OBJECT COMPUTING

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

Alien Worlds msigworlds Contract Audit (EOS)

Source for this analysis is github: Alien-Worlds/contracts/msigworlds
Commit tag: 722e38267153bb53b37431aa86bb0959bd5485e7
Forked github: eosdac/eosdac-contracts
Forked Commit tag: f213b0f07c425912692c38e8b7e07ad99da9ddca

Files: contracts/msigworlds/msigworlds.[ch]pp
Tests: contracts/msigworlds/msigworlds.tests.ts
Ricardian contracts: N/A
Business requirements and other engineering artifacts:
https://eosdac.io/tools/smart-contracts-explained/
contracts/daccustodian/README.md

Executive Summary

No critical security violations were detected in this audit.

Remark	Minor	Major	Critical
4	0	0	0

Participants

Primary auditors are Jack DiSalvatore, with Ciju John and Phil Mesnier 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

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 5 tables, 9 actions and 4 helper functions.

Remarks

- Proposals table primary key `id` not used as primary key in favor of `proposal_name`
- mix of `TABLE` keyword and `[[eosio:table(), eosio:contract()]]` syntax
- Action to blacklist contract actions, but no way to unblacklist them
- msigworlds.cpp#276 check that length of actions is less then max action call depth before entering loop

Minor

-

Major

- None

Critical

- None

Table/Singleton	Notes

typedef eosio::multi_index<
  "proposals"_n,
  proposal,
  indexed_by<
    "proposer"_n,
    const_mem_fun<
      proposal,
      uint64_t,
      &proposal::by_propser>
    >,
  indexed_by<
    "moddata"_n,
    const_mem_fun<
      proposal,
      uint64_t, &proposal::by_mod_date
    >
  >
> proposals;
	Remark: primary key \`id\` not used in favor of \`proposal_name\`

typedef eosio::multi_index<
  "approvals"_n,
  approvals_info
> approvals;
	No issues.

typedef eosio::multi_index<
  "invals"_n,
  invalidation
> invalidations;
	No issues.

typedef eosio::multi_index<
  "blockedactns"_n,
  blocked_action,
  indexed_by<
    "contractns"_n,
    const_mem_fun<
      blocked_action,
      uint128_t,
      &blocked_action::contract_and_actions
    >
  >
> blocked_actions;
	No issues.

using serial_singleton = eosio::singleton<"serial"_n, serial>;	**Remark**: mix of \`TABLE\` keyword and \`[[eosio:table(), eosio:contract()]]\` syntax

Action	Notes	Ricardian Contract
ACTION propose(name proposer, name proposal_name, std::vector<permission_level> requested, name dac_id, std::map<std::string, std::string> metadata, ignore<transaction> trx);	No issues.	No
ACTION approve(name proposal_name, permission_level level, name dac_id, const std::optional<eosio::checksum256> proposal_hash);	No issues.	No

ACTION unapprove(name proposal_name, permission_level level, name dac_id);	No issues.	No
ACTION deny(name proposal_name, permission_level level, name dac_id);	No issues.	No
ACTION cancel(name proposal_name, name canceler, name dac_id);	No issues.	No
ACTION exec(name proposal_name, name executer, name dac_id);	**Remark**: Line#276 check that length of actions is less then max action call depth before entering loop.	No
ACTION cleanup(name proposal_name, name dac_id);	No issues.	No
ACTION invalidate(name account, name dac_id);	No issues.	No
ACTION blockaction(name account, name action, name dac_id);	No issues. **Remark**: Consider writing an action to unblock an action.	No

Helper Functions	Notes
template <typename Function> std::vector<permission_level> get_approvals_and_adjust_table(name self, name proposal_name, Function &&table_op, name dac_id);	No issues.
uint64_t next_id(name dac_id);	No issues.
transaction_header get_trx_header(const char *ptr, size_t sz)	No issues.
bool trx_is_authorized(const std::vector<permission_level> &approvals, const std::vector<char> &packed_trx)	No issues.

Test Case	Scenario	Notes
block actions	With wrong auth - should fail with auth error	No issues.
	With correct auth - should succeed	No issues.
	Add existing action - should fail	No issues
propose	With wrong auth - should fail with auth error	No issues.
	With correct auth - with passed expiration - should fail with expired error	No issues.
	With correct auth - with context free actions included - should fail with context free actions error	No issues.
	With correct auth - with blocked action - should fail with blocked action error	No issues.
	With correct auth - with all correct params - should succeed	No issues.
	With correct auth - with all correct params - should populate proposals table	No issues.
	With correct auth - with all correct params - should populate approvals table	No issues.
	With correct auth - with existing msig with same name - should fail with duplicate name	No issues.
approve	With wrong auth - should fail with auth error	No issues.
	With correct auth - with non-existing proposal - should fail with prop not found error	No issues,
Test Case	**Scenario**	**Notes**
	With correct auth - with existing proposal - for unrequested auth - should fail with approval not on list found error	No issues.
	With correct auth - with existing proposal - for requested auth - should succeed	No issues.
	With correct auth - with existing proposal - for requested auth - should update approvals table	No issues.

	With correct auth - with existing proposal - for requested auth - should update proposal table	No issues.
	With correct auth - with existing proposal - for requested auth - should update proposal earliest exec date	Remark: No assertion check.
unapprove	with wrong auth - should fail with auth error	No issues.
	With correct auth - with non-existing proposal - should fail with prop not found error	No issues.
	With correct auth - with existing proposal - for previously unapproved auth - should fail with approval not on list found error	Remark: fails with ‘no approval previously granted’ error
	With correct auth - with existing proposal - for previously granted approval auth - should succeed	No issues.
	With correct auth - with existing proposal - for previously granted approval auth - should update approvals table	No issues.
	With correct auth - with existing proposal - for previously granted approval auth - should update proposal modification date	No issues.
deny	With wrong auth - should fail with auth error	No issues.
	With correct auth - with non-existing proposal - should fail with prop not found error	No issues.
	With correct auth - with existing proposal - for previously denied auth - should work	No issues.
	With correct auth - for previously granted approval auth - should succeed	No issues.
	With correct auth - for previously granted approval auth - should update approvals table	No issues.
Test Case	**Scenario**	**Notes**
	With correct auth - for previously granted approval auth - should update proposal modification date	No issues.
cancel	With wrong auth - should fail with auth error	No issues.
	With correct auth - with non-existing proposal - should fail with prop not found error	No issues.
	With correct auth - with existing proposal - for proposal from different creator auth - should fail with approval not on list found error	Remark: instead fails with ‘cannot cancel until expiration’ error
	With correct auth - when chancellor is the proposer - should succeed	No issues.
	With correct auth - when chancellor is the proposer - should update approvals table	No issues.
	With correct auth - when chancellor is the proposer - should update proposal modification date	No issues.
	With correct auth - after proposal has already been canceled - should fail with state error	No issues.
exec	prop2 should have the correct id	No issues.

	with wrong auth - should fail with auth error	No issues.
	with correct auth - with non-existing proposal - should fail with prop not found error	No issues.
	with correct auth - for proposal from that is already canceled - should fail with proposal not in pending state	No issues.
	Without sufficient auth to approve - should fail with transaction auth error	No issues.
	With expired transaction - propex should have the correct id	No issues.
	With expired transaction - should fail with expired error	No issues.
	With sufficient auth for transaction - should succeed	No issues.
	With sufficient auth for transaction - propgood should have the correct id	No issues.
Test Case	**Scenario**	**Notes**
	With sufficient auth for transaction - should update approvals table	No issues.
	With sufficient auth for transaction - should update proposal modification date	No issues.
	After proposal has already been executed - should fail with state error	No issues.
cleanup	With non existent proposal - should fail with not found error	No issues.
	Before being executed or canceled - propclean should have the correct id	No issues.
	Before being executed or canceled - should fail with wrong state error	No issues.
	With completed proposal - should succeed	No issues.
	With completed proposal - should erase the proposals and approvals record	No issues.

Test Coverage

The amount of actions tested divided by the number of total actions.

8 / 9 = 90%

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.