PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Zappy Finance

04 March 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Zappy Finance on the Telos EVM network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Zappy Finance
URL	https://zappy.finance/
Platform	Telos EVM
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
MasterChef	0x3D2c6bCED5f50f5412234b87fF0B445aBA4d10e9	MATCH
ZappyToken	0x9A271E3748F59222f5581BaE2540dAa5806b3F77	MATCH
UniswapV2ERC20	Dependency	MATCH
UniswapV2Factory	0x4be5Bf2233a0fd2c7D1472487310503Ec8E857be	MATCH
UniswapV2Pair	Dependency	MATCH
UniswapV2Router02	0xB9239AF0697C8efb42cBA3568424b06753c6da71	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	1	1	-	-
Medium	1	-	1	-
Low	3	1	-	2
Informational	22	8	-	14
Total	**27**	**10**	**1**	**16**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 MasterChef

ID	Severity	Summary	Status
01	HIGH	Deposits do not support fee-on-transfer tokens	RESOLVED
02	MEDIUM	Deposits and withdrawals can revert when the hard-cap is (almost) reached	PARTIAL
03	LOW	Setting to the zero address will break deposit and withdraw functionality	ACKNOWLEDGED
04	LOW	pendingZAP and updatePool will revert if totalAllocPoint is zero	ACKNOWLEDGED
05	INFO	Unnecessary dependency: SafeMath	ACKNOWLEDGED
06	INFO	zap can be made immutable	ACKNOWLEDGED
07	INFO	Lack of events for setZapPerSecond, add and dev	ACKNOWLEDGED
08	INFO	Pool uses the contract balance to figure out the total deposits	ACKNOWLEDGED
09	INFO	Validation not added for constructor variables	ACKNOWLEDGED
10	INFO	Missing token validation	ACKNOWLEDGED
11	INFO	Typographical errors	ACKNOWLEDGED
12	INFO	massUpdatePools() can run out of gas	ACKNOWLEDGED
13	INFO	pendingZAP does not account for the maxSupply	ACKNOWLEDGED
14	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
15	INFO	deposit, withdraw, emergencyWithdraw and dev can be made external	ACKNOWLEDGED
16	INFO	Gas optimization: Implementation of checkForDuplicate can unnecessarily run out of gas	ACKNOWLEDGED

1.3.2 ZappyToken

ID	Severity	Summary	Status
17	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
18	INFO	delegateBySig can be frontrun and cause denial of service	RESOLVED
19	INFO	mint can be made external	ACKNOWLEDGED
20	INFO	Typographical error	ACKNOWLEDGED

1.3.3 UniswapV2ERC20

ID	Severity	Summary	Status
21	INFO	The Approval event is not emitted if allowance is changed in transferFrom as suggested in the ERC-20 Token Standard (also present in Uniswap)	RESOLVED
22	INFO	permit can be front-run to prevent someone from calling removeLiquidityWithPermit (also present in Uniswap)	RESOLVED

1.3.4 UniswapV2Factory

ID	Severity	Summary	Status
23	INFO	Lack of event emissions for sensitive changes	RESOLVED

1.3.5 UniswapV2Pair

No issues found.

1.3.6 UniswapV2Router02

ID	Severity	Summary	Status
24	INFO	Phishing is possible by a malicious frontend by adjusting routes, tokens or from parameters (also present in Uniswap)	RESOLVED
25	INFO	Constructor inputs lack a zero address input verification	RESOLVED
26	INFO	Function addLiquidity does still not fully support reflective tokens upon transfer	RESOLVED
27	INFO	Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (this issue is present in most frontends)	RESOLVED

2 Findings (Zappy Farm)

2.1 MasterChef

The MasterChef is a staking contract which allows the user to deposit various LP tokens and receive Zappy Tokens as a reward. It mints an additional 10% of the tokens to the dev address. Additionally, this contract does not have any deposit fees.

2.1.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `setZapPerSecond`
- `add`
- `set`
- `dev`
- `transferOwnership`
- `renounceOwnership`

2.1.2 Issues & Recommendations

Issue #01	Deposits do not support fee-on-transfer tokens
Severity	HIGH SEVERITY
Description	The `deposit` function of the MasterChef presently lacks support for fee-on-transfer tokens. This means that during a deposit of such tokens, the Masterchef will actually receive less tokens than it thinks it has received. This not only leads to the fact that not everyone will be able to withdraw such tokens, it also can be exploited where a malicious user can drain the whole pool, resulting in absurd reward minting. This is because the reward distribution mechanism gives an emission multiplier if `lpSupply` in the `updatePool` function is smaller than the `user.amount` value.

Recommendation	Consider using a before-after pattern to account for the transfer tax.
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.safeTransferFrom(msg.sender, address(this), _amount);`
	`_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	
	Do note that such a pattern must always be accompanied with a `nonReentrant` modifier to avoid exploitation.

Resolution	RESOLVED The client has indicated that their Masterchef is solely supposed to work with LP tokens which do not have a transfer tax. The client should however remain extremely vigilant whenever adding a new token.

Issue #02	Deposits and withdrawals can revert when the hard-cap is (almost) reached
Severity	MEDIUM SEVERITY
Description	Currently, the `updatePool` function will fail when the `maxSupply` is reached as the `mint` function in the token will revert.
Recommendation	Consider implementing the following code to account for the decreased `zapReward`:

```
uint256 zapReward =
multiplier.mul(zapPerSecond).mul(pool.allocPoint).div(totalAllocPoint);
uint256 remaining = zap.getMaxTotalSupply().sub(zap.totalSupply());
uint256 devReward = zapReward.div(10));

if (remaining < zapReward.add(devReward)) {
     zapReward = remaining;
     devReward = 0;
}
if(zapReward > 0) {
    zap.mint(address(this), zapReward);
}
if (devReward > 0) {
    zap.mint(devaddr, devReward);
}
```

	The client should carefully test this behavior by fully going to this scenario in their test deployment.
Resolution	PARTIALLY RESOLVED The client has indicated that they will immediately disable emissions when the `maxSupply` is reached. This will minimize the impact of this issue. In case the client fails to do so, users can still withdraw using the `emergencyWithdraw` function on the contract.

Issue #03	**Setting `devaddr` to the zero address will break `deposit` and `withdraw` functionality**
Severity	LOW SEVERITY
Description	Within most token contracts, minting or transferring tokens to the zero address will revert the transaction. This could cause deposits to revert when the deposit fee is transferred to a zero `devaddr`.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like the following: `require(_devaddr != address(0), "!nonzero");` to the relevant configuration function.
Resolution	ACKNOWLEDGED The client has indicated that they will keep this in mind and avoid setting the `devaddr` to zero.

Issue #04	**`pendingZAP` and `updatePool` will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingZAP` and `updatePool` functions, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards since the `lastRewardTime` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.timestamp` and `lpSupply`, like so: `if (block.timestamp > pool.lastRewardTime && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	ACKNOWLEDGED

Issue #05	Unnecessary dependency: SafeMath
Severity	INFORMATIONAL
Location	Line 1536 `Using SafeMath for uint256;`
Description	Usage of SafeMath is considered best practice to secure the code against overflows or underflows. However, this is not necessary when Solidity version >= 0.8.0 is used.
Recommendation	Consider removing the SafeMath dependency and adjust the code accordingly or simply acknowledge this issue.
Resolution	ACKNOWLEDGED

Issue #06	zap can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	ACKNOWLEDGED

Issue #07	Lack of events for `setZapPerSecond`, `add` and `dev`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

Issue #08	Pool uses the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs and staking contracts, the total number of tokens in the contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits.
Resolution	ACKNOWLEDGED

Issue #09	Validation not added for constructor variables
Severity	INFORMATIONAL
Description	Within the constructor, there are important variables defined which are not validated at all. If, for example, the `devaddr` is set to the zero address, this will break deposits and withdrawals.
Recommendation	Consider adding all necessary validations within the constructor.
Resolution	ACKNOWLEDGED

Issue #10	Missing token validation
Severity	INFORMATIONAL
Description	Within the `add` function, governance can add specific pools for different `lpToken`; however, the validation for this `lpToken` is missing, which can lead to a broken pool if a `lpToken` is added that does not exist at all.
Recommendation	Consider adding `_lpToken.balanceOf(address(this))` for token validation.
Resolution	ACKNOWLEDGED

Issue #11	Typographical errors
Severity	INFORMATIONAL

Description	The contract contains a number of typographical errors which are enumerated below in a single issue in an effort to keep the report size reasonable.
	
	Line 1673
	`require(_allocPoint <= MaxAllocPoint, "add: too many alloc points!!");`
	
	The contract still mentions `add`.
	
	Line 1785 (example)
	`pool.lpToken.safeTransfer(address(msg.sender), oldUserAmount);`
	
	Casting `msg.sender` to `address` is unnecessary. This error can be found in various sections.

Recommendation	Consider fixing the above typographical errors.
Resolution	ACKNOWLEDGED

Issue #12	`massUpdatePools()` can run out of gas
Severity	INFORMATIONAL
Description	Within the `massUpdatePools()` function, `updatePools()` is called for each existing pool. However, if there are too many pools, this function may eventually run out of gas.
Recommendation	Consider adding a boolean variable `withUpdate` for the `set`, `add` and `setZapPerSecond` function which will call `massUpdatePools()` or not.
Resolution	ACKNOWLEDGED

Issue #13	`pendingZAP` does not account for the `maxSupply`
Severity	INFORMATIONAL
Description	During the pendingZAP function, there is currently no accounting for the case if the maxSupply is reached. This can lead to misleading information on the frontend.
Recommendation	Consider adding a logic that accounts for the maxSupply. Inspiration can be taken from a previous recommendation with regards to updatePool.
Resolution	ACKNOWLEDGED

Issue #14	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the `pendingZap` function, `accZAPPerShare` is based upon the `lpSupply` variable. `pool.accZAPPerShare = pool.accZAPPerShare.add(zapReward.mul(1e12).div(lpSupply);` However, if this `lpSupply` becomes a severely large value this will cause precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to 1e18 across the entire contract. It should be noted that even a precision of 1e18 can be imprecise in some edge cases. In case the client thinks it is probable that such tokens will be added, we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	ACKNOWLEDGED The client has indicated that they have no plans to add such tokens.

Issue #15	**`deposit`, `withdraw`, `emergencyWithdraw` and `dev` can be made external**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the `external` keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above functions as `external`.
Resolution	ACKNOWLEDGED

Issue #16	**Gas optimization: Implementation of `checkForDuplicate` can unnecessarily run out of gas**
Severity	INFORMATIONAL
Description	The `checkForDuplicate` function iterates over all pools to check if the token already exists. If a newly added token already exists in a previous pool, this check causes the add function to revert. Although we understand the value of this function, the implementation can be significantly improved by using a mapping.
Recommendation	Consider using `mapping(address =>bool) public tokenPoolExists;` to significantly improve the gas behavior of the `checkForDuplicate` function.
Resolution	ACKNOWLEDGED

2.2 ZappyToken

The Zappy Token is the governance token of the Zappy Farm. It is distributed amongst users staking in the MasterChef and has a maximum supply of 27,000,000 tokens. The token allows for Zappy tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the Zappy team. Users should therefore carefully inspect that ownership of this contract has been transferred to the MasterChef.

Finally, it contains similar voting logic as in tokens like Compound and Yam, however, the client has indicated that they have no immediate plans to use it.

2.2.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `mint`
- `transferOwnership`
- `renounceOwnership`

2.2.2 Token Overview

Address	0x9A271E3748F59222f5581BaE2540dAa5806b3F77
Token Supply	27,000,000
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	None

2.2.3 Issues & Recommendations

Issue #17	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	LOW SEVERITY
Description	The `mint` function allows the owner (contract deployer) to mint tokens before ownership is transferred to the Masterchef. This could be used to mint a large amount of tokens and potentially dump them on user generated liquidity when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the blockchain's explorer.
Recommendation	Consider being forthright if this `mint` function is to be used by letting your community know how much was minted, where the tokens are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED The token has been fully deployed and the ownership has already been transferred to the MasterChef. The client has also indicated that users who want to understand how the minting privileges were used before ownership was transferred can find this out either through on-chain transactions or by reading their documentation at https://docs.zappy.finance/token-supply-breakdown.

Issue #18	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if `delegateBySig` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `delegateBySig` transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example. This same issue also presents itself within the `permit` function from ERC20Permit.
Recommendation	Consider this issue if there are ever complaints by users that their transactions are failing. It could be the case that someone is using this vector against them. Additionally, derivative contracts that are using `delegateBySig` and `permit` should be adjusted for the fact that these function executions might have been front-ran.
Resolution	RESOLVED The client has indicated that they understand this behavior and will account for it if they ever develop contracts that use `delegateBySig` and `permit`.

Issue #19	`mint` can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the function as external.
Resolution	ACKNOWLEDGED

Issue #20	Typographical error
Severity	INFORMATIONAL
Location	Line 835 `require(totalSupply() + _amount <= _maxTotalSupply, "ERC20: minting more then MaxTotalSupply");`
Description	The contract contains the above typographical error. This error message should use *than* instead of *then*.
Recommendation	Consider fixing the typographical error.
Resolution	ACKNOWLEDGED

3 Findings (Zappy Core)

The Zappy Core section of this audit refers to the AMM deployment contracts, which are a straightforward fork of the trusted and respected Uniswap V2 codebase.

Minimal changes have been made: specifically the swap fee has been reduced to 0.2%. According to their documentation, 75% of this fee (0.15% of the total swap amount) is returned to liquidity stakers, while 25% of this fee (0.05% of the total swap amount) will be used to buy back ZAP and then distributed to ZAP stakers.

All issues mentioned below were not introduced by Zappy.

3.1 UniswapV2ERC20

The UniswapV2Erc20 contract is an implementation of the ERC-20 Token Standard for denominating pool tokens. It is a fork of Uniswap's UniswapV2Erc20 contract.

3.1.1 Issues & Recommendations

Issue #21	**The Approval event is not emitted if allowance is changed in `transferFrom` as suggested in the ERC-20 Token Standard (also present in Uniswap)**
Severity	INFORMATIONAL
Description	The ERC-20 standard specifies that an Approval event should be emitted when the allowance of a user changes. However, within the ERC20 implementation of both Uniswap and Zappy Finance, this is not done. You can read more about this improvement in Pull Request #65 of uniswap-core.
Recommendation	Consider adding `emit Approval(from, msg.sender, remaining)` in `transferFrom` when allowance is modified.
Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.

Issue #22	**`permit` can be front-run to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)**
Severity	INFORMATIONAL
Description	Currently if `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up `permit` transactions in the mempool and execute them before a contract can. The implications of this issue is that a bad actor could prevent a user from removing liquidity with a permit through the router. It is a denial-of-service attack which is present in all AMMs but which we have yet to witness being used since there is no profit from it.
Recommendation	Consider this issue if there are ever complaints by users that their `removeLiquidityWithPermit` transactions are failing. It could be the case that someone is using this vector against them. We do not recommend changing this behavior since it would cause a lot of extra work modifying the frontend to account for new permit behavior. This issue is also present in Uniswap after all.
Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.

3.2 UniswapV2Factory

The UniswapV2Factory is in charge of managing all the existing asset pairs and allows users to create new pairs if matched tokens for supplying liquidity have no existing contract pair. The UniswapV2Factory deploys UniswapV2Pair contracts.

3.2.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `setFeeTo`
- `setFeeToSetter`

3.2.2 Issues & Recommendations

Issue #23	Lack of event emissions for sensitive changes
Severity	INFORMATIONAL
Description	There is a lack of event emission for the following functions that result in sensitive changes and the behavior of the contract: - `setFeeTo` - `setFeeToSetter`
Recommendation	Emit events with the changed values.
Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.

3.3 UniswapV2Pair

The UniswapV2Pair is involved in storing the asset pairs in a contract for use by the router to add/remove liquidity in equally valued proportions and swap assets. It is a fork from the Uniswap version of this contract. The swapFee is set to 0.2% instead of 0.3%. 1/4 of all swapping fees goes to governance.

3.3.1 Issues & Recommendations

No issues found.

3.4 UniswapV2Router02

The UniswapV2Router02 is the interface for all DEX interactions such as adding liquidity, swapping and removing liquidity. It is a fork of the Uniswap version of this contract.

3.4.1 Issues & Recommendations

Issue #24	**Phishing is possible by a malicious frontend by adjusting routes, tokens or from parameters (also present in Uniswap)**
Severity	INFORMATIONAL
Description	A malicious (e.g. compromised) frontend can easily mislead users in approving malicious transactions, even if the router matches the address described in this report. An obvious example of how this can be done is by changing the to parameter which indicates to whom tokens or liquidity has to be sent. Other ways to phish could include using malicious routes or tokens.
Recommendation	Consider carefully protecting the frontend and ideally having an unchangeable IPFS fallback implementation for it. Users should also verify that they are on the correct website when doing a swap.
Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.

Issue #25	Constructor inputs lack a zero address input verification
Severity	INFORMATIONAL
Description	Inadvertently inputting zero address in the constructor will render the contract unusable and will require redeployment.

Recommendation	Consider implementing:
	`require( _factory != address(0), "Cannot be zero address" );`
	and a
	`require( _WETH != address(0), "Cannot be zero address" );`
	statement.

Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.

Issue #26	**Function `addLiquidity` does still not fully support reflective tokens upon transfer**
Severity	INFORMATIONAL
Description	The current `addLiquidity` function always assumes tokens are not reflective, thus resulting in waste should users be opting to transact with reflective tokens. This issue is also present in Uniswap itself.
Recommendation	The client may opt to include the following function that could help calculate the quotes beforehand for fee-on-transfer tokens. `addLiquiditySupportingFeeOnTransfer` first sends the `feeToken` to the pair, then it scales down the amount of `tokenB` to send to the pair based on how much arrived. You can read more about this issue here: https://github.com/Uniswap/v2-periphery/issues/106
Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.

Issue #27	**Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (this issue is present in most frontends)**
Severity	INFORMATIONAL
Description	A malicious DoS attack we have witnessed in practice is when a project wants to go live through a presale, people can instantiate the pair while there are no tokens yet. The malicious party will then sent some of the counterparty token to this pair so it has a partial balance (eg. 0.1 TELOS and 0 tokens). When `sync()` is then called, the pairs' reserves are updated to account for this balance. Due to a division by zero exception, many frontends cannot properly account for this state and will go through a blank page, preventing the original project from adding liquidity through the frontend.
Recommendation	Consider checking whether this is present in the frontend and adding a division by zero handler.
Resolution	RESOLVED The client has stated that they would like to remain compliant with Uniswap v2.


PALADIN
BLOCKCHAIN SECURITY