VANTAGEPOINT
security at the speed of development

Smart Contract Audit of Algorand VRF Randomness Beacon

Final Report

Prepared for:

Algorand Foundation Ltd

28 Oct 2022

FOR GENERAL DISTRIBUTION

Table Of Content

1. Executive Summary 3
   1.1. Overview 3
2. Project Details 4
   2.1. Scope 4
   2.2. Status 4
3. Risk Assessment 5
   3.1. Summary of Vulnerabilities 5
   3.2. Vulnerabilities Statistics 6
4. Detailed Description of Vulnerabilities 7
   4.1. ARC-04 Smart Contract with Router Class without Pragma or pragma 7
   4.2. Lack of Documentation on Graceful Handling of On-chain Unavailability 10
5. Appendix 12
   5.1. Disclaimer 12
   5.2. Risk Rating 12

1. Executive Summary

1.1. Overview

Vantage Point Security Pte Ltd was engaged by Algorand Foundation to conduct an Algorand smart contract security audit of the Algorand Verifiable Random Function (VRF) oracle. The VRF smart contract validate inputs provided by off-chain components and commits them as a global state once VRF proof is verified. The committed global state of the smart contract can be utilized by other smart contracts as a source of on-chain randomness.

PyTeal Code

The Algorand smart contract audit was conducted to identify security vulnerabilities, weaknesses and any instances of non-compliance to best practices within the smart contract. Algorand smart contract security review was conducted based on the following materials provided.

- https://github.com/ori-shem-tov/vrf-oracle
  Commit ID 6cb380b8eea73142b6bc943d70393cdcfcc71fb1

Vantage Point performed this review by first understanding the documentation of Algorand VRF Oracle Smart Contract and PyTeal code. We sought clarifications on potential issues, discrepancies, and flaws within the smart contract's logic through discussions with the Algorand Foundation team. The following observational issues were noted from this review.

Use of Router Class without Pragma or pragma

- As the smart contract makes use of the Router class which abstracts routing of different application calls with and without application arguments, all routing is handled by the Router class based on the registered methods. However, as the Router class is subject to potential backward-incompatible changes in future releases of PyTeal, it is recommended to pin the exact version of PyTeal compiler to avoid any unexpected changes to the compiled Teal code in the future if a different version of PyTeal compiler other than 0.17.0 is used.

Lack of Documentation on Graceful Handling of On-chain Unavailability

- As the smart contract serves as an on-chain randomness beacon for the Algorand ecosystem, it is important to highlight the risks involved in using an on-chain randomness beacon to avoid other smart contracts who are integrating the randomness from the beacon to their logic. To avoid loss or lock-up of assets, biased logic outcome or unexpected behavior of the smart contract, it is recommended to clearly highlight the risks and technical details related to scenarios when randomness is lost or no longer available.

Other than the highlighted recommended best practices on the pinning of PyTeal compiler version and documentation for developers in integrating the randomness beacon to other smart contracts, no other issues were observed during the audit. The smart contract makes good use of new VRFVerify opcode and is composable for its intended purpose where it would be called by other applications that require a verified and trust-worthy source of on-chain random number for its application logic. The outcome of this Algorand smart contract security audit engagement is provided as a detailed technical report that provides the smart contract owners a full description of the vulnerabilities identified, the associated risk rating foreach vulnerability, and detailed recommendations that will resolve the identified technical issue.

2. Project Details

2.1. Scope

App Name	Smart Contract Audit of Algorand VRF Randomness Beacon
Testing Window	22 Aug 2022 to 9 Sep 2022
Target URL	https://github.com/ori-shem-tov/vrf-oracle
Svn / Git Revision Number	6cb380b8eea73142b6bc943d70393cdcfcc71fb1
Project Type	Smart Contract Audit
App Type	Algorand Smart Contract
Items Completed	Smart Contract Audit for VRF
Issue Opening Date	22 Sep 2022 ● ARC-04 Smart Contract with Router Class without Pragma or pragma [Low] ● Lack of Documentation on Graceful Handling of On-chain Unavailability [Low]
Issue Closing Date	28 Sep 2022 ● ARC-04 Smart Contract with Router Class without Pragma or pragma [Low] 20 Oct 2022 ● Lack of Documentation on Graceful Handling of On-chain Unavailability [Low]

2.2. Status

Component	Review Type	Status
Security Audit	Smart Contract Audit	Completed

3. Risk Assessment

This chapter contains an overview of the vulnerabilities discovered during the project. The vulnerabilities are sorted based on the scoring categories CRITICAL, HIGH, MEDIUM and LOW. The category OBSERVATIONAL refers to vulnerabilities that have no risk score and therefore have no immediate impact on the system.

3.1. Summary of Vulnerabilities

Vulnerability Title	Risk Score	Closed
ARC-04 Smart Contract with Router Class without Pragma or pragma	Low	☑
Lack of Documentation on Graceful Handling of On-chain Unavailability	Low	☑

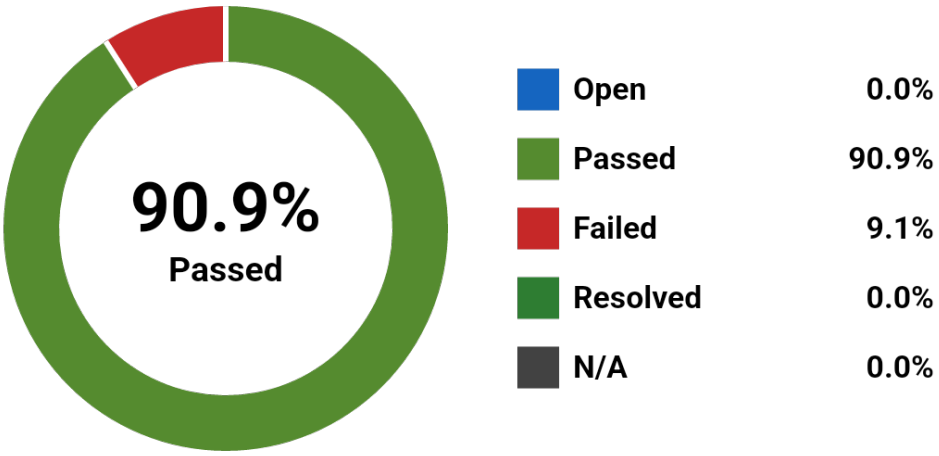
3.2. Vulnerabilities Statistics

Findings Overview

Total	Open	Closed
2	0	2

0	Critical	No findings
0	High	No findings
0	Medium	No findings
2	Low	0 Open, 2 Closed.
0	Observational	No findings

Test Case Status



4. Detailed Description of Vulnerabilities

4.1. ARC-04 Smart Contract with Router Class without Pragma or pragma

Low

Closed

BACKGROUND

ARC-04 Algorand Smart Contract Transaction Calling Conventions is a standard for encoding smart contract's method calls. ARC-04 allows wallets and dapp frontends to properly encode call transactions based on the interface description and explorers to show details of invoked methods. Smart contracts complying to ARC-04 standards respond to both method calls and bare app calls. To make it easier for an application to route across many base app calls and methods, PyTeal introduced Router class which adheres to the ARC-4 ABI conventions with respect to when methods and base app calls. As Router class is still expecting backwards-incompatible changes, it is recommended use pragma/Pragma expression to pin the version of PyTeal in the source code.

DESCRIPTION

Affected File

- https://github.com/ori-shem-tov/vrf-oracle/blob/beacon/pyteal/main.py

The smart contract in scope is an ARC-04 compliant smart contract which makes use of PyTeal's Router class for routing contract's methods. About Router class, PyTeal Documentation states below.

PyTeal Documentation - https://pyteal.readthedocs.io/en/stable/abi.html#creating-an-arc-4-program

```
Warning - Router usage is still taking shape and is subject to backwards
incompatible changes. Feel encouraged to use Router and expect a best-effort
attempt to minimize backwards incompatible changes along with a migration
path. For these reasons, we strongly recommend using pragma or the Pragma
expression to pin the version of PyTeal in your source code. See Version
Pragmas for more information.
```

As noted above, Router class is still expecting changes and therefore, it is strongly recommended to use Pragma or pragma to pin the version of PyTeal in the source code. However, in the affected code, pragma or Pragma was not observed.

PyTeal - main.py - Line 459-476

```
if __name__ == '__main__':
 compiled, clear, contract = vrf_beacon_abi().compile_program(version=7)
file_name = 'vrf_beacon_abi_approval.teal' # pylint: disable = C0103
 with open(file_name, 'w', encoding='utf8') as f:
 f.write(compiled)
 print(f'compiled {file_name}')
file_name = 'vrf_beacon_abi_clear.teal' # pylint: disable = C0103
 with open(file_name, 'w', encoding='utf8') as f:
 f.write(clear)
 print(f'compiled {file_name}')
file_name = 'contract.json' # pylint: disable = C0103
 with open(file_name, 'w', encoding='utf8') as f:
 print(json.dumps(contract.dictify(), indent=4), file=f)
 print(f'compiled {file_name}')
```

IMPACT

As PyTeal's Router class may have backward-incompatble changes in the future, if the PyTeal compiler version is not pinned, this PyTeal smart contract may compile in an expected way which may result in an unwanted routing of application calls.

RECOMMENDATIONS

It is recommended to make use of Pragma or pragma to pin the version of PyTeal in the source code to cater for potential backward-incompatable changes to the Router class.

Sample Code

```
if __name__ == '__main__':
    compiled, clear, contract = vrf_beacon_abi().compile_program(version=7)
    pragma(compiler_version="0.17.0")
    file_name = 'vrf_beacon_abi_approval.teal'  # pylint: disable = C0103
    with open(file_name, 'w', encoding='utf8') as f:
        f.write(compiled)
        print(f'compiled {file_name}')
    file_name = 'vrf_beacon_abi_clear.teal'  # pylint: disable = C0103
    with open(file_name, 'w', encoding='utf8') as f:
        print(f'compiled {file_name}')
    file_name = 'contract.json'  # pylint: disable = C0103
    with open(file_name, 'w', encoding='utf8') as f:
        print(json.dumps(contract.dictify(), indent=4), file=f)
        print(f'compiled {file_name}')
```

Do note that above is an example and correct range of compiler version should be determined and specified in the code.

COMMENT

Reviewed on 24 Sep 2022

It was noted that the commit 0cd21d5455a2d4ad1cb9cea9eae334a31606a688 from Algorand Foundation team addresses the issue identified by pinning the PyTeal compiler version to 0.17.0 through the use of pragma. With the below highlighted changes to the code, any attempt made to compile the PyTeal code using a different PyTeal compiler other than 0.17.0 would return an error.

Code Snippet - https://github.com/ori-shem-tov/vrf-oracle/blob/beacon/pyteal/main.py

```
if __name__ == '__main__':
    # this will immediately fail if the current PyTeal version does not
satisfy the
    # version constraint
    pragma(compiler_version="0.17.0")
    compiled, clear, contract = vrf_beacon_abi().compile_program(version=7)
    file_name = 'vrf_beacon_abi_approval.teal'  # pylint: disable = C0103
    with open(file_name, 'w', encoding='utf8') as f:
        f.write(compiled)
        print(f'compiled {file_name}')
    file_name = 'vrf_beacon_abi_clear.teal'  # pylint: disable = C0103
    with open(file_name, 'w', encoding='utf8') as f:
        f.write(clear)
        print(f'compiled {file_name}')
    file_name = 'contract.json'  # pylint: disable = C0103
    with open(file_name, 'w', encoding='utf8') as f:
        print(json.dumps(contract.dictify(), indent=4), file=f)
        print(f'compiled {file_name}')
```

Code Snippet - https://github.com/ori-shem-tov/vrf-oracle/blob/beacon/pyteal/requirements.txt

```
py-algorand-sdk==1.16.0
pyteal==0.17
```

REFERENCES

PyTeal - ABI Support
https://pyteal.readthedocs.io/en/stable/abi.html?highlight=Router#abi-support

PyTeal - Pragma
https://pyteal.readthedocs.io/en/stable/api.html?highlight=Router#pyteal.Pragma

4.2. Lack of Documentation on Graceful Handling of On-chain Unavailability

Low

Closed

BACKGROUND

On-chain oracles such as VRF Randomness beacon enable access to off-chain information to on-chain smart contracts so that the smart contract can process the acquired information and process them based on the smart contract logic. However, in a situation where information from on-chain oracles can no longer be trusted or become unavailable, other smart contracts which were making use of them may have the funds locked, behave unexpectedly or even suffer a financial loss.

DESCRIPTION

Affected Code

- https://github.com/ori-shem-tov/vrf-oracle/blob/beacon/pyteal/main.py

The smart contract in scope is an on-chain oracle which serves as a randomness beacon for the Algorand ecosystem. The smart contract leverages on the Verifiable Random Function (VRF) to serve as a random number generator which can be verified on-chain through the use of VRFVerify opcode introduced in Teal 7. Currently, off-chain components submit application call transactions every 8 blocks to the randomness beacon smart contract and have their random number validated through VRFVerify before it is saved as a global state. Also, the smart contract is permission-less in a sense that any Algorand account or even a smart contract can submit a valid application call and have the random number committed, as long as the verification of VRF proof is successful.
With the understanding of the above, the randomness beacon relies on keeping its private key safe. Once the private key is compromised, the output of VRF becomes predictable and randomness is lost. If other smart contracts integrate the random number from the randomness beacon into their logic, there. If there is no logic that gracefully falls back to alternative source of randomness or pauses state-changing operations, any smart contract that integrated the randomness beacon may face issues noted below.

Potential Issues with No Fallback or Graceful Handling of Randomness Unavailability

- Smart contracts that require randomness may not be able to approve any transaction and lock up assets
- Smart contracts that require randomness may not be making fair decisions or working as intended as randomness is lost

With the above risks identified in using the randomness beacon without fallback mechanism or graceful handling of randomness unavailability, it is very necessary to highlight such points in the documentation so that other smart contracts built with randomness beacon do not suffer from such issues.
Provided documentation related to the VRF randomness beacon at the time of audit did not have sufficient information on the risks involved and recommended best practices for using randomness beacon from smart contracts.

IMPACT

Smart contracts which make use of on-chain oracles without proper handling logic for unavailability or compromise of related infrastructure may suffer lock-up of assets, unavailability or unexpected behavior due to use of biased or false information.

RECOMMENDATIONS

To avoid lock-up, biased behavior and loss of assets, it is recommended to highlight the risks of using on-chain oracles, especially when related secrets, keys or infrastructure have been compromised or no longer available, and provide recommendations on best practices or options such as having another smart contract that works as an adapter between the on-chain oracle and other smart contracts or have a logic that falls-back to alternative source of information or suspends state changes temporarily upon unavailability of reliable on-chain oracle. Following points are some of the options often considered in the context of on-chain smart contracts where information is fed from off-chain components such as code running docker instance.

- Use of another smart contract "VRF Beacon Status Updater" which can only be updated by multisignature accounts. The smart contract should indicate the validity of the randomness on-chain through a global state so that other smart contracts can easily access the information about the validity of the randomness on-chain and have appropriate logic to handle in the event of compromise. This allows other smart contracts to pause any logics granting win/lose/transfer of assets based on the random number to protect any decision being made based on a compromised random number.
- Use of another smart contract "VRF Beacon Pointer" which can only be updated by a multisignature account that updates information such as availability of the randomness and applicationID of the currently valid VRF smart contract so that it points to the correct one all the time. This ensures continuity for the smart contracts making use of the VRF smart contract as the AppID can be updated by multisignature account. As long as multisignature accounts aren't compromised, it is possible to create another VRF smart contract with a new private/public key pair and have the "VRF Beacon Pointer" smart contract point to it so that the other smart contracts can check on the new AppId updated on the VRF Beacon Pointer and use it as a trusted randomness sources on-chain.

COMMENT

Reviewed on 23 Sep 2022

It was noted that the Algorand Foundation team provided more detailed documentation on the risks involved and best practices on using randomness beacon from smart contracts to properly handle the unavailability of randomness to prevent potential issues such as lock-up of smart contracts or loss of funds.

Provided Documentation

- https://developer.algorand.org/articles/usage-and-best-practices-for-randomness-beacon/

5. Appendix

5.1. Disclaimer

The material contained in this document is confidential and only for use by the company receiving this information from Vantage Point Security Pte. Ltd. (Vantage Point). The material will be held in the strictest confidence by the recipients and will not be used, in whole or in part, for any purpose other than the purpose for which it is provided without prior written consent by Vantage Point. The recipient assumes responsibility for further distribution of this document. In no event shall Vantage Point be liable to anyone for direct, special, incidental, collateral or consequential damages arising out of the use of this material, to the maximum extent permitted under law.

The security testing team made every effort to cover the systems in the test scope as effectively and completely as possible given the time budget available. There is however no guarantee that all existing vulnerabilities have been discovered. Furthermore, the security assessment applies to a snapshot of the current state at the examination time.

5.2. Risk Rating

All vulnerabilities found by Vantage Point will receive an individual risk rating based on the following four categories.

Critical

A CRITICAL finding requires immediate attention and should be given the highest priority by the business as it will impact business interest critically.

High

A HIGH finding requires immediate attention and should be given higher priority by the business.

Medium

A MEDIUM finding has the potential to present a serious risk to the business.

Low

A LOW finding contradicts security best practices and have minimal impact on the business.

Observational

An OBSERVATIONAL finding relates primarily to non-compliance issues,security best practices or are considered an additional security feature that would increase the security stance of the environment which could be considered in the future version of smart contract.