SECURITY AUDIT OF

XPLA WEARABLE COLLECTION

Public Report

Jun 12, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
NFT	A non-fungible token (NFT) is a unique digital identifier that cannot be copied, substituted, or subdivided, that is recorded in a blockchain, and that is used to certify authenticity and ownership.
XPLA	XPLA is an open-source blockchain leveraging Tendermint and the Cosmos SDK. It offers extensive experiences in De-Fi and P2O gaming, aiming to transition Web2 users into the Web3 space with EVM compatibility and developer-friendly SDKs.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jun 12, 2024. We would like to thank the XPLA for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the XPLA Wearable Collection. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team found some minor issues along with some recommendations in the application. XPLA team has resolved and updated these issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About XPLA

XPLA is a Tendermint-based Layer 1 blockchain that serves as a hub for digital media content. Based on the idea of 'Explore and Play', XPLA encompasses a wide range of digital content with a leading blockchain gaming infrastructure empowered by a sustainable ecosystem. As a universal content powerhouse, XPLA provides a sublime creative experience for all with its significant size game infrastructure.

With 'Play to Own' values at its core, XPLA fosters an environment where gamers' ownership and efforts are respected for a sustainable blockchain gaming ecosystem. Having recently onboarded top-notch IPs such as The Walking Dead: All-Stars, Summoners War: Chronicles, MiniGame Party, and Ace Fishing: Crew, including the first cross-chain game, Idle Ninja Online, XPLA is not merely progressing, but propelling ahead as a pivotal, content-driven blockchain, exhibiting an impressive spectrum of services.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of XPLA Wearable Collection.

It was conducted on the following git repository *https://github.com/xpladev/wearable-collection/* on commit `378028c10ac1c36065694a38305401ac61df7d1f`.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`166e746f27ed7b1a16fd4ca8596ae0fd22a284a87c2fe44e179cde54629e0e86`	`./wearable-collection/examples/schema.rs`
`62ef31b3c12eda62ef79073285224b2ee9f822c2526861b36510bb387db0f662`	`./wearable-collection/src/lib.rs`
`e1585726297db1c09ce68dfc61daaac3b95d5a4e6fa561041bd7d74be4ff4852`	`./cw721-base/examples/schema.rs`
`3771cd062b2ba81e97f25719edbcf4f4e20e74d09b94597e6ac004d29a45e70a`	`./cw721-base/src/execute.rs`
`8b668beafb9c49df059e9486a973b37227ef57c89b2cc1e8eb37e1ca0d046149`	`./cw721-base/src/multi_tests.rs`
`208759b034e87a39f347300c066279c8416e7e450d8b26db3732d0a6d2f06765`	`./cw721-base/src/error.rs`
`69dfe1fb6ccb034552f842a9d8182169a9b48bce47f7a2d9f1faa6be4607db57`	`./cw721-base/src/lib.rs`
`0a49286d130530cd21c195eaad2bcbdd90abe9c6cf07ddae7f6ed8ee376f2bda`	`./cw721-base/src/query.rs`
`9ab6430c06eaba32421fbe8c1c215f07b6386ed2992c90d8ddb9d79675d7c822`	`./cw721-base/src/helpers.rs`

3c4b7b2aa61ce85bc4439be06768ab7153e4fa551298a738ad4344543092d53a	./cw721-base/src/state.rs
0974a0c2c2c3cf0905a2a8b2a7f938473dcc933a5415ff9749d3cb0211880676	./cw721-base/src/msg.rs
972215902a5fcb251936d3a0facb545d193ad3bc3ffa33949df042a2ef429fc1	./cw721-base/src/upgrades/mod.rs
b50476e75eda437cc668fb3b47799ed628675202431a84f9e308c3f33cd3d62b	./cw721-base/src/upgrades/v0_17.rs
85b10b5705c86b2a886b388a929c6c77406dbd57051134a525dcb596a13d76af	./cw721-base/src/contract_tests.rs

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the CosmWasm smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Gas Usage, Gas Limit and Loops
- Reentrancy
- Access Control
- Logic Flaws
- Use Of Unsafe Libraries
- Arbitrary Message Execution
- Bech32 Address Validation and Normalization

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

XPLA acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. XPLA understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, XPLA agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the XPLA will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the XPLA, the final report will be considered fully accepted by the XPLA without the signature.

2. AUDIT RESULT

2.1. Overview

This is an XPLA blockchain-based smart contract designed to create a wearable collection of NFTs. Each NFT represents a unique wearable item, and multiple NFTs can be combined into a single collection, multi-layer nested collections are also supported. However, only collections with a depth of 1 can be transferred or sent to another account. The smart contract is written in Rust and uses the CosmWasm library.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of XPLA Wearable Collection.

XPLA fixed the code, according to Verichains's draft report.

#	Issue	Severity	Status
1	Multi-level composed NFT cannot be transferred	INFORMATIVE	Acknowledged
2	Token ID contains delimiter character should not be allowed	INFORMATIVE	Fixed
3	Vulnerable dependencies `cosmwasm-std`	INFORMATIVE	Fixed

2.2.1. Multi-level composed NFT cannot be transferred INFORMATIVE

Affected files:

- contracts/cw721-base/src/execute.rs

Assume that we have the following chain of composed NFTs (`A`, `B`, `C`):

`A (owner) -> B (owner@A) -> C (owner@B)`

If we want to transfer token `A`, the transfer will fail due to the dependent depth check. Furthermore, if we want to transfer token `B`, the transfer will also fail because of the token ownership check `token.owner != info.sender`. Therefore, it is impossible to transfer multi-level composed NFTs without decomposing them first. If this is the intended behavior, it should be documented in the contract's specification.

```
pub fn _transfer_composed_nft(
    &self,
    deps: DepsMut,
    info: &MessageInfo,
    recipient: &str,
    token_id: &str,
) -> Result<TokenInfo<T>, ContractError> {
    let mut token = self.tokens.load(deps.storage, token_id)?;

    // only can transfer composed token
    let is_dependent = self.is_dependency_token(&token)?;
    if is_dependent == false {
        return Err(ContractError::CannotTransfer{});
    }

    // only owner can transfer
    if token.owner != info.sender {
        return Err(ContractError::Ownership(OwnershipError::NotOwner));
    }

    //check dependednt deps (allow trasfer 1 deps)
    if let Some(d_tokens) = token.dependent_tokens.clone() {
        if d_tokens.len() > 0 {
            for d_id in d_tokens.iter() {
                let mut dependent_token = self.tokens.load(deps.storage, d_id)?;
                if let Some(d_t) = dependent_token.dependent_tokens.clone() {
                    if d_t.len() > 0 {
                        return Err(ContractError::TooMuchDeps{});
                    }
                }
                let new_owner = deps.api.addr_validate(recipient)?;
                let new_compose_owner = self.gen_dependent_token_owner(new_owner.as_str(),
token_id)?;
                dependent_token.owner = Addr::unchecked(new_compose_owner);
                dependent_token.approvals = vec![];
                self.tokens.save(deps.storage, d_id, &dependent_token)?;
            }
        }
    }

    // set owner and remove existing approvals
    token.owner = deps.api.addr_validate(recipient)?;
    token.approvals = vec![];
    self.tokens.save(deps.storage, token_id, &token)?;
    Ok(token)
}
```

UPDATESd

- **Jun 12, 2024**: This issue has been acknowledged and confirmed as intended behavior. Therefore, we have changed the severity from *MEDIUM* to *INFO*.

2.2.2. Token ID contains delimiter character `@` should not be allowed INFORMATIVE

Affected files:

- contracts/cw721-base/src/execute.rs

The current `Cw721Contract::mint` function does not check whether the newly minted token ID contains the delimiter character. This special character is used to generate the dependent tokens' owners. Allowing this character in the token ID may lead to confusing ownership information in the dependent tokens. For example, if we have the following chain of tokens:

`A (owner) -> B@X (owner@A) -> C (owner@B@X)`

In this case, the owner of the C token would be `owner@B@X`.

RECOMMENDATION

To prevent such confusion, the contract should not allow the character in the token ID.

UPDATESd

- **Jun 12, 2024**: This issue has been acknowledged and fixed by XPLA team.

2.2.3. Vulnerable dependencies `cosmwasm-std` INFORMATIVE

Affected files:

- Cargo.lock

```
[[package]]
name = "cosmwasm-std"
version = "1.5.3"
source = "registry+https://github.com/rust-lang/crates.io-index"
checksum = "ef8666e572a3a2519010dde88c04d16e9339ae751b56b2bb35081fe3f7d6be74"
```

The current version of `cosmwasm-std` is `1.5.3`, which contains some arithmetic overflow vulnerabilities in the following functions: `Uint{256,512}::pow`, `Int{256,512}::pow`, `Int{256,512}::neg`. The detail of the vulnerabilities can be found in the following links:

https://github.com/CosmWasm/advisories/blob/main/CWAs/CWA-2024-002.md

However, the affected functions are not used in the contract, and `overflow-checks = true` is also set, so the vulnerabilities are not exploitable in this case.

RECOMMENDATION

It is recommended to update the `cosmwasm-std` to the latest version to avoid potential vulnerabilities.

UPDATESd

- **Jun 12, 2024**: This issue has been acknowledged and fixed by XPLA team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 12, 2024*	Public Report	Verichains Lab

Table 2. Report versions history