OBJECT COMPUTING

Object Computing, Inc.
12140 Woodcrest Executive Dr.
Suite 300
St. Louis, MO 63141
+1 314.579.0066

Alien Worlds Token Contract Audit (EOS)

Source for this analysis is github: Alien-Worlds/alienworlds-contracts
Commit tag 1a656355b7c2c217898bde31c1929ab7029fe3b6
Files contracts/eosio.token/eosio.token.[ch]pp
No tests available
No ricardian contract sources were available
Business requirements and other engineering artifacts were found in the “ClickUp” document found at https://doc.clickup.com/d/h/hfd6b-9788/49593a316fe4392/hfd6b-1448

Executive Summary

No obvious signs of severe or critical security violations were detected in this audit.

Ricardian contract exists but is missing an entry for the **addvesting** action. Lack of ricardian contracts is not necessarily a risk but is a lost opportunity for ensuring that the coded logic is congruent with the identified business needs.

Participants

Primary Auditors were Jack DiSalvatore, with Ciju John providing a review.

Available Tools

Static analyzers

- **EOSafe** appears to be an Academic exercise last updated nearly 4 years ago
- **EOSlime** is a javascript tool that is also at least 2 years old
- **Klevoya** is a wasm analyzer tool using simple pattern matches

No static analyzers were used in this exercise. I tried to build the EOSafe tool, but it would not compile. The EOSlime tool was too complicated for me to determine how to build it. Klevoya was not useful because I could not generate an ABI file for the mining contract.

Known EOS Contract Vulnerabilities:

List origin github:slowmisg/**eos-smart-contract-security-best-practices**/Readme_EN.md commit tag 5f77e19e50373d341e17a003c492388e9891a2c0

- **Numerical Overflow:** when doing token math, use the `asset` object operands and do not perform operations on the `uint256` that comes from `asset.amount`
- **Authorization Check:** make sure function parameters are consistent with the caller and use `require_auth ` to check
- **Apply Check:** if the contract implements an `apply` function, bind each key action and code to meet the requirements, in order to avoid abnormal and illegal calls.

- **Transfer Error Prompt:** When processing a notification triggered by `require_recipient`, ensure that `transfer.to` is `_self`
- **Random Number Practice:** Random number generator algorithm should not introduce controllable or predictable seeds
- **Rollback Attack:** There is a time difference between when an API node initiates the request and synchronizes to the BP node. (In the context of a gambling dapp) Attackers use this vulnerability to place bets on the API node. If they find that there is no winning, they can return the EOS for each bet so that they can rejoin the game without cost. Attackers can repeat this process until they have a winning bet. One solution to this is for your dapp to use read/write separation and have Read-Only and Write-Only nodes to interact with the smart contract.

We relied on **visual inspection** to look for instances of code that were similar to the examples found in the above document as well as best practices accumulated through many years of experience.

Findings And Recommendations

We examined 1 contract action and 1 private function. The contract action is documented in clickup, but not the modifications to the private function.

- The commented out `retire` declaration on eosio.token.hpp line 158 can be removed.
- The comment out `clearvesting` action on eosio.token.hpp lines 235 - 244 is commented out and can be removed.
- If the vesting token is different from this contract token, then you could define the `vesting_quantity` (eosio.token.hpp#187) as an `extended_asset`. Then when you check the it in eosio.token.cpp#138 you could not only check that the symbols match, but also that the contract account matches. Therefore, you could prevent someone from creating a new fake token of the same symbol and trying to vest with that.

Table Signature	In ClickUp	Notes
typedef eosio::multi_index<"vestings"_n, vesting_item> vestings;	No	No issues.

Action Signature	In ClickUp	Notes
void token::retire(const asset &quantity, const string &memo)	No	Remark Code is commented out and can be removed.
void token::addvesting(const name &account, const time_point_sec &vesting_start, const uint32_t &vesting_length, const asset &vesting_quantity)	Yes	Remark line#215 the `vestings` table is scoped to itself, it could be initialized once in a constructor.

Function Signature	In ClickUp	Notes

void token::sub_balance(const name &owner, const asset &value)	No	Remark line#136 the `vestings` table is scoped to itself, it could be initialized once in a constructor. Major line#146 check that `vesting_length`is not zero to avoid a division by 0 error. Minor line#146 Since `vesting_quantity` and `vesting_length` cannot be negative, use `uint64_t` instead of `int64_t`.

Summary

Remark	Minor	Major	Critical
3	1	1	0

No methodology definitively proves the absence of vulnerabilities. Following assessment and remediation, modifications to an application, its platform, network environment, and new threat vectors may result in new application security vulnerabilities.