CD SECURITY

AUDIT REPORT

SUPRA VRF

April 2023

Introduction

A time-boxed security review of the **Supra VRF Subscription** protocol was done by **ddimitrov22** and **chrisdior4** , with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts.

About **Supra**

Supra VRF is an on-chain and off-chain application for randomness that aims to provide unbiased randomness. The process of clients accessing the app is as follows:

1. Clients need to contact the SupraAdmin and provide a wallet address that will be whitelisted.
2. Clients need to deposit funds and maintain a minimum balance inside the `DepositContract`.
3. After that, clients can add as many contracts as they want to.
4. The fee for each contract will be billed to the Client address which added the contract.

After being whitelisted and funded, Client contracts can request randomness. The process is described below:

1. A contract call `generateRequest` inside `SupraRouterContract` with all the input parameters required.
2. The `DepositContract` will be internally called to perform input validation.
3. If all checks are passed, the `SupraRouterContract` will call `rngRequest` inside `SupraGeneratorContract`.
4. `rngRequest` emits an event with all the information needed for off-chain purposes.
5. A whitelisted `FreeNode` makes an RPC call to `VRFNode` which generates a signature and returns it to the `FreeNode`.
6. The `FreeNode` will call `generateRngCallback` to `verify` the signature and `collectFund` from the `clientAddress`.

More docs.

Threat Model

Roles & Actors

- `Clients` - wallet addresses which can add contracts and pay for each request.
- `Contracts` - contracts that `generateRequest`.

- `SupraAdmin` - the owner of Supra have access to a number of onlyOwner-specific functions(e.g can add/remove clients to the whitelist).
- `FreeNodes` - off-chain nodes that use information to process the requests.
- `Developer` - the developer that will manage the contracts.
- `Approver` - address which can approve changes like confirming cold wallets.

Security Interview

Q: What in the protocol has value in the market?

A: The funds deposited by the clients.

Q: What is the worst thing that can happen to the protocol?

A: To lose ownership of the contracts and randomness to be rigged.

Q: In what case can the protocol/users lose money?

A: If an attacker is able to drain funds from the `DepositContract`.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 448b8d934a0591e10aea871d4a405e3fa7aa28c4**

Scope

The following smart contracts were in scope of the audit:

- `DepositContrac.sol`
- `IDepositContrac.sol`
- `SupraGeneratorContract.sol`
- `ISupraRouterContract.sol`
- `SupraRouterContract.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 4 issues
- Low: 3 issues
- Informational: 11 issues

Findings Summary

ID	Title	Severity
[M-01]	Lack of two-step role transfer	Medium
[M-02]	A `client` can lose money if it is removed from the `whitelist`	Medium
[M-03]	Use `call` instead of `transfer` when sending ETH	Medium
[M-04]	Missing input validation can lead to setting wrong values	Medium
[L-01]	It is possible to `updateSubscription` with `endTime` in the past	Low
[L-02]	`isContract` checks can be bypassed	Low
[L-03]	Combine `generateRequest` functions into a single function	Low
[I-01]	Unused variables can be deleted	Informational
[I-02]	Unnecessary `require` check	Informational
[I-03]	Functions visibility can be changed	Informational
[I-04]	Prefer Solidity Custom Errors over require statements with strings	Informational
[I-05]	Typos in error message and comments	Informational
[I-06]	Missing functions	Informational
[I-07]	Repeating code can be made into a modifier	Informational
[I-08]	Some functions are missing event emission	Informational
[I-09]	Missing NatSpec documentation	Informational
[I-10]	Contracts are not inheriting their interface	Informational
[I-11]	Use newer pragma statement	Informational

Detailed Findings

[M-01] Lack of two-step role transfer

Severity

Impact: High, because the `SupraAdmin` role has access to crucial functions

Likelihood: Low, because it requires an error or compromised `SupraAdmin`

Description

The contracts lack two-step role transfer. All of the contracts use the `onlyOwner` modifier which shows that this role is important for the project. Especially the `DepositContract` where the funds deposited by clients will be stored. The basic check for a zero-address is performed, however the case when the address receiving the role is inaccessible is not covered properly. Also, it can be seen in the documentation that this is one of the main security concerns of the team. The ownership transfer should be done with great care and two-step role transfer should be preferable.

Recommendations

Use Ownable2Step by OpenZeppelin.

CLIENT

Acknowledged - corrected.

[M-02] A `client` can lose money if it is removed from the `whitelist`

Severity

Impact: High, because the `client` will suffer financial loss

Likelihood: Low, because a malicious/compromised owner is required

Description

When a `client` is removed from the `whitelist`, it loses access to the functions inside of the smart contracts. This is a problem when a `client` is removed but still have funds deposited in the `DepositContract`. Thus, the `client` will not be able to withdraw his funds because of the first `require` statement in `withdrawFundClient`:

```
    function withdrawFundClient(uint256 _amount) external whenNotPaused {
        require(_amount <= checkClientFund(msg.sender) ,"Amount exceeds
your deposit");
        ...
    }
```

The `checkClientFund` will revert because it will check if the caller is whitelisted. Even worse case is if a malicious owner decides to remove clients from the `whitelist` and leaving them without a chance to withdraw their money.

Recommendations

Call `withdrawFundClient` inside the `removeClientFromWhitelist` function so the `clients` are returned any left money in their balance.

CLIENT

Acknowledged - corrected.

[M-03] Use `call` instead of `transfer` when sending ETH

Severity

Impact: Medium, because if the recipient is a smart contract or a specific multisig, the transaction may fail

Likelihood: Medium, because the `transfer` method might be deprecated in the future

Description

The `transfer` method is used to withdraw funds from the contract. If the recipient is a smart contract that has a `receive` or `fallback` function that takes up more than the 2300 gas which is the limit of `transfer` or a specific multi-sig wallet, so usage of `transfer` is discouraged. Furthermore, there are proposals and discussions to deprecate the `transfer` method (check here).

Recommendations

Use a `call` with value instead of `transfer` and add a `require` statement to check if it is successful.

CLIENT

Acknowledged - corrected.

[M-04] Missing input validation can lead to setting wrong values

Severity

Impact: High, as it can upgrade contracts to zero address

Likelihood: Low, as it requires owner error/misconfiguration

Description

The `updateDepositContract` and `updateGeneratorContract` are validated to be contracts but are missing zero address checks which means that those contracts can be upgraded to zero address. Also, there is missing zero address and values checks in the `constructor` of `DepositContract`. This means that `minBalanceLimitSupra` can be set to 0 and none of the contracts are protected from setting to `address(0)`.

Recommendations

Add appropriate `require` statements to check for zero address and `msg.value > 0` where needed.

CLIENT

Acknowledged - corrected.

[L-01] It is possible to `updateSubscription` with `endTime` in the past

The `updateSubscription` can be called by `SupraAdmin` to set `_newEndTime` for a specific subscription. However, the only check is the following `require` statement:

```
require(_newEndTime > subscriptionPeriod[_clientAddress].startDate, "New time
should be in future");
```

This is wrong because the `_newEndTime` can still be in the past. Consider changing it to `_newEndTime > block.timestamp + X days` to make sure that it is in the future.

CLIENT

Acknowledged - corrected.

[L-02] `isContract` checks can be bypassed

The `require(!isContract(_clientWalletAddress),"")` check is used in many places to make sure the input parameter is an EOA. This check can easily be bypassed. During contract creation, the code of the contract is equal to zero, and `isContract` check will return false if a method is called in the constructor. Also, preventing a contract is an anti-pattern in security and interoperability considerations.

CLIENT

Acknowledged - corrected.

[L-03] Combine `generateRequest` functions into a single function

There are two functions with the same name - `generateRequest`. The only difference between them is the `_clientSeed` input parameter. Consider removing the one without the `_clientSeed` because the only difference is that it encodes the same parameters with 0 (which is the default value if `_clientSeed` is not used) instead of `_clientSeed`. This will make the `SupraRouterContract` less confusing and more readable, and will gas optimize the contract at the same time.

CLIENT

Acknowledged.

[I-01] Unused variables can be deleted

The below variables are not used and can be deleted:

```
    bool private ownerApproved;
    bool private approverApproved;
```

CLIENT

Acknowledged - corrected.

[I-02] Unnecessary `require` check

File: `DepositContract.sol`

`require(!isContract(_freeNodeWallet), "Cannot be a contract address");` in `removeFreeNodeFromWhitelist` is not needed because it is impossible to add `FreeNode` to the whitelist because of the same check. Consider removing it.

CLIENT

Acknowledged - corrected.

[I-03] Functions visibility can be changed

Functions that are marked as `public` and are not called inside the same contract can be set to `external` (e.g. `generateRequest`). `rngRequest` even can be set to internal because it is only possible to call if the `msg.sender == supraRouterContract`.

CLIENT

Acknowledged - corrected.

[I-04] Prefer Solidity Custom Errors over require statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Remove all require statements and use Custom Errors instead.

CLIENT

Acknowledged - corrected.

[I-05] Typos in error message and comments

`reqired` -> `required`

`tha` -> `than`

`upgradibility` -> `upgradeability`

`addresss` -> `address`

`emmited` -> `emitted`

`fullfill` -> `fullfil`

`succcess` -> `success`

CLIENT

Acknowledged - corrected.

[I-06] Missing functions

There are functions in the docs that are not present in the contracts (e.g. `checkDepositByClient_`, `reconcileExecution_`). Make sure the docs and the actual code are aligned.

CLIENT

Acknowledged - corrected.

[I-07] Repeating code can be made into a modifier

File: `DepositContract.sol`

There are 5 instances of a repeated`isClientWhitelisted` require statement that can be made into a modifier which will make the code look more neat and organised. Also it will improve the readability.

CLIENT

Acknowledged - corrected.

[I-08] Some functions are missing event emission

File: `DepositContract.sol`

Functions like `confirmColdWallet` and `setMinBalanceClient` are not emitting an event. State-changing methods should emit events so that off-chain monitoring can be implemented. Make sure to emit a proper event in each state-changing method to follow best practices.

CLIENT

Acknowledged - corrected.

[I-09] Missing NatSpec documentation

File: `DepositContract.sol`

Function `updateGeneratorRouter` is missing the `_newRouter`@param field. Consider adding it.

CLIENT

Acknowledged - corrected.

[I-10] Contracts are not inheriting their interface

`DepositContract.sol` and `SupraRouter.sol` are not inheriting their interface while this is a best practice for the contract implementations to inherit their interface definition. Doing so would improve the contract's clarity, and force the implementation to comply with the defined interface. `SupraRouter.sol` contract is inheriting `IDepositContract` and not its own interface. Also do not forget to include the `override` keyword whenever you use a method inherited from the interface.

CLIENT

Acknowledged - corrected.

[I-11] Use newer pragma statement

All of the contracts use version `0.8.10` while the latest version is `0.8.19`. Consider upgrading the version to a newer one to use bugfixes and optimizations in the compiler.

CLIENT

Acknowledged - corrected.