OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 37	2021-11-25	Zapmore, DoD4uFN	Audit Final

Audit Notes

Audit Date	2021-10-26 - 2021-11-24
Auditor/Auditors	DoD4uFN, Mechwar
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB566654752

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Contents

Project Information

Name	paprprintr
Description	“paprprintr at its core, is an algorithmic stablecoin driven by elastic expansion and burn. Inspired by Basis Cash and other predecessors, we have applied principles of those protocols and innovated our own new mechanisms that effectively maintains the $1 peg level.”
Website	https://paprprintr.finance/
Contact	Hash 🇭🇰#2935
Contact information	@Hash 🇭🇰#2935 on Discord
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	BSC

Audit of Paprprintr

The project team resolved all found vulnerabilities in the contracts. One open issue requires the implementation of a timelock (issue #3).

Obelisk was commissioned by Paprprintr on the 25th of October 2021 to conduct a comprehensive audit of Paprprintrs' Pool contracts. The following audit was conducted between the 26th of October 2021 and the 24th of November 2021. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited either by the project team or users.

The audit was conducted on not-yet-published contracts. This meant that the project team could easily resolve vulnerabilities that were found to be in the audited contracts.

The only issue not solved in the contracts is issue #3 which the project team intentionally left open as a backup plan in case of tokens sent to the wrong address. This needs to be timelocked with a 72-hour timelock to be seen as mitigated (which it currently isn't).

As we always recommend a timelock of 72 hours so that everyone invested in the project has ample time to react to changes, issue #19 is marked as partial-mitigated as there is a timelock implemented, but only for 24 hours.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if they deem that it's worth solving these issues.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the Paprprintr project.

Please read the full document for a complete understanding of the audit.

Summary Table

Finding	ID	Severity	Status
Reentrancy Vulnerability	0001	High Risk	Closed
Swapping Tokens Can Be Frontrun	0002	Medium Risk	Closed
Owner Can Disable The Rewards	0003	Medium Risk	Open
Addresses Are Not Valid	0004	Low Risk	Closed
Insufficient Pending Dividends For Swap To PRNTR	0005	Informational	Closed
Local Copies Of OpenZeppelin Contracts	0006	Informational	Closed
Use Of tx.origin	0007	Informational	Closed
Not Checking Current Balance Before Transferring Rewards	0008	Informational	Closed
Not Utilizing Safe Transfer	0009	Low Risk	Closed
Unused Contracts	0010	Informational	Closed
Outdated Compiler Version	0011	Informational	Closed
Variables Not Declared As Constants	0012	Informational	Closed
Withdraw Function Reverting	0013	Informational	Closed
Compilation Failure	0014	Informational	Closed
Unused Variables	0015	Informational	Closed
No Events Emitted For Changes To Protocol Values	0016	Informational	Closed
Redundant Function	0017	Low Risk	Closed
Unverified Token Contracts	0018	High Risk	Closed
No Timelock	0019	Low Risk	Partially Mitigated
Changes To Deployed Contract	0020	Informational	Closed

Findings

Manual Analysis

Reentrancy Vulnerability

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 1112-1126 PAPRUSDCpoolFarmingVariant.sol -> 1175-1190

```
1    function updateAccount(address account) private {
2        disburseTokens();
3        uint pendingDivs = getPendingDivs(account);
4        //we get some PRNTR at current rate
5        swapToPRNTR(pendingDivs);
6        uint256 prntrBal = IERC20(prntr).balanceOf(address(this));
7        if (pendingDivs > 0) {
8            require(Token(prntr).transfer(account, prntrBal), "Could not transfer tokens.");
9            totalEarnedTokens[account] = totalEarnedTokens[account].add(pendingDivs);
10           totalClaimedRewards = totalClaimedRewards.add(pendingDivs);
11           emit RewardsTransferred(account, pendingDivs);
12       }
13       lastClaimedTime[account] = now;
14       lastDivPoints[account] = totalDivPoints;
15   }
```

```
1    function withdraw(uint amountToWithdraw) public onlyOneBlock {
2        require(amountToWithdraw > 0, "Cannot withdraw 0 Tokens!");
3
4        require(depositedTokens[msg.sender] >= amountToWithdraw, "Invalid amount to withdraw");
5
6        updateAccount(msg.sender);
7
8        require(Token(trustedDepositTokenAddress).transfer(msg.sender, amountToWithdraw), "Could not
  transfer tokens.");
9
10       depositedTokens[msg.sender] = depositedTokens[msg.sender].sub(amountToWithdraw);
11       totalTokens = totalTokens.sub(amountToWithdraw);
12
13       if (holders.contains(msg.sender) && depositedTokens[msg.sender] == 0) {
14           holders.remove(msg.sender);
15       }
16   }
```

DESCRIPTION	Given a token with malicious code, the above methods could be vulnerable to reentrancy. The Checks Effects

	Interactions pattern has not been applied in the above methods.
RECOMMENDATION	It's recommended to update the *lastDivPoints* in *updateAccount*() and *depositedTokens* in *withdraw*() before the transferring of funds.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

Swapping Tokens Can Be Frontrun

FINDING ID	#0002
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 1109

```
1     IUniswapRouterETH(unirouter).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amount, 0,
  usdcToPRNTRRoute, address(this), now);
2
```

DESCRIPTION	Tokens are exchanged via a DEX router but do not provide a slippage limit. The rewards from the swap can be front-run.
RECOMMENDATION	Provide a slippage limit for the token as a parameter or use an oracle's time-weighted average price (TWAP) in order to prevent front running.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 9aa711dcd5c91f0f23baa44f1e12532d73b7e509

Owner Can Disable The Rewards

FINDING ID	#0003
SEVERITY	Medium Risk
STATUS	Open
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 1064 PAPRUSDCpoolFarmingVariant.sol -> 1076-1077 PAPRUSDCpoolFarmingVariant.sol -> 1289 - 1306

```
1     uint public adminCanClaimAfter = 1 minutes;
2
```

```
1         contractDeployTime = now;
2         adminClaimableTime = contractDeployTime.add(adminCanClaimAfter);
```

```
1     // function to allow owner to claim *other* modern ERC20 tokens sent to this contract
2     function transferAnyERC20Token(address _tokenAddr, address _to, uint _amount) public onlyOwner {
3         // require(_tokenAddr != trustedRewardTokenAddress && _tokenAddr !=
  trustedDepositTokenAddress, "Cannot send out reward tokens or staking tokens!");
4
5         require(_tokenAddr != trustedDepositTokenAddress, "Admin cannot transfer out deposit tokens
  from this vault!");
6         require((_tokenAddr != trustedRewardTokenAddress) || (now > adminClaimableTime), "Admin
  cannot Transfer out Reward Tokens Yet!");
7         require(Token(_tokenAddr).transfer(_to, _amount), "Could not transfer out tokens!");
8     }
9
10    // function to allow owner to claim *other* modern ERC20 tokens sent to this contract
11    function transferAnyOldERC20Token(address _tokenAddr, address _to, uint _amount) public
  onlyOwner {
12        // require(_tokenAddr != trustedRewardTokenAddress && _tokenAddr !=
  trustedDepositTokenAddress, "Cannot send out reward tokens or staking tokens!");
13
14        require(_tokenAddr != trustedDepositTokenAddress, "Admin cannot transfer out deposit tokens
  from this vault!");
15        require((_tokenAddr != trustedRewardTokenAddress) || (now > adminClaimableTime), "Admin
  cannot Transfer out Reward Tokens Yet!");
16
17        OldIERC20(_tokenAddr).transfer(_to, _amount);
18    }
```

DESCRIPTION	The functions *transferAnyERC20Token*() and *transferAnyOldERC20Token*() are needed in order to be able to withdraw any tokens sent to this address by mistake. This address should not allow the withdrawal of the deposit tokens or reward tokens. In this situation, the admin can withdraw the reward tokens 1 minute after the contract's deployment timestamp.
RECOMMENDATION	Do not allow the withdrawal of the reward tokens.
RESOLUTION	Team comment: We prefer to have safety measures applied in case. But the ownership will be transferred to a timelock. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Addresses Are Not Valid

FINDING ID	#0004
SEVERITY	Low Risk
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 1049 - 1054

```
1 // PAPR BUSD LP Tokens / Farming tokens
2 address public trustedDepositTokenAddress = 0xf7A7eCeB7BE17695B31aFe76D90182f0b5152995;
3 // Earned Tokens address : PRNTR
4 address public trustedRewardTokenAddress = 0xB7F7644f999D34fB58cE91b3dBc26B0Bf7081337;
5
6 address public prntr = 0xb40e2e77aADe08b19235Db2e6E709447842AF813;
```

DESCRIPTION	The following addresses are not valid according to *https://bscscan.com/*.
RECOMMENDATION	Add valid addresses to the contract.
RESOLUTION	The project team acknowledged the fact and it will be reviewed on-chain. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78 On-chain review: PAPRUSDCpoolFarmingVariant trustedDepositTokenAddress trustedRewardTokenAddress PRNTRUSDCpoolFarmingVariant trustedDepositTokenAddress trustedRewardTokenAddress

Redundant Function

FINDING ID	#0017
SEVERITY	Low Risk
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 285-302

```
1     // function to allow owner to claim *other* modern
  ERC20 tokens sent to this contract
2     function transferAnyERC20Token(address _tokenAddr,
  address _to, uint _amount) public onlyOwner {
3         // require(_tokenAddr != trustedRewardTokenAddress
  && _tokenAddr != trustedDepositTokenAddress, "Cannot send
  out reward tokens or staking tokens!");
4
5         require(_tokenAddr != trustedDepositTokenAddress,
  "Admin cannot transfer out deposit tokens from this
  vault!");
6         require((_tokenAddr != trustedRewardTokenAddress)
  || (block.timestamp > adminClaimableTime), "Admin cannot
  Transfer out Reward Tokens Yet!");
7         IERC20(_tokenAddr).safeTransfer(_to, _amount);
8     }
9
10    // function to allow owner to claim *other* modern
  ERC20 tokens sent to this contract
11    function transferAnyOldERC20Token(address _tokenAddr,
  address _to, uint _amount) public onlyOwner {
12        // require(_tokenAddr != trustedRewardTokenAddress
  && _tokenAddr != trustedDepositTokenAddress, "Cannot send
  out reward tokens or staking tokens!");
13
14        require(_tokenAddr != trustedDepositTokenAddress,
  "Admin cannot transfer out deposit tokens from this
  vault!");
15        require((_tokenAddr != trustedRewardTokenAddress)
  || (block.timestamp > adminClaimableTime), "Admin cannot
  Transfer out Reward Tokens Yet!");
16
17        OldIERC20(_tokenAddr).transfer(_to, _amount);
18    }
```

DESCRIPTION	The above functions are identical. Their difference is that the *OldIERC20* interface contains

	the same *transfer()* definition as *IERC20* interface. The *transferAnyOldERC20Token()* uses the non-safe version of *transfer()* function.
RECOMMENDATION	Remove the redundant function.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

Insufficient Pending Dividends For Swap To PRNTR

FINDING ID	#0005
SEVERITY	Informational
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 1112 - 1126

```
function updateAccount(address account) private {
    disburseTokens();
    uint pendingDivs = getPendingDivs(account);
    //we get some PRNTR at current rate
    swapToPRNTR(pendingDivs);
    uint256 prntrBal = IERC20(prntr).balanceOf(address(this));
    if (pendingDivs > 0) {
        require(Token(prntr).transfer(account, prntrBal), "Could not transfer tokens.");
        totalEarnedTokens[account] = totalEarnedTokens[account].add(pendingDivs);
        totalClaimedRewards = totalClaimedRewards.add(pendingDivs);
        emit RewardsTransferred(account, pendingDivs);
    }
    lastClaimedTime[account] = now;
    lastDivPoints[account] = totalDivPoints;
}
```

DESCRIPTION	The swap to PRNTR should occur when there are some pending dividends. If not, the swap will go through and will increase the gas cost.
RECOMMENDATION	Include *swapToPRNTR* and setting of *prntrBal* in the check for *pendingDivs* greater than zero.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Local Copies Of OpenZeppelin Contracts

FINDING ID	#0006
SEVERITY	Informational
STATUS	Closed
LOCATION	Everywhere in the file PAPRUSDCpoolFarmingVariant.sol.

DESCRIPTION	The contract includes local copies of OpenZeppelin contracts. The logic within these contracts is identical to OpenZeppelin 3.4.0: *SafeMath.sol* *IERC20.sol* *ERC20.sol* *SafeERC20.sol* *Address.sol*
RECOMMENDATION	Import OpenZeppelin instead of using local copies.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Use Of tx.origin

FINDING ID	#0007
SEVERITY	Informational
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 760 PAPRUSDCpoolFarmingVariant.sol -> 779

```
1 return _status[block.number][tx.origin];
```

```
1         _status[block.number][tx.origin] = true;
2
```

DESCRIPTION	Be aware that tx.origin might not stay useful in future updates, Vitalik: "Do NOT assume that tx.origin will continue to be usable or meaningful."
RECOMMENDATION	Be aware of the downsides of using tx.origin and watch out for changes concerning tx.origin.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Not Checking Current Balance Before Transferring Rewards

FINDING ID	#0008
SEVERITY	Informational
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 1112-1126

```
1     function updateAccount(address account) private {
2         ...
3         uint256 prntrBal =
  IERC20(prntr).balanceOf(address(this));
4         if (pendingDivs > 0) {
5             require(Token(prntr).transfer(account,
  prntrBal), "Could not transfer tokens.");
6             ...
7         }
8         ...
9     }
```

DESCRIPTION	The method above transfers the user's rewards. But the *prntrBal* is the balance of the contract, after the swapping of pending rewards to prntr.
RECOMMENDATION	Check for any available prntr balance before swapping and subtract it from the *prntrBal*.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

Withdraw Function Reverting

FINDING ID	#0013
SEVERITY	Informational
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 109-126

```
    function updateAccount(address account) private {
        disburseTokens();
        uint pendingDivs = getPendingDivs(account);
        //we get some PRNTR at current rate
        require(pendingDivs > 0, 'Thats what she said');
        swapToPRNTR(pendingDivs);
        uint256 prntrBal =
IERC20(prntr).balanceOf(address(this));
        require(prntrBal > 0, 'Thats what he said');
        if (pendingDivs > 0) {
            lastDivPoints[account] = totalDivPoints;
            IERC20(prntr).safeTransfer(account, prntrBal);
            totalEarnedTokens[account] =
totalEarnedTokens[account].add(pendingDivs);
            totalClaimedRewards =
totalClaimedRewards.add(pendingDivs);
            emit RewardsTransferred(account, pendingDivs);
        }
        lastClaimedTime[account] = block.timestamp;

    }
```

DESCRIPTION	The *withdraw()* can revert because of the new require statements in *updateAccount()* function. By adding these statements, the user is unable to call the *withdraw()* function if there were no dividends accumulated. The user would need to use *emergencyWithdraw()* function to retrieve their funds. Additionally, proper and descriptive messaging in the *require* statements is encouraged.

RECOMMENDATION	Decide if this functionality is intended or not.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

Static Analysis

Not Utilizing Safe Transfer

FINDING ID	#0009
SEVERITY	Low Risk
STATUS	Closed
LOCATION	• PAPRUSDCpoolFarmingVariant.sol -> 1104: require(Token(trustedRewardTokenAddress).transferFrom(msg.sender, address(this), amount), "Cannot add balance!"); • PAPRUSDCpoolFarmingVariant.sol -> 1119: require(Token(prntr).transfer(account, prntrBal), "Could not transfer tokens."); • PAPRUSDCpoolFarmingVariant.sol -> 1164: require(Token(trustedDepositTokenAddress).transferFrom(msg.sender, address(this), amountToDeposit), "Insufficient Token Allowance"); • PAPRUSDCpoolFarmingVariant.sol -> 1182: require(Token(trustedDepositTokenAddress).transfer(msg.sender, amountToWithdraw), "Could not transfer tokens."); • PAPRUSDCpoolFarmingVariant.sol -> 1203: require(Token(trustedDepositTokenAddress).transfer(msg.sender, amountToWithdraw), "Could not transfer tokens."); • PAPRUSDCpoolFarmingVariant.sol -> 1295: require(Token(_tokenAddr).transfer(_to, _amount), "Could not transfer out tokens!");

DESCRIPTION	SafeERC20 is included in the contract, but *transfer* and *transferFrom* functions are used instead of *safeTransfer* and *safeTransferFrom*.
RECOMMENDATION	Make use of Openzeppelin's safe transfer functions.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Unused Contracts

FINDING ID	#0010
SEVERITY	Informational
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 7-48

```
1 contract Operator is Context, Ownable {
2     // ...
3 }
```

DESCRIPTION	The *Operator* contract is not used within the *PAPRUSDCpoolFarmingVariant* contract.
RECOMMENDATION	Remove the *Operator* contract.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Outdated Compiler Version

FINDING ID	#0011
SEVERITY	Informational
STATUS	Closed
LOCATION	Everywhere in the file PAPRUSDCpoolFarmingVariant.sol.

DESCRIPTION	Out of date compiler, versions may have vulnerabilities that have been fixed in later versions.
RECOMMENDATION	Using up-to-date compiler versions when possible is best practice. It is recommended to upgrade to Solidity 8.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit cf2e3ccb8de1a6d55c7152dd6701e6f852442c78

Variables Not Declared As Constants

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	• PAPRUSDCpoolFarmingVariant.sol -> 1050: address public trustedDepositTokenAddress = 0xf7A7eCeB7BE17695B31aFe76D90182f0b5152995; • PAPRUSDCpoolFarmingVariant.sol -> 1052: address public trustedRewardTokenAddress = 0xB7F7644f999D34fB58cE91b3dBc26B0Bf7081337; • PAPRUSDCpoolFarmingVariant.sol -> 1054: address public prntr = 0xb40e2e77aADe08b19235Db2e6E709447842AF813; • PAPRUSDCpoolFarmingVariant.sol -> 1055: address public unirouter = 0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506; • PAPRUSDCpoolFarmingVariant.sol -> 1058: uint public disburseAmount = 50000e18; • PAPRUSDCpoolFarmingVariant.sol -> 1060: uint public disburseDuration = 1826 days; • PAPRUSDCpoolFarmingVariant.sol -> 1064: uint public adminCanClaimAfter = 1 minutes; • PAPRUSDCpoolFarmingVariant.sol -> 1068: uint public disbursePercentX100 = 100e2; • PAPRUSDCpoolFarmingVariant.sol -> 1101: uint internal pointMultiplier = 1e18;

DESCRIPTION	The aforementioned contract variables are not assigned after their initialization.
RECOMMENDATION	Define these variables as constants.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

Compilation Failure

FINDING ID	#0014
SEVERITY	Informational
STATUS	Closed
LOCATION	• PAPRUSDCpoolFarmingVariant.sol -> 9: import '@openzeppelin/contracts/token/ERC20//utils/SafeERC20.sol'; • PAPRUSDCpoolFarmingVariant.sol -> 13: import "./interfaces/IUniswapRouterETH.sol";

DESCRIPTION	1. There should not be double backslashes in the import path for *SafeERC20*. 2. *IUniswapRouterETH.sol* interface is missing from the *Paprprintr-contracts* repository. 3. *IUniswapV2Pair.sol* is not imported.
RECOMMENDATION	1. Remove the double backslashes in the import path for *SafeERC20*. 2. Include the *IUniswapRouterETH.sol* interface file in the *Paprprintr-contracts* repository. 3. Add an `import` statement for *IUniswapV2Pair.sol*.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

Unused Variables

FINDING ID	#0015
SEVERITY	Informational
STATUS	Closed
LOCATION	• PAPRUSDCpoolFarmingVariant.sol -> 96: uint256 amount = _amount; • PAPRUSDCpoolFarmingVariant.sol -> 97: (uint256 reserve0, uint256 reserve1, uint32 blockTimestampLast) = IUniswapV2Pair(trustedDepositTokenAddress).getReserves(); • PAPRUSDCpoolFarmingVariant.sol -> 104: uint256 amount = _amount; • PAPRUSDCpoolFarmingVariant.sol -> 105: (uint256 reserve0, uint256 reserve1, uint32 blockTimestampLast) = IUniswapV2Pair(trustedDepositTokenAddress).getReserves();

DESCRIPTION	Contract values are set but never used: • amount • blockTimestampLast
RECOMMENDATION	Remove the variables or incorporate them into the contract functionality.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

No Events Emitted For Changes To Protocol Values

FINDING ID	#0016
SEVERITY	Informational
STATUS	Closed
LOCATION	PAPRUSDCpoolFarmingVariant.sol -> 85-88

```
1     function setSlippage(uint256 _slippage) public
  onlyOwner {
2         require(slippage < 99, 'Too high ser');
3         slippage = _slippage;
4     }
```

DESCRIPTION	Functions that change important variables should emit events such that users can more easily monitor the change.
RECOMMENDATION	Emit events from these functions.
RESOLUTION	The project team has implemented the recommended fix. Reviewed in commit 0d71dc09c5e8be57a239915c59523b83d5500f91

On-Chain Analysis

Unverified Token Contracts

FINDING ID	#0018
SEVERITY	High Risk
STATUS	Closed
LOCATION	trustedDepositTokenAddress - PanamaPapr LPs prntrLPAddress - PanamaPapr LPs

DESCRIPTION	The aforementioned tokens, that are used in the protocol have unverified contracts on-chain.
RECOMMENDATION	Verify these contracts.
RESOLUTION	The project team has verified the aforementioned LP contracts.

No Timelock

FINDING ID	#0019
SEVERITY	Low Risk
STATUS	Partially Mitigated
LOCATION	PRNTRUSDCpoolFarmingVariant PAPRUSDCpoolFarmingVariant

DESCRIPTION	The following contracts have not had their ownership transferred to a timelock contract yet: - PRNTRUSDCpoolFarmingVariant - PAPRUSDCpoolFarmingVariant
RECOMMENDATION	Transfer ownership to the timelock contract.
RESOLUTION	The project team has transferred ownership to a timelock contract with a delay of 24 hours.

Changes To Deployed Contract

FINDING ID	#0019
SEVERITY	Informational
STATUS	Closed
LOCATION	PRNTRUSDCpoolFarmingVariant PAPRUSDCpoolFarmingVariant

DESCRIPTION	Minor adjustments to the contracts were made prior to deployment. PRNTRUSDCpoolFarmingVariant: - Renamed contract to PRNTRUSDCpoolFarmingVariant - *trustedDepositTokenAddress* to 0x383df401Da236724AC16bB6765d8310c6Ad6e923 - *trustedRewardTokenAddress* to 0x980a5AfEf3D17aD98635F6C5aebCBAedEd3c3430 - *prntr* to 0xD047764d8915E0C482A8Dd5804830BB8ff5a5285 - *unirouter* to 0x2DF7AEb4C3F15D9c5Db595e8118ff6dB4A154d6D - *disburseAmount* to 40000e18 - *disburseDuration* to 365 days - PAPRUSDCpoolFarmingVariant.sol 96: *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve1, reserve0);* to *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve0, reserve1);* - PAPRUSDCpoolFarmingVariant.sol 102: *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve1, reserve0);* to *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve0, reserve1);* PAPRUSDCpoolFarmingVariant: - *trustedDepositTokenAddress* to 0x9e7f2318c9060FAeafd9106ccB6530941C3BCF4a - *trustedRewardTokenAddress* to 0x980a5AfEf3D17aD98635F6C5aebCBAedEd3c3430 - *prntrLPAddress* to 0x383df401Da236724AC16bB6765d8310c6Ad6e923 - *prntr* to

	0xD047764d8915E0C482A8Dd5804830BB8ff5a5285 - *unirouter* to 0x2DF7AEb4C3F15D9c5Db595e8118ff6dB4A154d6D - *disburseAmount* to 60000e18 - *disburseDuration* to 365 days - PAPRUSDCpoolFarmingVariant.sol 95: *(uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(trustedDepositTokenAddress).getReserves();* to *(uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(prntrLPAddress).getReserves();* - PAPRUSDCpoolFarmingVariant.sol 96: *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve1, reserve0);* to *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve0, reserve1);* - PAPRUSDCpoolFarmingVariant.sol 101: *(uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(trustedDepositTokenAddress).getReserves();* to *(uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(prntrLPAddress).getReserves();* - PAPRUSDCpoolFarmingVariant.sol 102: *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve1, reserve0);* to *(uint256 amountOut) = IUniswapRouterETH(unirouter).getAmountOut(_amount, reserve0, reserve1);*
RECOMMENDATION	No changes are necessary.
RESOLUTION	N/A

Appendix A - Reviewed Documents

Document	Address
Paprprintr-contracts/blob/main/LPPools/PAPRUSDCpoolFarmingVariant.sol	N/A
PRNTRUSDCpoolFarmingVariant.sol	0x910b08b8b3881af0adac19ab0bd87a3207c44516
PAPRUSDCpoolFarmingVariant.sol	0x7e4ee21411ae669c944ec8c756f6d60e0eb72642
Timelock	0x7d4d75CE96D579bCEeF751ED4FBBc215AF174157

Revisions

Revision 1: 36c4a693cc56e3ec2b6fea5084348c1fbff373ab
Revision 2: cf2e3ccb8de1a6d55c7152dd6701e6f852442c78
Revision 3: 0d71dc09c5e8be57a239915c59523b83d5500f91

Imported Contracts

OpenZeppelin: 3.1.2

Externally Owned Accounts

Timelock owner: 0x68e993f21633Eb95b42564bD28Ca650078ee96A9

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
Low Risk	A vulnerability that can cause the loss of protocol functionality.
Informational	Non-security issues such as functionality, style, and convention.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved by other methods such as revoking contract ownership. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were updated to fix the issue in some parts of the code.
Partially Mitigated	Fixed by project-specific methods which cannot be verified on-chain. Examples include compounding at a given frequency.
Open	The finding was not addressed.

Appendix D - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

 ObeliskOrg

 ObeliskOrg

OBELISK

Part of Tibereum Group