OBJECT COMPUTING

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

Alien Worlds eosdactoken Contract Audit (EOS)

Source for this analysis is github: Alien-Worlds/eosdac-contracts
Commit tag 722e38267153bb53b37431aa86bb0959bd5485e7
Files: contracts/eosdactokens/eosdactokens.[ch]pp
Test: contracts/eosdactokens/eosdactokens.test.ts
Ricardian contracts: contracts/eosdactokens/eosdactokens.contracts.md
Business requirements and other engineering artifacts: https://eosdac.io/tools/smart-contracts-explained/

Executive Summary

No critical security violations were detected in this audit.

Remark	Minor	Major	Critical
7	0	0	0

Participants

Primary auditors are Jack DiSalvatore and 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

We examined 8 tables, 15 actions and 6 helper functions. The contract actions are documented on eosdac.io.

Remarks

- Replace “magic numbers” with descriptive constant variable names.
- Remove redundant comments.
- eosdactokens.hpp#121-122 remove commented out functions.
- eosdactokens.cpp be consistent with variables for readability. Ex: `same_payer` and `name{}` are used interchangeably.
- eosdactokens.cpp line#354 - When a user unstakes all of their tokens, it first emplaces a new staking row and then deletes it immediately in the `sub_stake` function.

Minor
-
Major
-
Critical
-

Table/Singleton	Notes
stakes	No issues.
unstakes	No issues.
staketime	No issues.
stakeconfig	Remark: Line#79 replace `60 * 60 * 24 * 3` with constant `THREE_DAYS` Remark: Line#80 replace `60 * 60 * 24 * 30 * 9` with constant `NINE_MONTHS`
members	Remark: Line#93 comment is redundant
memberterms	Remark: Line#104 move primary key `version` to top of struct for readability and consistency.
accounts	No issues.
stat	No issues.

Action	Notes	Ricardian Contract
create	No issues.	Yes
issue	No issues.	Yes
unlock	No issues.	Yes
burn	Remark: line#78 move `send_balance_notification` call to the end of the function to follow the convention of sending actions last.	Yes
transfer	Remark: line#122 move `send_balance_notification` call to the end of the function to follow the convention of sending actions last.	Yes
Test Case	**Scenario**	**Notes**
newmemterms	No issues.	Yes
memberreg	No issues.	Yes
memberunreg	No issues.	Yes
updateterms	No issues.	No

// staking		
xferstake	No issues. (why does this add stake?)	No
stake	No issues.	No
unstake	Remark: When a user unstakes all of their tokens, it is unnecessary to first emplace a new staking row (line#354) and then delete it immediately (in the `sub_stake` function).	No
staketime	No issues.	No
stakeconfig	No issues.	No
cancel	No issues.	No

Helper Functions	Notes
sub_balance	No issues.
add_balance	No issues.
sub_stake	No issues.
add_stake	No issues.
send_stake_notification	No issues.
send_balance_notification	No issues.

Test Case	Scenario	Notes
create	With invalid token symbol should fail with invalid symbol error	No issues.
	With negative token quantity should failed with invalid supply error	No issues. Remark: Line#46 use of magic numbers, could be a constant like `UINT64_T_MAX`.
	With negative token quantity should fail with supply must be positive error	No issues.
	With valid asset should succeed	No issues.
	With existing token should fail with existing token error	No issues.
issue	With invalid token symbol should fail with symbol error	No issues.
	With non existing token should fail with non-existing token error	No issues.
	With invalid quantity should fail with invalid quantity error	No issues. Remark: Line#107 use of magic numbers, could be a constant like `UINT64_T_MAX`.
	With negative quantity should fail with non-positive issue error	No issues.
	With invalid precision should fail with invalid precision error	No issues.
	To issuser should increase issuer balance	No issues.
	To account other than the issuer - should fail with assertion	No issues.
	To other - without a dac - should fail with DAC not found or symbol error	No issues.
	To other - with a dac - should increase the other balance	No issues.
unlock		No tests.
Test Case	**Scenario**	**Notes**
burn		
transfer	With staking enabled - with nothing staked from sender - should succeed	No issues.
	With staking enabled - with some staked but not enough liquid for transfer - should fail	No issues.
	With staking enabled - with some staked but enough liquid for transfer - should succeed	No issues.
	With some staked and exactly enough liquid for transfer - should succeed	No issues.
newmemterms		No tests.

memberreg		No tests.
memberunreg		No tests.
updateterms		No tests.
xferstake		No tests.
stake	Stake without staking enabled - should fail with staking not enabled error	No issues.
	With staking enabled - with invalid quantity should fail with dac not found for symbol error	No issues. Remark: actually fails with invaild quantity error. Uses magic numbers.
	With staking enabled - with negative quantity should fail with more than liquid error	No issues.
	With staking enabled - with more than available balance should fail with more than liquid error	No issues.
	With staking enabled - with correct amount should succeed	No issues.
	With staking enabled - staking again should add more	No issues.
	With staking enabled - again with more than liquid balance should fail with more than liquid error	No issues.
Test Case	**Scenario**	**Notes**
unstake	Without staking enabled with staking not enabled error - should fail	No issues.
	With staking enabled - should fail for negative quantity with non positive error	No issues.
	With staking enabled - should fail for amount in excess of stake with unstake over staked amount error	No issues.
	With staking enabled - should fail for no stake with no stake found error	No issues.
	With staking enabled - should succeed	No issues. Remark: lines#414-415 arbitrary magic number `20000`.
	With staking enabled - transfer should fail if not yet released	No issues.
	With staking enabled - transfer should succeed once released	No issues.
staketime		No tests.
stakeconfig		No tests.
cancel		No tests.

Test coverage 4 / 15 = ~27%

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.