298

Directories

path	files	code	comment	blank	total
blockchains	11	1,085	49	241	1,375
business	6	1,074	12	150	1,236
database	12	1,185	43	219	1,447
database (Files)	5	863	23	167	1,053
database/migrations	7	322	20	52	394
database/migrations (Files)	2	47	0	19	66
database/migrations/versions	5	275	20	33	328
plugins	3	211	3	51	265

6. Scope of Work

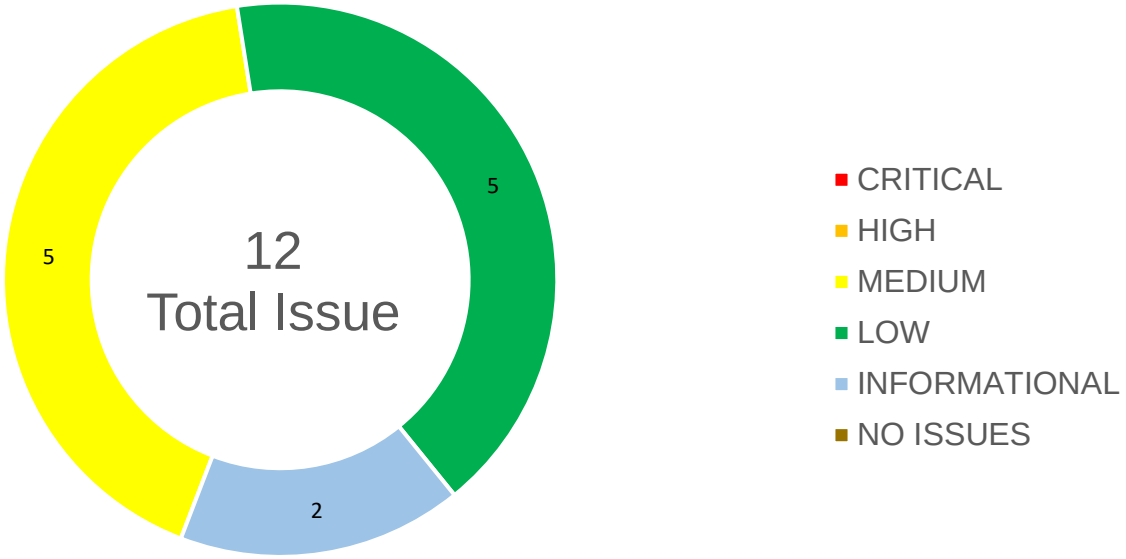
The Pantos Team provided us with the files that needs to be tested. The scope of the audit is the service & validator node codebase and architecture.

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

1. Proper Authentication and Authorization Mechanisms: Ensure only authorized users can access APIs and initiate/validate cross-chain transfers.
2. Secure Handling of Private Keys: Verify that private keys are securely stored, encrypted, and access is strictly controlled and logged. Input Validation and Sanitization: Ensure all user inputs and data are validated and sanitized to protect against injections, XSS, and CSRF.
3. Comprehensive Logging and Monitoring: Verify that critical actions and events are logged, securely stored, monitored, and alerts for suspicious activities are in place.
4. Codebase Adheres to Best Practices and Checks for Common Vulnerabilities: Ensure the codebase adheres to best practices, is free of common vulnerabilities, and follows secure coding standards.

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	Error Handling in initialize_package	MEDIUM	ACKNOWLEDGED
6.2.2	Concurrency Bug in _create_instance Method	MEDIUM	FIXED
6.2.3	Non-Unique Constraints Handling	MEDIUM	FIXED
6.2.4	Debug Mode in Production	MEDIUM	FIXED
6.2.5	Queue Purging at Startup Could Lead to Data Loss	MEDIUM	INVALID BY DESIGN
6.2.6	Broad Exception Handling in get_cross_blockchain_bids Method	LOW	ACKNOWLEDGED
6.2.7	Insufficient Log Message Context	LOW	FIXED
6.2.8	Broad Exception Handling in update_node_registrations	LOW	ACKNOWLEDGED
6.2.9	Error Handling during Table Initialization	LOW	ACKNOWLEDGED

6.2.10	is_main_module Function Might Not Reliably Detect All Celery Worker Processes	LOW	FIXED
6.2.11	Unsafe Assumption in read_transfer_nonce Function	INFORMATIONAL	FIXED
6.2.12	Replace Assertions with Runtime Type Checking	INFORMATIONAL	ACKNOWLEDGED

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 **no High issues** in the code of the smart contract.

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

6.2.1 Error Handling in initialize_package
Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: servicenode/database/_init_.py
Update: If you consider the function as a standalone unit, the comment is fair. We could implement the suggestion in order to avoid misusing it.

Attack / Description	If the initialize_package() function encounters an error, the session might still get initialized incorrectly. There is no rollback mechanism or logging to handle such cases.

Code	Line 93 - 127 (_init_.py): `def initialize_package(is_flask_app: bool = False) -> None:` `  # Before connecting, run alembic to ensure` `  # the database schema is up to date` `  if is_flask_app and config['database']['apply_migrations']:` `  run_migrations(config['database']['alembic_config'],` `  config['database']['url'])` `  global _sql_engine` `  _sql_engine = sqlalchemy.create_engine(` `  config['database']['url'], pool_size=config['database']['pool_size'],` `  max_overflow=config['database']['max_overflow'], pool_pre_ping=True,` `  echo=config['database']['echo'])` `  global _session_maker` `  _session_maker = sqlalchemy.orm.sessionmaker(bind=_sql_engine)` `  # Initialize the tables` `  with _session_maker.begin() as session:` `  assert isinstance(session, Session)  # type hint` `  # Blockchain table` `  statement = sqlalchemy.select(sqlalchemy.func.max(Blockchain_.id))` `  max_blockchain_id = session.execute(statement).scalar_one_or_none()` `  for blockchain in sorted(Blockchain):` `  if (max_blockchain_id is None`

```
            or max_blockchain_id < blockchain.value):
        session.add(
            Blockchain_(id=blockchain.value, name=blockchain.name))
    # Transfer status table
    statement = sqlalchemy.select(sqlalchemy.func.max(TransferStatus_.id))
    max_transfer_status_id = session.execute(
        statement).scalar_one_or_none()
    for transfer_status in sorted(TransferStatus):
        if (max_transfer_status_id is None
            or max_transfer_status_id < transfer_status.value):
        session.add(
            TransferStatus_(id=transfer_status.value,
                            name=transfer_status.name))
```

Result/Recommendation

Add exception handling and rollback mechanisms to ensure that any errors encountered during initialization are properly logged and handled. Updated implementation:

```
def initialize_package(is_flask_app: bool = False) -> None:
    try:
        if is_flask_app and config['database']['apply_migrations']:
            run_migrations(config['database']['alembic_config'],
                           config['database']['url'])

        global _sql_engine
        _sql_engine = sqlalchemy.create_engine(
            config['database']['url'], pool_size=config['database']['pool_size'],
            max_overflow=config['database']['max_overflow'], pool_pre_ping=True,
            echo=config['database']['echo'])
        global _session_maker
        _session_maker = sqlalchemy.orm.sessionmaker(bind=_sql_engine)
```

```
        # Initialize the tables
        with _session_maker.begin() as session:
            assert isinstance(session, Session)  # type hint
            # Blockchain table
            statement = sqlalchemy.select(sqlalchemy.func.max(Blockchain_.id))
            max_blockchain_id = session.execute(statement).scalar_one_or_none()
            for blockchain in sorted(Blockchain):
                if (max_blockchain_id is None
                        or max_blockchain_id < blockchain.value):
                    session.add(
                        Blockchain_(id=blockchain.value, name=blockchain.name))
            # Transfer status table
            statement = sqlalchemy.select(sqlalchemy.func.max(TransferStatus_.id))
            max_transfer_status_id = session.execute(
                statement).scalar_one_or_none()
            for transfer_status in sorted(TransferStatus):
                if (max_transfer_status_id is None
                        or max_transfer_status_id < transfer_status.value):
                    session.add(
                        TransferStatus_(id=transfer_status.value,
                                        name=transfer_status.name))

    except Exception as e:
        _logger.critical('Error initializing package: %s', str(e), exc_info=True)
        if _sql_engine:
            _sql_engine.dispose()
        raise
```

6.2.2 Concurrency Bug in _create_instance Method

Severity: MEDIUM

Status: FIXED

File(s) affected: servicenode/database/access.py
Update: https://github.com/pantos-io/servicenode/releases/tag/1.8.2

Attack / Description	The Pantos service node ecosystem uses various database operations to manage blockchains, contracts, bids, and transfers. While the _create_instance method is designed to be thread-safe, other database operations (like create, read, update, delete) do not leverage the same locking mechanism, leading to concurrency concerns. The _create_instance method in the database module uses threading locks to create new instances of database models in a thread-safe manner. However, it does not guarantee thread-safe access across other methods that may simultaneously interact with the same database tables. This creates a risk of race conditions and potential data inconsistencies, especially when concurrent tasks (e.g., Celery tasks) are involved. The impact are potential data corruption due to race conditions. Possible integrity violations where concurrent access to the database isn't adequately serialized. Inconsistent application state which may lead to unexpected behaviors.
Code	Line 483 - 499 (access.py): `def _create_instance(model: Base, lock: threading.Lock,` `**kwargs: typing.Any) -> int:` `"""Create a new model instance in a thread-safe manner.` `"""` `with lock:` `with get_session_maker().begin() as session:` `# New session necessary to allow committing after the new` `# model instance has been added (flush is not sufficient in` `# a multithreaded environment)` `instance = session.query(model).filter_by(**kwargs).one_or_none()`

```
        if instance is None:
            # Instance has been added by another thread in between
            instance = model(**kwargs)
            session.add(instance)
            session.flush()
        return instance.id
```

Result/Recommendation

Ensure that thread-safe mechanisms are applied consistently across all database operations. This can include:

- Utilizing threading.Lock in other database methods as well.
- Implement a broader transaction management strategy, employing SQLAlchemy's session management effectively.

```
# Example of adding locks to other methods
_bid_lock = threading.Lock()
_transfer_lock = threading.Lock()

def create_bid(source_blockchain: Blockchain,
               destination_blockchain: Blockchain, execution_time: int,
               valid_until: int, fee: int) -> None:
    with _bid_lock:
        bid = Bid(source_blockchain_id=source_blockchain.value,
                  destination_blockchain_id=destination_blockchain.value,
                  execution_time=execution_time, valid_until=valid_until, fee=fee)
        with get_session_maker().begin() as session:
            session.add(bid)
            session.flush()
```

```
def create_transfer(source_blockchain: Blockchain, destination_blockchain: Blockchain,
sender_address: str,
                    recipient_address: str, source_token_address: str, destination_token_address: str,
amount: int, fee: int,
                    sender_nonce: int, signature: str, hub_address: str, forwarder_address: str) -> int:
    with _transfer_lock:
        try:
            with get_session_maker().begin() as session:
                source_token_contract = _read_token_contract(session,
blockchain_id=source_blockchain.value, address=source_token_address)
                source_token_contract_id = (source_token_contract.id if source_token_contract is not
None else _create_token_contract(blockchain_id=source_blockchain.value,
address=source_token_address))
                destination_token_contract = _read_token_contract(session,
blockchain_id=destination_blockchain.value, address=destination_token_address)
                destination_token_contract_id = (destination_token_contract.id if
destination_token_contract is not None else
_create_token_contract(blockchain_id=destination_blockchain.value,
address=destination_token_address))
                # More logic...
                transfer = Transfer(source_blockchain_id=source_blockchain.value,
                                    destination_blockchain_id=destination_blockchain.value,
                                    sender_address=sender_address, recipient_address=recipient_address,
                                    source_token_contract_id=source_token_contract_id,
                                    destination_token_contract_id=destination_token_contract_id,
amount=amount, fee=fee,
                                    sender_nonce=sender_nonce, signature=signature,
hub_contract_id=hub_contract_id,
                                    forwarder_contract_id=forwarder_contract_id,
status_id=TransferStatus.ACCEPTED.value)
                session.add(transfer)
                session.flush()
```

	return int(transfer.id) except sqlalchemy.exc.IntegrityError as e: session.rollback() if UNIQUE_SENDER_NONCE_CONSTRAINT in str(e): raise SenderNonceNotUniqueError(source_blockchain, sender_address, sender_nonce) raise

6.2.3 Non-Unique Constraints Handling

Severity: MEDIUM
Status: FIXED
File(s) affected: servicenode/database/access.py
Update: https://github.com/pantos-io/servicenode/releases/tag/1.8.2

Attack / Description	There is a bug in the handling of UNIQUE_SENDER_NONCE_CONSTRAINT in the create_transfer function where, on encountering an IntegrityError, the session is rolled back but not restarted correctly. This can leave the session in an inconsistent state, potentially leading to further issues or errors. When attempting to create a transfer record and encountering a unique constraint violation for the sender_nonce, the current implementation rolls back the session but does not restart or cleanly handle the session afterward. This improper session management can result in an inconsistent or unstable state.
Code	Line 195 - 200 (access.py): `except sqlalchemy.exc.IntegrityError as e:` `session.rollback()` `if UNIQUE_SENDER_NONCE_CONSTRAINT in str(e):` `raise SenderNonceNotUniqueError(source_blockchain, sender_address,` `sender_nonce)` `raise`

Result/Recommendation	Ensure that after a session rollback, the session is properly restarted or closed to maintain a consistent state. def create_transfer(source_blockchain: Blockchain, destination_blockchain: Blockchain, sender_address: str, recipient_address: str, source_token_address: str, destination_token_address: str, amount: int, fee: int, sender_nonce: int, signature: str, hub_address: str, forwarder_address: str) -> int: """Create a transfer database record.""" session = get_session_maker()() try: with session.begin(): # ... [creation logic] ... transfer = Transfer(source_blockchain_id=source_blockchain.value, destination_blockchain_id=destination_blockchain.value, sender_address=sender_address, recipient_address=recipient_address, source_token_contract_id=source_token_address, destination_token_contract_id=destination_token_address, amount=amount, fee=fee, sender_nonce=sender_nonce, signature=signature, hub_contract_id=hub_address, forwarder_contract_id=forwarder_address, status_id=TransferStatus.ACCEPTED.value) session.add(transfer) session.flush() return int(transfer.id) except sqlalchemy.exc.IntegrityError as e: session.rollback()

	if UNIQUE_SENDER_NONCE_CONSTRAINT in str(e): raise SenderNonceNotUniqueError(source_blockchain, sender_address, sender_nonce) raise finally: # Ensure the session is cleanly handled after rollback session.close() # Optionally, start a new session if needed for subsequent operations

6.2.4 Debug Mode in Production

Severity: MEDIUM
Status: FIXED
File(s) affected: servicenode/__main__.py, validatornode/__main__.py
Update: The built-in Flask web server is never used in production environments. We use the wsgi.py entry point instead.

Attack / Description	The debug mode for the Flask application is directly controlled by a configuration flag. This poses a security risk if the debug flag is inadvertently set to True in a production environment. Debug mode should never be enabled in production, as it can expose sensitive information and allow for unauthorized access to the application's internals. Potential Impact: • Exposure of sensitive information through detailed error messages. • Increased risk of unauthorized access to the application's internal state. • Possibilities for an attacker to leverage debug mode to identify and exploit vulnerabilities.
Code	Line 1- 18 (main.py): """Entry point for running the Pantos service node application in Flask's built-in web server.

```
"""
from pantos.servicenode.application import create_application
from pantos.servicenode.configuration import config

if __name__ == '__main__':
    application = create_application()
    host = config['application']['host']
    port = config['application']['port']
    ssl_certificate = config['application'].get('ssl_certificate')
    ssl_private_key = config['application'].get('ssl_private_key')
    ssl_context = (None if ssl_certificate is None else
                   (ssl_certificate, ssl_private_key))
    debug = config['application']['debug']
    application.run(host=host, port=port, ssl_context=ssl_context, debug=debug,
                    use_reloader=False)
```

Line 1 – 18 (main.py):

```
"""Entry point for running the Pantos Validator Node application in
Flask's built-in web server.

"""
from pantos.validatornode.application import create_application
from pantos.validatornode.configuration import config
```

	`` if __name__ == '__main__': application = create_application() host = config['application']['host'] port = config['application']['port'] ssl_certificate = config['application'].get('ssl_certificate') ssl_private_key = config['application'].get('ssl_private_key') ssl_context = (None if ssl_certificate is None else (ssl_certificate, ssl_private_key)) debug = config['application']['debug'] application.run(host=host, port=port, debug=debug, ssl_context=ssl_context, use_reloader=False) ```
Result/Recommendation	Override the debug configuration in production to always be False. Add a check to ensure that the application does not run in debug mode if it's deployed in a production environment. """Entry point for running the Pantos service node application in Flask's built-in web server. """ from pantos.servicenode.application import create_application from pantos.servicenode.configuration import config if __name__ == '__main__': application = create_application() host = config['application']['host'] port = config['application']['port'] ssl_certificate = config['application'].get('ssl_certificate') ssl_private_key = config['application'].get('ssl_private_key')

	ssl_context = (None if ssl_certificate is None else (ssl_certificate, ssl_private_key)) # Force debug to False in production environment = config['application'].get('environment', 'development') debug = config['application']['debug'] if environment != 'production' else False application.run(host=host, port=port, ssl_context=ssl_context, debug=debug, use_reloader=False) Alternative Solution Introduce an additional configuration setting to differentiate between environments (e.g., development, staging, production) and enforce debug=False in non-development environments.

6.2.5 Queue Purging at Startup Could Lead to Data Loss

Severity: MEDIUM
Status: INVALID BY DESIGN
File(s) affected: servicenode/celery.py
Update: We should always purge this queue at startup. The suggestion doesn't take the context into account, as we always populate the bids queue immediately after clearing it. This issue is invalid, as it is a design choice. The confusion arose from a lack of context/documentation provided during the audit.

Attack / Description	The current implementation of the Celery initialization module includes a step to purge the bids queue at startup. This action could lead to the loss of important data if the application restarts unexpectedly, thereby impacting the reliability and integrity of the application's processing capabilities. Impact: • Loss of queued tasks in the bids queue. • Potential data integrity and processing reliability issues. • Unexpected restart or crash recovery could lead to incomplete processing of tasks.

Code	Line 58 - 65 (celery.py): `if is_main_module():  # pragma: no cover` `  # purge the bids queue at startup` `  with celery_app.connection_for_write() as connection:` `  try:` `  connection.default_channel.queue_purge(_BIDS_QUEUE_NAME)` `  except amqp.exceptions.NotFound as error:` `  _logger.warning(str(error))` `  initialize_plugins(start_worker=True)`
Result/Recommendation	Instead of purging the bids queue at startup, consider implementing a mechanism to only purge queues based on specific conditions or flags. Alternatively, ensure that queue purging is optional and can be controlled through configuration settings. Proposed Code Changes: 1. Introduce a configuration flag to control queue purging: # In configuration file (e.g., service-node-config.yml) celery: purge_bids_queue_at_startup: false 2. Modify the code to check the flag before purging the queue: if is_main_module(): # pragma: no cover _logger.info('Initializing the Celery application...') initialize_application(False)

	if config['celery'].get('purge_bids_queue_at_startup', False): # purge the bids queue at startup with celery_app.connection_for_write() as connection: try: connection.default_channel.queue_purge(_BIDS_QUEUE_NAME) except amqp.exceptions.NotFound as error: _logger.warning(str(error)) initialize_plugins(start_worker=True)

LOW ISSUES

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

6.2.6 Broad Exception Handling in get_cross_blockchain_bids Method

Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: servicenode/business/bids.py
Update: We log the error in the layers above the function call and we do so by including the exact error which happened in the stack trace. This would not provide additional value.

Attack / Description	The get_cross_blockchain_bids method in the BidInteractor class uses a broad except Exception: block to capture and handle errors. This approach can mask specific exceptions that could provide more detailed information for debugging purposes. Catching more specific exceptions would enhance the code's structure and maintainability, leading to a more robust and clear error handling system.
Code	Line 79 - 104 (bids.py): try:

```
        _logger.info('Reading cross-blockchain bids from database')
        raw_bids = database_access.read_cross_blockchain_bids(
            source_blockchain_id, destination_blockchain_id)
        bids = []
        signer_config = get_signer_config()
        signer = get_signer(signer_config['pem'],
                            signer_config['pem_password'])
        for bid in raw_bids:
            bid_message = signer.build_message('', int(bid.fee),
                                               int(bid.valid_until),
                                               source_blockchain_id,
                                               destination_blockchain_id,
                                               int(bid.execution_time))
            signature = signer.sign_message(bid_message)
            bids.append({
                'fee': int(bid.fee),
                'execution_time': int(bid.execution_time),
                'valid_until': int(bid.valid_until),
                'signature': signature
            })
    except Exception:
        raise BidInteractorError(
            'unable to read cross-blockchain bids from '
            f'{Blockchain(source_blockchain_id)} to '
            f'{Blockchain(destination_blockchain_id)} from database')
```

	return bids
Result/Recommendation	Replace the broad except Exception: block with more specific exception handling. For example: try: _logger.info('Reading cross-blockchain bids from database') raw_bids = database_access.read_cross_blockchain_bids(source_blockchain_id, destination_blockchain_id) bids = [] signer_config = get_signer_config() signer = get_signer(signer_config['pem'], signer_config['pem_password']) for bid in raw_bids: bid_message = signer.build_message('', int(bid.fee), int(bid.valid_until), source_blockchain_id, destination_blockchain_id, int(bid.execution_time)) signature = signer.sign_message(bid_message) bids.append({ 'fee': int(bid.fee), 'execution_time': int(bid.execution_time), 'valid_until': int(bid.valid_until), 'signature': signature }) except database_access.DatabaseError as db_err: _logger.error(f"Database error occurred: {str(db_err)}") raise BidInteractorError('unable to read cross-blockchain bids due to database issues from ' f"{Blockchain(source_blockchain_id)} to " f"{Blockchain(destination_blockchain_id)}") except signer.SignerError as signer_err:

	_logger.error(f"Signer error occurred: {str(signer_err)}") raise BidInteractorError('unable to read cross-blockchain bids due to signer issues from ' f"{Blockchain(source_blockchain_id)} to " f"{Blockchain(destination_blockchain_id)}") except Exception as ex: _logger.error(f"An unexpected error occurred: {str(ex)}") raise BidInteractorError('unable to read cross-blockchain bids from ' f"{Blockchain(source_blockchain_id)} to " f"{Blockchain(destination_blockchain_id)} from database") Using specific exception handling will provide more useful error messages for debugging. It increases the robustness of the error handling system by dealing with different failure scenarios appropriately.

6.2.7 Insufficient Log Message Context

Severity: LOW
Status: FIXED
File(s) affected: servicenode/business/bids.py
Update: https://github.com/pantos-io/servicenode/releases/tag/1.8.2

Attack / Description	Current log messages, like _logger.info('Reading cross-blockchain bids from database'), are too generic. They do not provide enough information to pinpoint the exact operation being logged. Including specific details such as source and destination blockchain IDs would make the logs more informative and useful in diagnosing issues.
Code	Line 53 - 57 (bids.py):

	def get_cross_blockchain_bids(self, source_blockchain_id: int, destination_blockchain_id: int) \ -> typing.List[typing.Dict[str, typing.Any]]: """Get all cross-blockchain bids for the given source and destination blockchain ID.
Result/Recommendation	Enhance the log messages to include source and destination blockchain IDs: def get_cross_blockchain_bids(self, source_blockchain_id: int, destination_blockchain_id: int) -> typing.List[typing.Dict[str, typing.Any]]: """Get all cross-blockchain bids for the given source and destination blockchain ID. """ try: _logger.info(f'Reading cross-blockchain bids from database for source blockchain ID: {source_blockchain_id} and destination blockchain ID: {destination_blockchain_id}') raw_bids = database_access.read_cross_blockchain_bids(source_blockchain_id, destination_blockchain_id) ...

6.2.8 Broad Exception Handling in update_node_registrations

Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: servicenode/business/node.py
Update: We log the error in the layers above the function call and we do so by including the exact error which happened in the stack trace. This would not provide additional value.

Attack / Description	The method update_node_registrations is designed to update the service node registrations on all supported blockchains. Within this method, an except Exception block is used to catch and raise a custom NodeInteractorError. However, this approach is too broad and can catch exceptions that should be handled separately or can provide more detailed information. The broad exception handling: • Catches all exceptions, including those that might not need to be caught. • Provides a generic error message without specific details about what went wrong. • Hides bugs that may be better addressed individually.
Code	Line 25 - 75 (node.py): `class NodeInteractor(Interactor):` `"""Interactor for managing the service node itself.` `"""` `def update_node_registrations(self) -> None:` `"""Update the service node registrations on all supported` `blockchains.` `Raises` `------` `NodeInteractorError` `If a service node registration cannot be updated.` `"""` `for blockchain in Blockchain:` `_logger.info('updating the service node registration on '`

```
                '{}'.format(blockchain.name))
        try:
            blockchain_config = get_blockchain_config(blockchain)
            if not blockchain_config['active']:
                continue
            to_be_registered = blockchain_config['registered']
            blockchain_client = get_blockchain_client(blockchain)
            is_registered = blockchain_client.is_node_registered()
            if to_be_registered and is_registered:
                old_node_url = blockchain_client.read_node_url()
                new_node_url = config['application']['url']
                if old_node_url != new_node_url:
                    blockchain_client.update_node_url(new_node_url)
            elif to_be_registered:
                is_unbonding = blockchain_client.is_unbonding()
                if is_unbonding:
                    # Service node was unregistered but the stake
                    # has not been withdrawn yet
                    blockchain_client.cancel_unregistration()
                else:
                    # Not yet registered
                    unstaking_address = blockchain_config[
                        'unstaking_address']
                    node_url = config['application']['url']
                    node_stake = blockchain_config['stake']
```

```
                    blockchain_client.register_node(
                        node_url, node_stake, unstaking_address)
                elif is_registered:
                    # Not to be registered anymore
                    blockchain_client.unregister_node()
                # Do nothing if neither registered nor to be registered
            except Exception:
                raise NodeInteractorError(
                    'unable to update the service node registration on '
                    '{}'.format(blockchain.name))
```

Result/Recommendation

Catch specific exceptions where possible to provide more detailed error handling and improve the clarity of the error messages. Below is an improved version of the update_node_registrations method:

```
class NodeInteractor(Interactor):
    """Interactor for managing the service node itself."""

    def update_node_registrations(self) -> None:
        """Update the service node registrations on all supported blockchains.

        Raises
        ------
        NodeInteractorError
            If a service node registration cannot be updated.
        """
        for blockchain in Blockchain:
            _logger.info('updating the service node registration on '
```

```
                    '{}'.format(blockchain.name))
            try:
                blockchain_config = get_blockchain_config(blockchain)
                if not blockchain_config['active']:
                    continue
                to_be_registered = blockchain_config['registered']
                blockchain_client = get_blockchain_client(blockchain)
                is_registered = blockchain_client.is_node_registered()
                if to_be_registered and is_registered:
                    old_node_url = blockchain_client.read_node_url()
                    new_node_url = config['application']['url']
                    if old_node_url != new_node_url:
                        blockchain_client.update_node_url(new_node_url)
                elif to_be_registered:
                    is_unbonding = blockchain_client.is_unbonding()
                    if is_unbonding:
                        blockchain_client.cancel_unregistration()
                    else:
                        unstaking_address = blockchain_config[
                            'unstaking_address']
                        node_url = config['application']['url']
                        node_stake = blockchain_config['stake']
                        blockchain_client.register_node(
                            node_url, node_stake, unstaking_address)
                elif is_registered:
                    blockchain_client.unregister_node()
                # Do nothing if neither registered nor to be registered
            except (SpecificException1, SpecificException2) as e:
                _logger.error(f'Error updating the service node registration on {blockchain.name}:
{str(e)}')
                raise NodeInteractorError(f'unable to update the service node registration on
{blockchain.name}') from e
```

	except Exception as e: _logger.error(f'Unexpected error updating the service node registration on {blockchain.name}: {str(e)}') raise NodeInteractorError(f'unable to update the service node registration on {blockchain.name}') from e By catching specific exceptions, you can handle different error conditions appropriately and provide more informative error messages. The generic Exception catch block is only used as a fallback for unexpected errors, which are then logged and re-raised, preserving the exception context.

6.2.9 Error Handling during Table Initialization

Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: servicenode/database/init.py
Update: However, if you consider the function as a standalone unit, the comment is fair. We could implement the suggestion in order to avoid misusing it.

Attack / Description	During the initialization of tables in the database, the code does not handle potential exceptions that may occur when adding records. This can lead to unhandled exceptions and potential instability during the table initialization process.
Code	Line 93 - 127 (init.py): `def initialize_package(is_flask_app: bool = False) -> None:` `  # Before connecting, run alembic to ensure` `  # the database schema is up to date` `  if is_flask_app and config['database']['apply_migrations']:` `  run_migrations(config['database']['alembic_config'],`

```
                config['database']['url'])


global _sql_engine
_sql_engine = sqlalchemy.create_engine(
    config['database']['url'], pool_size=config['database']['pool_size'],
    max_overflow=config['database']['max_overflow'], pool_pre_ping=True,
    echo=config['database']['echo'])
global _session_maker
_session_maker = sqlalchemy.orm.sessionmaker(bind=_sql_engine)
# Initialize the tables
with _session_maker.begin() as session:
    assert isinstance(session, Session)  # type hint
    # Blockchain table
    statement = sqlalchemy.select(sqlalchemy.func.max(Blockchain_.id))
    max_blockchain_id = session.execute(statement).scalar_one_or_none()
    for blockchain in sorted(Blockchain):
        if (max_blockchain_id is None
                or max_blockchain_id < blockchain.value):
            session.add(
                Blockchain_(id=blockchain.value, name=blockchain.name))
    # Transfer status table
    statement = sqlalchemy.select(sqlalchemy.func.max(TransferStatus_.id))
    max_transfer_status_id = session.execute(
        statement).scalar_one_or_none()
    for transfer_status in sorted(TransferStatus):
```

	if (max_transfer_status_id is None or max_transfer_status_id < transfer_status.value): session.add(TransferStatus_(id=transfer_status.value, name=transfer_status.name))
Result/Recommendation	Wrap the table initialization code in a try-except block to handle potential SQLAlchemy errors. def initialize_package(is_flask_app: bool = False) -> None: # Before connecting, run alembic to ensure # the database schema is up to date if is_flask_app and config['database']['apply_migrations']: run_migrations(config['database']['alembic_config'], config['database']['url']) global _sql_engine _sql_engine = sqlalchemy.create_engine(config['database']['url'], pool_size=config['database']['pool_size'], max_overflow=config['database']['max_overflow'], pool_pre_ping=True, echo=config['database']['echo']) global _session_maker _session_maker = sqlalchemy.orm.sessionmaker(bind=_sql_engine) # Initialize the tables try: with _session_maker.begin() as session: assert isinstance(session, Session) # type hint # Blockchain table statement = sqlalchemy.select(sqlalchemy.func.max(Blockchain_.id)) max_blockchain_id = session.execute(statement).scalar_one_or_none() for blockchain in sorted(Blockchain): if (max_blockchain_id is None

	or max_blockchain_id < blockchain.value): session.add(Blockchain_(id=blockchain.value, name=blockchain.name)) # Transfer status table statement = sqlalchemy.select(sqlalchemy.func.max(TransferStatus_.id)) max_transfer_status_id = session.execute(statement).scalar_one_or_none() for transfer_status in sorted(TransferStatus): if (max_transfer_status_id is None or max_transfer_status_id < transfer_status.value): session.add(TransferStatus_(id=transfer_status.value, name=transfer_status.name)) except sqlalchemy.exc.SQLAlchemyError as e: _logger.error(f"Error initializing tables: {e}") raise DatabaseError("Failed to initialize tables") from e

6.2.10 is_main_module Function Might Not Reliably Detect All Celery Worker Processes

Severity: LOW
Status: FIXED
File(s) affected: validatornode/celery.py
Update:

Attack / Description	The is_main_module function in the Celery initialization module might not reliably detect all Celery worker processes, depending on how they're started. This could lead to inconsistencies, especially during the initialization phase of the Celery application.
Code	Line 26 - 27 (celery.py):

	def is_main_module() -> bool: return __name__ == '__main__' or any('celery' in arg for arg in sys.argv)
Result/Recommendation	Enhance the is_main_module detection logic to ensure it reliably detects all instances of Celery worker processes. Example: def is_main_module() -> bool: potential_celery_markers = ['celery', 'worker', 'beat', 'flower'] return __name__ == '__main__' or any(marker in sys.argv for marker in potential_celery_markers)

INFORMATIONAL ISSUES

During the audit, softstack's experts found **2 Informational issue** in the code of the smart contract.

6.2.11 Unsafe Assumption in read_transfer_nonce Function

Severity: INFORMATIONAL
Status: FIXED
File(s) affected: servicenode/database/access.py
Update: https://github.com/pantos-io/servicenode/releases/tag/1.8.2

Attack / Description	The read_transfer_nonce function in the module for creating, reading, updating, and deleting database records assumes that the query result will always contain at least one row. This assumption can lead to an IndexError if the query returns an empty result set. The function directly accesses result[0][0] without checking if result is empty. If no matching record is found for the given internal_transfer_id, result will be an empty list, and attempting to access the first element will raise

	an IndexError. Proof of Concept: If no record matches the given internal_transfer_id, an empty result set is returned, causing the following line to raise an IndexError: return result[0][0] # type: ignore This scenario can occur during normal operation if an invalid or non-existent internal_transfer_id is provided.
Code	Line 252 - 270 (access.py): def read_transfer_nonce(internal_transfer_id: int) -> int: """Read the nonce of a transfer database record. Parameters ---------- internal_transfer_id : int The unique internal ID of the transfer. Returns ------- int The nonce of the transfer. """ statement = sqlalchemy.select(Transfer.nonce).filter(Transfer.id == internal_transfer_id)

	with get_session() as session: result = session.execute(statement).fetchall() return result[0][0] # type: ignore
Result/Recommendation	Add a check to ensure that the result set is not empty before accessing the first element. If no record is found, raise an appropriate custom exception (e.g., DatabaseError) or handle the case as needed. def read_transfer_nonce(internal_transfer_id: int) -> int: """Read the nonce of a transfer database record. Parameters ---------- internal_transfer_id : int The unique internal ID of the transfer. Returns ------- int The nonce of the transfer. Raises ------ DatabaseError If no transfer record with the given ID is found. """ statement = sqlalchemy.select(

	Transfer.nonce).filter(Transfer.id == internal_transfer_id) with get_session() as session: result = session.execute(statement).fetchall() if not result: raise DatabaseError(f"No transfer record found with ID: {internal_transfer_id}") return result[0][0] # type: ignore

6.2.12 Replace Assertions with Runtime Type Checking

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: validatornode/entities.py

Attack / Description	In entities.py, the CrossChainTransfer class uses assertions for type checking within the from_dict method. Assertions might be disabled in production environments, which could lead to critical type errors going unnoticed.
Code	Line 129 – 186 (entities.py) @staticmethod def from_dict(dict_: CrossChainTransferDict) -> 'CrossChainTransfer': """Create a cross-chain transfer instance from its dictionary representation. Parameters ---------- dict_ : CrossChainTransferDict

```
        The dictionary representation of the cross-chain transfer.

    Returns
    -------
    CrossChainTransfer
        The created cross-chain transfer instance.

    """
    source_blockchain_id = dict_['source_blockchain_id']
    assert isinstance(source_blockchain_id, int)
    destination_blockchain_id = dict_['destination_blockchain_id']
    assert isinstance(destination_blockchain_id, int)
    source_hub_address = dict_['source_hub_address']
    assert isinstance(source_hub_address, str)
    source_transfer_id = dict_['source_transfer_id']
    assert isinstance(source_transfer_id, int)
    source_transaction_id = dict_['source_transaction_id']
    assert isinstance(source_transaction_id, str)
    source_block_number = dict_['source_block_number']
    assert isinstance(source_block_number, int)
    source_block_hash = dict_['source_block_hash']
    assert isinstance(source_block_hash, str)
    sender_address = dict_['sender_address']
    assert isinstance(sender_address, str)
    recipient_address = dict_['recipient_address']
```

```
    assert isinstance(recipient_address, str)
    source_token_address = dict_['source_token_address']
    assert isinstance(source_token_address, str)
    destination_token_address = dict_['destination_token_address']
    assert isinstance(destination_token_address, str)
    amount = dict_['amount']
    assert isinstance(amount, int)
    fee = dict_['fee']
    assert isinstance(fee, int)
    service_node_address = dict_['service_node_address']
    assert isinstance(service_node_address, str)
    is_reversal_transfer = dict_['is_reversal_transfer']
    assert isinstance(is_reversal_transfer, bool)
    return CrossChainTransfer(Blockchain(source_blockchain_id),
                              Blockchain(destination_blockchain_id),
                              BlockchainAddress(source_hub_address),
                              source_transfer_id, source_transaction_id,
                              source_block_number, source_block_hash,
                              BlockchainAddress(sender_address),
                              BlockchainAddress(recipient_address),
                              BlockchainAddress(source_token_address),
                              BlockchainAddress(destination_token_address),
                              amount, fee,
                              BlockchainAddress(service_node_address),
                              is_reversal_transfer)
```

Result/Recommendation	Replace assert statements with explicit type checks that raise appropriate exceptions if the types are incorrect. This ensures that type errors are caught regardless of the environment in which the code is running. @staticmethod def from_dict(dict_: CrossChainTransferDict) -> 'CrossChainTransfer': """Create a cross-chain transfer instance from its dictionary representation. Parameters ---------- dict_ : CrossChainTransferDict The dictionary representation of the cross-chain transfer. Returns ------- CrossChainTransfer The created cross-chain transfer instance. Raises ------ TypeError If any of the values in the dictionary do not match the expected types. """ if not isinstance(dict_['source_blockchain_id'], int): raise TypeError('source_blockchain_id must be an int') if not isinstance(dict_['destination_blockchain_id'], int): raise TypeError('destination_blockchain_id must be an int') if not isinstance(dict_['source_hub_address'], str): raise TypeError('source_hub_address must be a str') if not isinstance(dict_['source_transfer_id'], int): raise TypeError('source_transfer_id must be an int')

```
    if not isinstance(dict_['source_transaction_id'], str):
        raise TypeError('source_transaction_id must be a str')
    if not isinstance(dict_['source_block_number'], int):
        raise TypeError('source_block_number must be an int')
    if not isinstance(dict_['source_block_hash'], str):
        raise TypeError('source_block_hash must be a str')
    if not isinstance(dict_['sender_address'], str):
        raise TypeError('sender_address must be a str')
    if not isinstance(dict_['recipient_address'], str):
        raise TypeError('recipient_address must be a str')
    if not isinstance(dict_['source_token_address'], str):
        raise TypeError('source_token_address must be a str')
    if not isinstance(dict_['destination_token_address'], str):
        raise TypeError('destination_token_address must be a str')
    if not isinstance(dict_['amount'], int):
        raise TypeError('amount must be an int')
    if not isinstance(dict_['fee'], int):
        raise TypeError('fee must be an int')
    if not isinstance(dict_['service_node_address'], str):
        raise TypeError('service_node_address must be a str')
    if not isinstance(dict_['is_reversal_transfer'], bool):
        raise TypeError('is_reversal_transfer must be a bool')

    return CrossChainTransfer(
        Blockchain(dict_['source_blockchain_id']),
        Blockchain(dict_['destination_blockchain_id']),
        BlockchainAddress(dict_['source_hub_address']),
        dict_['source_transfer_id'],
        dict_['source_transaction_id'],
        dict_['source_block_number'],
        dict_['source_block_hash'],
        BlockchainAddress(dict_['sender_address']),
```

```
        BlockchainAddress(dict_['recipient_address']),
        BlockchainAddress(dict_['source_token_address']),
        BlockchainAddress(dict_['destination_token_address']),
        dict_['amount'],
        dict_['fee'],
        BlockchainAddress(dict_['service_node_address']),
        dict_['is_reversal_transfer']
    )
```

6.3 Verify Claims

6.3.1 Proper Authentication and Authorization Mechanisms: Ensure only authorized users can access APIs and initiate/validate cross-chain transfers.
Status: tested and verified ✅

6.3.2 Secure Handling of Private Keys: Verify that private keys are securely stored, encrypted, and access is strictly controlled and logged. Input Validation and Sanitization: Ensure all user inputs and data are validated and sanitized to protect against injections, XSS, and CSRF.
Status: tested and verified ✅

6.3.3 Comprehensive Logging and Monitoring: Verify that critical actions and events are logged, securely stored, monitored, and alerts for suspicious activities are in place.
Status: tested and verified ✅

6.3.4 Codebase Adheres to Best Practices and Checks for Common Vulnerabilities: Ensure the codebase adheres to best practices, is free of common vulnerabilities, and follows secure coding standards.
Status: tested and verified ✅

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the codebase and architecture provided by the Pantos team. The main objective of the audit was to verify the security and functionality claims of the codebase. 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.
- No high severity issues were found.
- 5 medium severity issues were found.
- 5 low severity issues were discovered
- 2 informational issues were identified

The audit report provides detailed descriptions of each identified issue, including severity levels, 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