BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

DATE: 8th AUG 2024

PREPARED FOR: OKP4 COSMWASM

About BlockApex

Founded in early 2021, is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

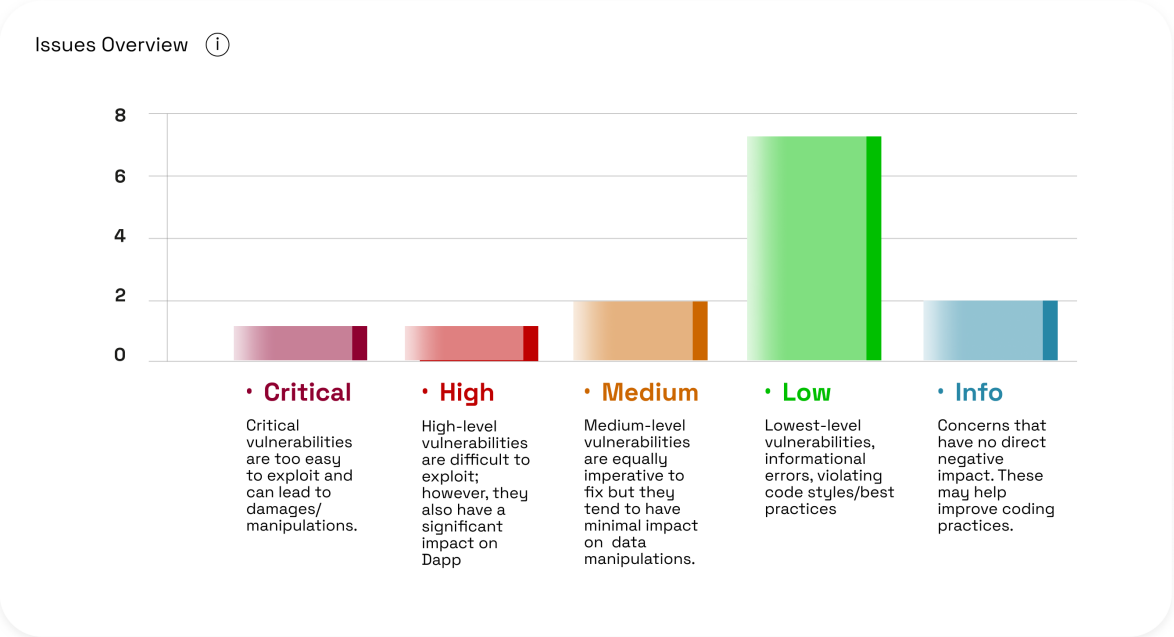
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

Contents

1 Executive Summary

Our team conducted a Filtered Audit, engaging Three auditors to independently examine the OKP4 CosmWasm Contracts. This rigorous approach included a meticulous line-by-line code review to identify potential vulnerabilities. Following the initial manual inspection, all flagged issues were thoroughly re-examined and re-tested to confirm their validity and ensure accuracy in our final assessment.



1.1 Scope

1.1.1 In Scope

Timeline:

- **3rd April, 2024:** Project kick off
- **8th April, 2024:** Project Status Meeting #1
- **15th April, 2024:** Project Status Meeting #2
- **22nd April, 2024:** Project Status Meeting #3
- **30th April, 2024:** Final Triage Meeting
- **3rd May, 2024:** Initial Report Delivery
- **8th Aug, 2024:** Final Report Delivery

Targets:

- **Contracts in Scope**: `Contracts/*`
- **Initial Commit Hash**: cde785fbd2dad71608d53f8524e0ef8c8f8178af
- **Type**: *Cosmwasm*
- **Platform**: *Cosmos*
- **Language**: *Rust*

1.1.2 Project Coverage:

The OKP4 ecosystem presents a sophisticated architecture designed to facilitate the decentralized orchestration of off-chain resources through its public blockchain. This audit encompasses a detailed examination of key smart contracts within this ecosystem, specifically focusing on their roles, functionalities, and integration within the broader network.

Objectarium Contract

- **Purpose and Functionality**: The Objectarium contract leverages the CosmWasm framework to offer robust data storage solutions on the Cosmos blockchain network. This contract is pivotal for managing arbitrary, unstructured data which can include anything from simple text files to complex structured data. The flexibility of the Objectarium contract makes it indispensable for a wide array of decentralized applications (dApps) that depend on diverse storage requirements.

Key Features:

- **Versatile Data Storage**: Supports various data types, ensuring broad application compatibility.
- **Data Immortality and Accessibility**: Ensures data is immutable and permanently accessible once stored, unless explicitly removed.
- **Dynamic Data Management**: Provides functionalities such as pinning/unpinning and forgetting objects, which help manage storage longevity and costs effectively.
- **Transparent and Controlled Data Handling**: The contract operations ensure transparency in data handling while giving users control over their data permanence through pinning mechanisms.

Technical Insights: The contract facilitates operations like object storage with optional pinning, and retrieval or deletion based on pin status. This is managed through a series of smart contract functions that handle the lifecycle of the data stored, adhering to predefined limits and configurations set during instantiation.

Law Stone Contract

- **Purpose and Functionality**: The Law Stone smart contract serves as a governance backbone within the OKP4 ecosystem, enabling Governance as a Service (GaaS). It is tailored to store, manage, and ensure the availability of governance rules which are crucial for the operation of various zones within the network.

Key Features:

- **Immutable Rule Storage**: Once deployed, the rules within cannot be altered, only queried or deactivated.
- **Rule Enforcement and Management**: Facilitates the questioning of rules and the deactivation (breaking the stone) of rules when they are no longer applicable.

Technical Insights: The Law Stone contract primarily interacts with the Objectarium to store and pin the governance rules. It includes mechanisms to unpin and remove these rules if they become obsolete, ensuring that the governance structure remains relevant and efficient.

Dataverse Contract

- **Purpose and Functionality**: The Dataverse contract orchestrates the management and interaction of a vast array of digital resources within the OKP4 ecosystem. This includes datasets, computational resources, and identity solutions, all governed by customizable policies enacted within various zones.

Key Features:

- **Resource Management**: Facilitates the registration, update, and deregistration of digital resources.
- **Decentralized Identity (DID)**: Implements DIDs to provide a robust framework for resource identification and interaction.
- **Claims and Credentials**: Manages verifiable credentials and claims about resources, enhancing the trust and integrity within the ecosystem.

Technical Insights: This contract plays a critical role in the administration of the dataverse's resources, leveraging both the Objectarium and Law Stone contracts to maintain a coherent and secure governance framework.

Cognitarium Contract

- **Purpose and Functionality**: The Cognitarium contract leverages the CosmWasm framework to facilitate the storage and querying of semantic data using RDF (Resource Description Framework) within the Cosmos blockchain network. This contract is designed as a semantic database, providing robust capabilities for handling complex data representations through triples, essential for supporting decentralized applications that require semantic data interoperability and complex data management.

Key Features:

- **Advanced Data Representation**: Supports RDF data formats including RDF/XML, Turtle, N-Triples, and N-Quads, catering to diverse data serialization needs.
- **Semantic Querying Capability**: Provides sophisticated querying options with SPARQL-like syntax, allowing for intricate data retrieval and manipulation.
- **Flexible Data Management**: Includes functionalities for inserting, deleting, and querying RDF triples, providing comprehensive management of semantic data. Scalable and Secure Data Handling: Ensures data integrity and security while allowing scalability through controlled triple insertion and removal based on defined policies.

Technical Insights: The Cognitarium contract is integral to the OKP4 ecosystem's ability to manage semantic data effectively. It interacts seamlessly with other components within the ecosystem to provide a structured and queryable layer of data that enhances the overall functionality and utility of the blockchain network. This contract not only supports the dynamic insertion and deletion of data but also enforces data integrity and access controls through its advanced query validation mechanisms, ensuring that the stored data remains consistent and secure across transactions.

1.1.3 Out of Scope

Features or functionalities not explicitly listed within the “In Scope” section, such as backend operations unrelated to the direct functioning of the smart contracts or external system integrations, are considered outside the scope of this audit.

1.2 Methodology

The codebase was audited using a filtered audit technique by a team of Three(3) auditors over a span of 3.5 weeks. The process began with a reconnaissance phase where the auditors developed a foundational understanding of the codebase, alongside relevant documentation and whitepapers. This initial phase helped form presumptions for the developed codebase. As the audit progressed into the manual code review phase, auditors made the Proofs of Concept (POCs) to verify their findings. This phase was designed to identify logical flaws, complemented by code optimizations, software and security design patterns, code styles, and best practices.

1.3 Project Goals

The engagement was scoped to provide a security assessment of the Okp4 Cosmwasm smart contract. Specifically, we sought to answer the following non-exhaustive list of questions:

1. Does the smart contract properly handle different data types ?
2. Are the access control mechanisms effectively implemented ?
3. Could there be DOS in contract’s data storage or retrieval functionalities?
4. Is there any potential risk that could compromise the integrity or availability of the law stone’s rules and facts?
5. How does the contract safeguard against unauthorized breaking of the law stone by the contract admin ?
6. How are resources, including services and digital assets within the Dataverse, securely registered, managed, and accessed particularly regarding their association with unique URIs and Registrars?

1.4 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.5 Summary of Findings Identified

S.No	Severity	Findings	Status
#1	CRITICAL	Denial of Service (DOS) via Front-Running Leads to Law Stone Initialization Failure	FIXED
#2	HIGH	Unauthorized Object Deletion Triggers Denial of Service	ACKNOWLEDGED
#3	MEDIUM	Unrestricted Access to Break_Stone if Deployed without admin	FIXED
#4	MEDIUM	Arithmetic Overflow in CosmWasm-Std Affects OKP4 Contracts	FIXED
#5	LOW	Format Restriction in Data Submission Leads to Underutilization of RDF Options	ACKNOWLEDGED
#6	LOW	Not All Storage Elements Are Exposed Through Queries.	FIXED
#7	LOW	Insufficient information in Events	FIXED
#8	LOW	Lack of Storage Address Validation in Law-Stone Instantiation	ACKNOWLEDGED
#9	LOW	Insufficient Input Validation in Objectarium Bucket Instantiation	ACKNOWLEDGED
#10	LOW	Inaccurate Compressed Size Tracking on Object Deletion in Objectarium	FIXED
#11	LOW	Inadequate Input Validation in Objectarium Contract Instantiation	FIXED
#12	INFO	Insufficient Error Handling in Object Access Within Objectarium	ACKNOWLEDGED
#13	INFO	Arbitrary Fund Acceptance in OKP4 Contracts	FIXED

2 Findings and Risk Analysis

2.1 Denial of Service (DOS) via Front-Running Leads to Law Stone Initialization Failure

Severity: Critical

Status: Fixed

Location :

1. `contracts/okp4-law-stone/src/contract.rs`
2. `contracts/okp4-objectarium/src/contract.rs`

Description The instantiation process of the Law Stone contract is susceptible to a front-running vulnerability when interacting with the Objectarium contract for storing .pl files. This vulnerability stems from the public visibility of transaction data in the mempool, which allows attackers to intercept and replicate the initialization parameters. The core issue arises during the `instantiate` function's call to `store_object` in the Objectarium. If an attacker captures and submits the same data/program to the Objectarium ahead of the legitimate transaction, the Law Stone's initialization will fail, leading to repeated Denial of Service (DoS)

Impact This vulnerability exposes the Law Stone contract to a persistent threat of initialization failure, which can be systematically exploited to prevent its deployment.

Proof of Concept :

1. **Monitoring the Mempool**: An attacker monitors the mempool for upcoming Law Stone instantiation transactions, extracting the program data to be stored.
2. **Executing a Front-Running Attac**k: Using the extracted data, the attacker quickly constructs and broadcasts a transaction to the Objectarium to store and pin this data.
3. **Blocking Law Stone Initialization**: When the legitimate Law Stone transaction executes, it attempts to store the already saved program, resulting in an "*Object Already Stored*" error and thus failing the initialization.

Recommendation Ensure that the Objectarium is aware of the Law Stone's dependencies and enforces checks that the calling contract matches expected parameters.

The issue has been fixed in this PR#550

2.2 Unauthorized Object Deletion Triggers Denial of Service

Severity: High

Status: Acknowledged

Location :

1. `contracts/okp4-objectarium/src/contract.rs`

Description In the Objectarium contract, the `store_object` function allows users to store data objects with an option to pin them. Pinning an object prevents its deletion unless explicitly unpinned by the owner. However, a issue arrises when objects are stored without being pinned (`pin`: **`false`**). In such cases, any user, regardless of ownership, can invoke the `forget_object` function to delete these objects. This flaw arises from the lack of proper ownership verification and permission checks before object deletion. Consequently, this issue allows unauthorized users to delete objects they do not own, leading to potential Denial of Service (DoS) attacks.If a user consistently stores objects with pin set to false, an attacker can monitor these actions and systematically delete newly stored objects by repeatedly calling the `forget_object` function. This behavior can repeatedly disrupt legitimate users' attempts to store data, effectively denying them the service of the contract.

Impact The primary impact of this vulnerability is unauthorized data manipulation leading to Denial of Service. By allowing any user to delete objects they do not own, the system's integrity and reliability are compromised. This not only violates access controls but also exposes the system to targeted attacks where critical data can be maliciously wiped out

Proof of Concept The following sequence outlines the vulnerability:

1. A user stores an object without setting it as pinned, effectively leaving it vulnerable to deletion.
2. Another user, who is not the owner, can then call the forget_object function, specifying the ID of the unpinned object.
3. Due to insufficient permission checks in the forget_object function, the object is deleted without verifying if the caller is the rightful owner.

This process is demonstrated in the provided test case:

```
#[test]
    fn forget_object_same_user() {
        let mut deps = mock_dependencies();
        let info = mock_info(&quot;creator&quot;, &amp;[]);

        // Setup: Instantiate the contract
        let instantiate_msg = InstantiateMsg {
            bucket: &quot;test&quot;.to_string(),
            config: Default::default(),
            limits: Default::default(),
            pagination: Default::default(),
        };
```

```
13          instantiate(deps.as_mut(), mock_env(), info.clone(), instantiate_msg)
14              .expect(&quot;instantiation should work&quot;);
15
16          // Store an object and pinning false
17          let data = general_purpose::STANDARD.encode(&quot;okp4&quot;);
18          let store_msg = ExecuteMsg::StoreObject {
19              data: Binary::from_base64(data.as_str()).unwrap(),
20              pin: false,
21              compression_algorithm: Some(CompressionAlgorithm::Passthrough),
22          };
23          execute(deps.as_mut(), mock_env(), info.clone(), store_msg)
24              .expect(&quot;storing object should work&quot;);
25
26
27         // Attempt to pin the object from 2 diff users
28          let pin_msg = ExecuteMsg::PinObject {
29              id: &quot;315d0d9ab12c5f8884100055f79de50b72db4bd2c9bfd3df049d89640fed1fa6&
                    quot;.to_string(),
30          };
31          execute(deps.as_mut(), mock_env(), mock_info(&quot;bob&quot;, &amp;[]), pin_msg.
                clone())
32              .expect(&quot;pinning should work&quot;);
33          execute(deps.as_mut(), mock_env(), mock_info(&quot;alice&quot;, &amp;[]), pin_msg)
34              .expect(&quot;pinning should not work&quot;);
35
36
37          // Unpins the object from 2 diff users
38          let unpin_msg = ExecuteMsg::UnpinObject {
39              id: &quot;315d0d9ab12c5f8884100055f79de50b72db4bd2c9bfd3df049d89640fed1fa6&
                    quot;.to_string(),
40          };
41          execute(deps.as_mut(), mock_env(), mock_info(&quot;bob&quot;, &amp;[]), unpin_msg.
                clone())
42              .expect(&quot;unpinning should not be working&quot;);
43          execute(deps.as_mut(), mock_env(), mock_info(&quot;alice&quot;, &amp;[]),
                unpin_msg.clone())
44              .expect(&quot;unpinning should not be working&quot;);
45
46
47         // Forgets the object not by the owner
48          let forget_msg = ExecuteMsg::ForgetObject {
49              id: &quot;315d0d9ab12c5f8884100055f79de50b72db4bd2c9bfd3df049d89640fed1fa6&
                    quot;.to_string(),
50          };
51          let res = execute(deps.as_mut(), mock_env(), mock_info(&quot;bob&quot;, &amp;[]),
                forget_msg).unwrap();
52          print!(&quot;{:?}&quot;, res);
53
54      }
```

Recommendation To mitigate this vulnerability, implement stringent ownership checks within the `forget_object` function to ensure that only the object's owner can delete it. This could involve Verifying that the caller of `forget_object` matches the stored owner's information before allowing the object's deletion

The issue has been discussed in this PR#551

2.3 Unrestricted Access to Break_Stone if Deployed without admin

Severity: Medium

Status: Fixed

Location :

1. `contracts/okp4-law-stone/src/contract.rs`

Description The Law Stone contract utilizes the `store_object` function of the Objectarium to store .pl files containing rule sets. A key function within this contract is `break_stone`, designed to modify or remove these rules by unpinning or forgetting them. The function implements an admin check to restrict access: if an admin address is configured, it ensures that only the admin can execute the function by comparing the info.sender with the admin address. Specifically, the function logic:

```
{
    Some(admin_addr) if admin_addr != info.sender =&gt; Err(ContractError::Unauthorized),
    _ =&gt; Ok(()),
};
```

indicates that if an admin is set and the caller is not the admin, the function will deny access. However, if no admin is defined, particularly in deployments initiated with the `--no-admin` flag in CosmWasm CLI, the function proceeds without any access restrictions. This lack of checks in the absence of an admin means that anyone can invoke `break_stone` and potentially disrupt the rule enforcement by the Law Stone, impacting the governance or operational constraints enforced by these rules.

Recommendation :

Enforce Admin Configuration: Modify the contract deployment process to require an admin address explicitly. This change would prevent the contract from being deployed without admin oversight.

Default Admin Fallback: Implement a default admin setting that can be used if no specific admin is provided during deployment, ensuring there's always some level of controlled access.

The issue has been fixed in this PR#552

2.4 Arithmetic Overflow in CosmWasm-Std Affects OKP4 Contracts

Severity: Medium

Status: Fixed

Location :

1. `contracts/okp4-*/contract.rs`

Library: `cosmwasm-std version 1.5.3`

Description The OKP4 ecosystem is currently using version 1.5.3 of the cosmwasm-std library, which has been found to contain arithmetic overflow issues as detailed in advisory CWA-2024-002 . This vulnerability affects all contracts that perform arithmetic operations, including Objectarium, Cognitarium, Dataverse, and Law Stone. Arithmetic overflows can alter the expected behavior of smart contracts by causing computations to wrap incorrectly.

Impact This overflow can lead to incorrect data processing, resulting in potential state corruption or mismanagement of contract logic. It directly threatens the reliability and effectiveness of the contract's intended functionalities.

Proof of Concept

```
/ save bucket stats
        BUCKET.update(deps.storage, |mut bucket| -&gt; Result&lt;_, ContractError&gt; {
            let stat = &amp;mut bucket.stat;
            stat.size += size;
            stat.object_count += Uint128::one();
            stat.compressed_size += compressed_size;
            Ok(bucket)
        })?;
```

Recommendation Upgrade the cosmwasm-std library to the latest patched version as recommended in the advisory

The issue has been fixed in this PR#553

2.5 Format Restriction in Data Submission Leads to Underutilization of RDF Options

Severity: Low

Status: Acknowledged

Location :

1. `contracts/okp4-cognitarium/src/contract.rs`
2. `contracts/okp4-dataverse/src/contract.rs`

Description The Cognitarium smart contract is designed to support multiple RDF formats for data submission, including *NQuads*, *RDF.XML*, *Turtle*, and *NTriples*. This versatility is intended to enhance the contract's adaptability and user experience by allowing for flexible data representation. However, the Dataverse contract, which controls the data input to the Cognitarium, is currently hardcoded to use only the *NQuads* format. This restriction arises from a limitation within the `submit_claims` function, which lacks the capability to accept a format parameter, contrary to what is documented . As a result, despite the Cognitarium's ability to handle various formats, this functionality is underutilized due to the Dataverse's fixed format implementation.

This issue not only limits the flexibility of data input but also leads to a discrepancy between the system's documented capabilities and its actual functionality. The mismatch can cause confusion among users and developers, who may expect broader format support based on the official contract documentation.

Recommendation :

1. **Dynamic Format Handling**: Revise the `submit_claims` function in the Dataverse contract to include a *format* parameter, allowing users to specify the desired RDF format for their data submissions. This adjustment will enable the contract to dynamically select the appropriate data handling method based on user input.
2. **Documentation Alignment**: Update the contract documentation to accurately reflect the operational capabilities and limitations. If the implementation of dynamic format selection is deferred, the documentation should clearly state that currently, only the *NQuads* format is supported.

The issue has been Discussed in this PR#554

2.6 Not All Storage Elements Are Exposed Through Queries.

Severity: Low

Status: Fixed

Location :

1. `contracts/okp4-objectarium/src/contract.rs`
2. `contracts/okp4-cognitarium/src/contract.rs`

Description The Cognitarium and Objectarium contracts exhibit limitations in their query functionalities, This forces users and other contracts to perform a raw query to read the stored value, tying their code to the current implementation of the cognitarium contract, which is error-prone.

1. **Cognitarium Contract Limitations**: This contract lacks query functions for critical state variables such as *NAMESPACE_KEY_INCREMENT* and *BLANK_NODE_IDENTIFIER_COUNTER*. These variables are fundamental for tracking the increments of namespace keys and the identifiers for blank nodes, pivotal for organizing and retrieving semantic data. The absence of query functions for these variables restricts the ability to monitor and manage internal state changes effectively.
2. **Objectarium Contract Limitations**: Similarly, the Objectarium contract does not provide adequate query functions for accessing important metadata about buckets, such as the owner details, statistics (*size*, *compressed_size*, *object_count*), and other configuration parameters. This limitation hinders users or applications from retrieving essential information that could assist in assessing the usage, management, and ownership of buckets.

Recommendation We recommend exposing a smart query that returns the above-mentioned elements

The issue has been fixed in this PR#555

2.7 Insufficient information in Events

Severity: Low

Status: Fixed

Location :

1. `contracts/okp4-law-stone/src/contract.rs`
2. `contracts/okp4-objectarium/src/contract.rs`

Description :

- **Law Stone Contract:** The `break_stone` function in the Law Stone contract fails to emit detailed events for actions undertaken within the function. Notably, actions such as *unpinning* or *forgetting* objects lack corresponding information in event emissions, which are crucial for auditing and tracing the state changes within the contract.
- **Objectarium Contract:** The functions within the Objectarium contract consistently emit events that only include the "*action*" and "*id*" parameters. This results in the omission of critical information such as *pin counts*, the *pin status* of objects, and the *remaining pins* after an object is forgotten. This lack of detail in event logs hampers effective monitoring and debugging of the system.

Recommendation :

1. **For the Law Stone Contract:** Enhance the `break_stone` function to include detailed event emissions for all significant actions, particularly for *unpinning* and *forgetting* operations. Each event should detail the affected objects and the resultant state changes.
2. **For the Objectarium Contract:** Modify event emissions to include additional data such as pin counts, the current pin status of each object, and updates following any "*forget*" operations. This improvement will enable better insight into the contract's operation and facilitate data tracking and integrity checks.

The issue has been fixed in this PR#556

2.8 Lack of Storage Address Validation in Law-Stone Instantiation

Severity: Low

Status: Acknowledged

Location :

1. `contracts/okp4-law-stone/src/contract.rs`

Description The instantiation process of the law-Stone does not include validation for the `storage_address` provided in the `InstantiateMsg`.This will result in a transaction failure

Recommendation Implement proper validation checks in the instantiate function to ensure the `storage_address` is well-formed and authorized for the intended operational context.

The issue has been discussed in this PR#557

2.9 Insufficient Input Validation in Objectarium Bucket Instantiation

Severity: Low

Status: Acknowledged

Location :

1. `contracts/okp4-objectarium/src/contract.rs`

Description The Objectarium contract's instantiate function lacks necessary validations on the bucket name parameter during the bucket creation process. This oversight allows for the creation of buckets with arbitrary lengths and potentially malicious content in their names. The absence of strict checks and sanitization on the bucket names could facilitate phishing attacks or cross-site scripting (XSS) vulnerabilities when these names are displayed in a frontend application.

Impact If exploited, this vulnerability could lead to phishing attacks and execution of unauthorized scripts in the context of a user's session (XSS), compromising the security of the frontend application and the integrity of user interactions.

Proof of Concept Currently, there are no restrictions on the characters or length of the bucket name input. An attacker can create a bucket with a name that includes special characters or scripts that are executable when rendered in web environments. For example:

1. Creating a bucket with the name containing HTML or JavaScript code snippets might lead to XSS or phishing risks if the bucket name is rendered directly in the web UI without proper sanitization.

```
pub struct InstantiateMsg {
    /// The name of the bucket.
    /// The name could not be empty or contains whitespaces.
    /// If name contains whitespace, they will be removed.
    pub bucket: String,
```

Recommendation Implement a validation check to ensure all input conforms to the URL-safe alphabet as specified in RFC 4648, using the defined character set.

The issue has been discussed in this PR#558

2.10 Inaccurate Compressed Size Tracking on Object Deletion in Objectarium

Severity: Low

Status: Fixed

Location :

1. `contracts/okp4-objectarium/src/contract.rs`

Description In the Objectarium contract, there is a discrepancy in how compressed sizes are handled during the lifecycle of an object. While the `store_object` function correctly increments the `compressed_size` statistic upon storing an object, the corresponding decrement operation is missing in the `forget_object` function when an object is removed. This oversight leads to inaccurate tracking of the compressed data size within the system,

Proof of Concept :

Compression Size Increment: In the `store_object` function, the compressed size of data is added to the bucket statistics upon successful storage:

```
// Save compressed object data and update stats
let compressed_size = (compressed_data.len() as u128).into();
BUCKET.update(deps.storage, |mut bucket| -&gt; Result&lt;_, ContractError&gt; {
    let stat = &amp;mut bucket.stat;
    stat.compressed_size += compressed_size;
    Ok(bucket)
})?;
```

Missing Decrement on Deletion: In the `forget_object` function, while the total size and object count are decremented, the compressed size adjustment is notably absent:

```
// Object removal without updating compressed size
BUCKET.update(deps.storage, |mut b| -&gt; Result&lt;_, ContractError&gt; {
    b.stat.object_count -= Uint128::one();
    b.stat.size -= object.size;
    Ok(b)
});
objects().remove(deps.storage, id.clone());
```

Recommendation To resolve this issue, update the `forget_object` function to include a decrement operation for the `compressed_size` stat similar to how it handles other metrics.

The issue has been fixed in this PR#559

2.11 Inadequate Input Validation in Objectarium Contract Instantiation

Severity: Low

Status: Fixed

Location :

1. `contracts/okp4-objectarium/src/contract.rs`

Description The instantiation process in the Objectarium contract lacks comprehensive input validation, specifically for the parameters associated with bucket configuration and limits. This deficiency may lead to configurations that render the contract functionally ineffective or vulnerable to misuse. The `try_new` method in the Bucket class currently only checks for an empty bucket name, overlooking critical validations on the numerical limits set for the bucket.

Impact Failing to validate input parameters adequately may result in the creation of buckets that are either too restrictive or too loosely defined, leading to operational inefficiencies or vulnerabilities

Proof of Concept

```
pub fn instantiate(
    deps: DepsMut&lt;&#x27;_&gt;,
    _env: Env,
    info: MessageInfo,
    msg: InstantiateMsg,
) -&gt; Result&lt;Response, ContractError&gt; {
    let bucket = Bucket::try_new(
        info.sender,
        msg.bucket,
        msg.config.into(),
        msg.limits.into(),
        msg.pagination.try_into()?,
    )?;

    set_contract_version(deps.storage, CONTRACT_NAME, CONTRACT_VERSION)?;
    BUCKET.save(deps.storage, &amp;bucket)?;

    Ok(Response::default())
}
```

```
impl Bucket {
    pub fn try_new(
        owner: Addr,
        name: String,
        config: BucketConfig,
        limits: BucketLimits,
        pagination: Pagination,
    ) -&gt; Result&lt;Self, BucketError&gt; {
        let n: String = name.split_whitespace().collect();
        if n.is_empty() {
            return Err(EmptyName);
        }

        Ok(Self {
            owner,
            name: n,
```

```
17              config,
18              limits,
19              pagination,
20              stat: BucketStat {
21                  size: Uint128::zero(),
22                  compressed_size: Uint128::zero(),
23                  object_count: Uint128::zero(),
24              },
25          })
26      }
27  }
```

Recommendation Enhance the validation logic within the `Bucket::try_new` method to include checks on all parameters.

The issue has been fixed in this PR#560

2.12 Insufficient Error Handling in Object Access Within Objectarium

Severity: Info

Status: Acknowledged

Location :

1. `contracts/okp4-objectarium/src/contract.rs`

Description The `pin_object` function in the Objectarium contract lacks preemptive checks to confirm the existence of an object before attempting operations on it. This oversight leads to attempts to manipulate non-existent objects, resulting in generic errors which are not descriptive of the actual issue. This issue is also evident in similar operations, such as `forget_object`, where object existence is assumed rather than verified before proceeding with further logic.

Proof of Concept The `pin_object` function attempts to load an object directly based on the provided ID, without verifying if the object actually exists in storage. If the object does not exist, the operation fails, and a non-specific error is returned:

```
pub fn pin_object(
    deps: DepsMut&lt;&#x27;_&gt;,
    info: MessageInfo,
    object_id: ObjectId,
) -&gt; Result&lt;Response, ContractError&gt; {

    // ....
    let object = objects().load(deps.storage, id.clone())?; // Potential fail point if
        object does not exist

    // Further operations...
}
```

Recommendation Implement and enforce existence checks before performing any operations on objects within the contract.

The issue has been discussed in this PR#561

2.13 Arbitrary Fund Acceptance in OKP4 Contracts

Severity: Info

Status: Fixed

Location :

1. `contract/okp4-*`

Description The OKP4 contracts currently do not have mechanisms to handle or refund arbitrary funds sent to them. This absence can result in the loss of funds mistakenly sent to these contracts, as there is no method implemented for withdrawing such funds.

Impact Funds sent in error to OKP4 contracts are irretrievable.

Recommendation Implement checks to reject all funds sent to the contract, preventing unauthorized or accidental transfers.

The issue has been fixed in this PR#562

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts