OBJECT COMPUTING

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

Alien Worlds atomicassets Contract Audit (EOS)

Source for this analysis is github.com/Alien-Worlds/eosdac-contracts
Commit tag: 722e38267153bb53b37431aa86bb0959bd5485e7
Original source repository: https://github.com/pinknetworkx/atomicassets-contract
Files: contracts/stakevote/stakevote.[ch]pp
Test: *tests source not available*
Ricardian contracts: *ricardian contracts not available*
Business requirements and other engineering artifacts: https://eosdac.io/tools/smart-contracts-explained/

Executive Summary

This is a fairly large contract, with 35 actions defined. It is also a very solid implementation. This review found only one remarkable issue. Given that the Alien Worlds clone of this contract is identical to the original, and we did not have a record of a previous audit, we chose to review the entire contract.

Remark	Minor	Major	Critical
1	0	0	0

Participants

Primary auditor is Phil Mesnier, with Ciju John providing a review.

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 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
- **Reentrancy Attack:** A reentrancy attack can occur when you create a function that makes an external call to another untrusted contract before it resolves any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the effects were resolved.

  This simplest example is when a contract does internal accounting with a balance variable and exposes a withdraw function. If the vulnerable contract transfers funds before it sets the balance to zero, the attacker can recursively call the withdraw function repeatedly and drain the whole 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

The examination looked at 8 tables, 35 actions, 9 helper functions and 0 tests. While there are no Ricardian contracts, each of the action and helper functions is preceded by a comment block which provided a decent synopsis of the function.

Remarks

- A magic number was found.

Minor

- None.

Major

- None.

Critical

- None.

Table/Singleton	Notes
collections_s	No issues
schemas_s	No issues
templates_s	No issues
assets_s	No issues
offers_s	No issues
balances_s	No issues
config_s	No issues
tokenconfigs_s	No issues

Action	Notes	Ricardian Contract
init	No issues	no
admincoledit	No issues	no
setversion	No issues	no
addconftoken	No issues	no
transfer	No issues	no
createcol	Remark: Magic number on line 109	no
setcoldata	No issues	no
addcolauth	No issues	no
remcolauth	No issues	no
addnotifyacc	No issues	no
remnotifyacc	No issues	no
setmarketfee	No issues	no
forbidnotify	No issues	no
createschema	No issues	no
extendschema	No issues	no
createtempl	No issues	no
locktemplate	No issues	no
mintasset	No issues	no
setassetdata	No issues	no
announcedepo	No issues	no
withdraw	No issues	no
backasset	No issues	no
burnasset	No issues	no
createoffer	No issues	no
Action	**Notes**	**Ricardian Contract**
canceloffer	No issues	no
acceptoffer	No issues	no

declineoffer	No issues	no
payofferram	No issues	no
logtransfer	No issues	no
lognewoffer	No issues	no
lognewtempl	No issues	no
logmint	No issues	no
logsetdata	No issues	no
logbackasset	No issues	no
logburnasset	No issues	no

Helper Functions	Notes
internal_transfer	No issues
internal_back_asset	No issues
internal_decrease_balance	No issues
notify_collection_accounts	No issues
check_has_collection_auth	No issues
check_name_length	No issues
get_assets	No issues
get_schemas	No issues
get_templates	No issues

Test Case	Scenario	Notes
[empty set]		

Test coverage = (number of tested actions) / (number of actions)
none

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.