Competitive Security Assessment

PeanutV4

Dec 27th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

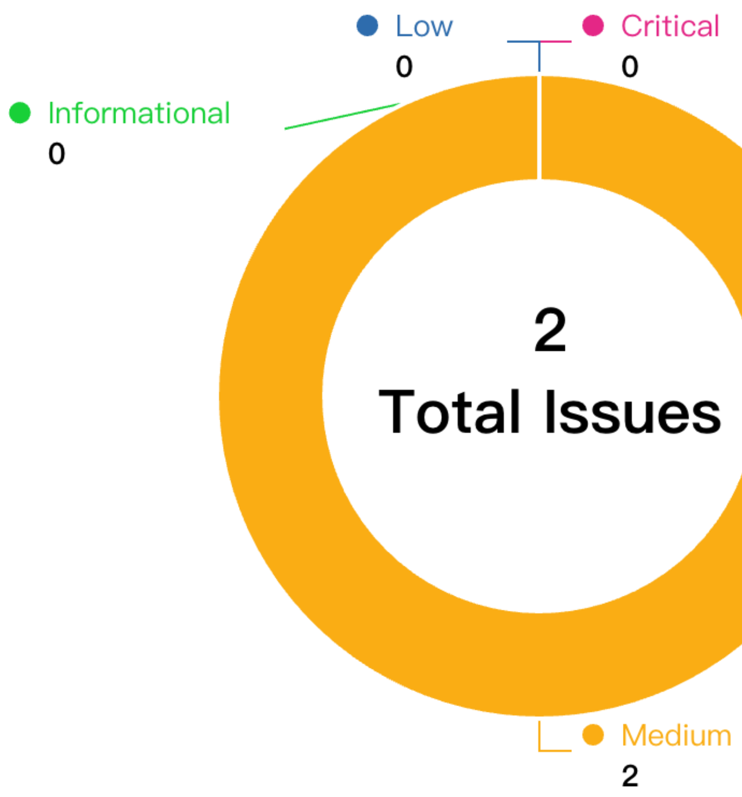
Project Detail

Project Name	PeanutV4
Platform & Language	Solidity
Codebase	• https://github.com/peanutprotocol/peanut-contracts • audit commit - 808f76ebb78154551f1facff31c878226fd27530 • final commit - 3da9f24376aa634ed85d83c076086f2732633e18
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./src/V4/PeanutV4.sol	2a457b4ac4abb02b0c8092fd25357638ab2841ebeb904e6a529b8cd1fae55c9e

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
PV4-1	bypass `withdrawDepositSender` timestamp check	Logical	Medium	Acknowledged	toffee
PV4-2	Need sign more information	Signature Forgery or Replay	Medium	Fixed	zigzag

PV4-1:bypass `withdrawDepositSender` timestamp check

Category	Severity	Client Response	Contributor
Logical	Medium	Acknowledged	toffee

Code Reference

- code/src/V4/PeanutV4.sol#L324

```
324:function withdrawDeposit(
```

Description

toffee : in the function `withdrawDepositSender()` there is a check to make sure the withdraw cannot be done until 24 hours passing the initial deposit.

```
        require(block.timestamp >= _deposit.timestamp + 24 hours, "NOT 24 HOURS YET");
```

However, the `withdrawDeposit` lacks the same check on `_deposit.timestamp`, makes it possible for anyone to call `withdrawDeposit` with its own address as `_recipientAddress` and bypass the withdraw time lock.

Recommendation

toffee : add below check in the `withdrawDeposit` function

```
        require(block.timestamp >= _deposit.timestamp + 24 hours, "NOT 24 HOURS YET");
```

Client Response

Acknowledged.Yes, thanks! We have removed the timestamp check completely as of this commit https://github.com/peanutprotocol/peanut-contracts/blob/3da9f24376aa634ed85d83c076086f2732633e18/src/V4/PeanutV4.sol

PV4-2:Need sign more information

Category	Severity	Client Response	Contributor
Signature Forgery or Replay	Medium	Fixed	zigzag

Code Reference

- code/src/V4/PeanutV4.sol#L336

```
336:_recipientAddressHash == ECDSA.toEthSignedMessageHash(keccak256(abi.encodePacked(_recipientAddre
ss))),
```

Description

zigzag : In the `withdrawDeposit` function, the sign hash only depend on `_recipientAddress` . Let’s assume the following scenario: Ailce deposit five `deposits` like 1,2,3,4,5 and set the `pubKey20` to Ailce. Now Ailce wants to approve `deposits 1` Bob and gives the _signature.

```
 require(
            _recipientAddressHash == ECDSA.toEthSignedMessageHash(keccak256(abi.encodePacked(_recipi
entAddress))),
            "HASHES DO NOT MATCH"
        );
        // check that the signer is the same as the one stored in the deposit
        address depositSigner = getSigner(_recipientAddressHash, _signature);
        require(depositSigner == _deposit.pubKey20, "WRONG SIGNATURE");
```

It is normal action if Bob wants to withdraw `deposits 1`.But Bob also can withdraw `deposits 2` because the check will be bypassed.

Recommendation

zigzag :

```
        require(
+            _recipientAddressHash == ECDSA.toEthSignedMessageHash(keccak256(abi.encodePacked(_recipientAddress, index))),
            "HASHES DO NOT MATCH"
        );
        // check that the signer is the same as the one stored in the deposit
        address depositSigner = getSigner(_recipientAddressHash, _signature);
        require(depositSigner == _deposit.pubKey20, "WRONG SIGNATURE");
```

Client Response

Fixed.True, thank you! Fixed as of https://github.com/peanutprotocol/peanut-contracts/blob/3da9f24376aa634ed85d83c076086f2732633e18/src/V4/PeanutV4.sol

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.