OBJECT COMPUTING

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

Alien Worlds stakevote Contract Audit (EOS)

Source for this analysis is github.com/Alien-Worlds/eosdac-contracts
Commit tag: 722e38267153bb53b37431aa86bb0959bd5485e7
Original source for comparison github.com/eosdac/eosdac-contracts
Commit tag for comparison: f213b0f07c425912692c38e8b7e07ad99da9ddca

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 review effort limited its scope to the Alien-Worlds supplied extensions to the original EOSDAC supplied, and previously vetted contract. The coverage is limited to two new notification callback handlers and a common service function shared by both of these.

No issues of any severity above a remark were detected in this audit. No tests nor Ricardian contracts were found in the Stakevote contract directory.

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 3 actions and 2 tables. No tests or ricardian contracts were found

Remarks
- Balanceobsv is an action with no effect..

Minor
- None.

Major
- None.

Critical
- None.

Table/Singleton	Notes
config	No issues
weights	No issues

Action	Notes	Ricardian Contract
stakeobsv	No issues	no
balanceobsv	Remark - implementation is effectively a no-op.	no
updateconfig	No issues	no

Helper Functions	Notes

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.