Spartan Protocol

Security Assessment

October 5th, 2020

For :
Spartan Protocol

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Camden Smallwood @ CertiK
camden.smallwood@certik.org

OVERVIEW

Project Summary

Project Name	Spartan Protocol
Description	A protocol for incentivized liquidity pools and synthetic assets on the Binance smart chain.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 74990ce6bad3ac5eb3dd2eb404dc9b27f66e6578 2. 6538ef20c6864e74800f811c30eb16133da631fb

Audit Summary

Delivery Date	Oct. 5, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Sep. 18, 2020 - Sep. 24 2020

Vulnerability Summary

Total Issues	20
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	19

EXECUTIVE SUMMARY

We conducted a balanced academic and system-based analysis of the codebase and were unable to identify any potential attack vectors that can be exploited. The development team behind the Spartan Protocol project has applied the latest security standards to their codebase, enforcing certain security principles such as funds never remaining at rest that greatly enhance the security of the contracts.

We relayed our optimization findings to the team and after collaborative discussion we came to the conclusion that the optimal choice is to not apply them as they are not substantial optimizations and would require changes across the whole codebase.

FINDINGS

ID	Title	Type	Severity
SPA-01	Unused Function Call Results	Implementation	Informational
SPA-02	Potentially Redundant `bool` Variable Return	Implementation	Informational
SPA-03	Usage of `tx.origin`	Implementation	Minor
SPA-04	Variable State Mutability	Optimization	Informational
SPA-05	Variable Visibility Specifier	Syntax	Informational
SPA-06	Variable State Mutability	Optimization	Informational
SPA-07	BEP-20 `approve` Race Condition	Implementation	Informational
SPA-08	Contract Bytecode Optimization	Optimization	Informational
SPA-09	Redundant Casting	Optimization	Informational
SPA-10	Migration Guard	Implementation	Informational
SPA-11	Function Re-use	Optimization	Informational
SPA-12	Function Re-use	Optimization	Informational
SPA-13	Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational
SPA-14	Unused Fallback Function	Optimization	Informational
SPA-15	Variable State Mutability	Optimization	Informational
SPA-16	External Call Optimization	Optimization	Informational
SPA-17	Redundant Casting	Optimization	Informational
SPA-18	Incorrect Pool Age Measurement	Implementation	Informational
SPA-19	Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational
SPA-20	Supplementary Documentation Request	Documentation	Informational

SPA-01: Unused Function Call Results

Type	Severity	Location
Implementation	Informational	Router L237, L238, L261, L485, L492, L573, L577, L590

Description:

The linked code segments are performing BEP-20 compliant `transfer` operations on arbitrary tokens. The standard denotes that a return variable should be set that dictates whether the `transfer` operation succeeded.

Recommendation:

Certain tokens that are otherwise BEP-20 compliant fail to satisfy this requirement and may fail if a strict check is imposed and as such, we advise to utilize a fork of the OpenZeppelin `SafeERC20` library that is minimally adapted to work with BEP-20 tokens.

The library essentially renders the return variable optional, however should it exist it should evaluate to `true`.

SPA-02: Potentially Redundant `bool` Variable Return

Type	Severity	Location
Implementation	Informational	Router L197-L200

Description:

The linked code segment conducts a `transfer` operation, however it utilizes a non-standard `transferTo` function name and returns a `bool` variable that is always `true`.

Recommendation:

As the return variable remains unused within the contract, we advise that it is omitted unless the contract is meant to implement a standard that is undocumented within the code or the project's `README.md` file.

SPA-03: Usage of `tx.origin`

Type	Severity	Location
Implementation	Minor	Pool L198

Description:

The linked code segment conducts a token transfer between the original transaction invocator and the input `recipient` variable.

Recommendation:

While this design choice was made so that the removal of liquidity is possible instantenously, this is an invalid design paradigm. As `tx.origin` is utilized, any contract on the Binance chain is capable of redirecting its calls to the `Pool` contract and consequently draining the funds of the original invocator.

We reached out to Binance and were informed that smart contracts on the Binance chain do not need an audit before being deployed on the mainnet, rendering this attack vector as `Medium` in severity as a tangible attack vector exists.

Smart contracts on the Binance chain have the luxury of being fully audited before being deployed on the main-net, meaning this attack vector is `Minor`, however a smart contract can purposefully avoid "detection" from an audit by allowing an arbitrary function call execution i.e. governance systems. As such, this is a tangible attack vector.

We advise that either `msg.sender` is utilized and the workflows on `Router.sol` are revamped, or that a conditional is imposed that utilizes `tx.origin` only if the invoker of the function is the `Router` contract and otherwise utilizes `msg.sender`.

SPA-04: Variable State Mutability

Type	Severity	Location
Optimization	Informational	Pool L114, L115, L122, L123

Description:

The `BASE`, `TOKEN`, `decimals` and `genesis` variables are only assigned to once during the contract's creation.

Recommendation:

As their naming convention implies, they are meant to remain unchanged throughout the lifetime of the contract apart from their original definition during the contract's creation.

As such, we advise that the `immutable` mutability specifier is set on both variables to firstly ensure that their naming convention is properly reflected in code and lastly to greatly optimize the gas cost involved in utilizing them as they are hot-swapped directly in the code as `memory` reads rather than the `storage` reads they currently do.

We advise that the same optimization is applied to the `genesis` variable even though it is meant to be in lower-case to conform to the interface specification.

Finally, the `decimals` variable can directly be declared as a `constant` as it is statically assigned to the value literal of `18`.

SPA-05: Variable Visibility Specifier

Type	Severity	Location
Syntax	Informational	Pool L90

Description:

The linked variables contain no explicit visibility specifiers set.

Recommendation:

We advise that an explicit visibility specifier is set for those variables to aid the readers of the codebase and streamline compiler upgrades as each compiler can set a different default visibility specifier.

SPA-06: Variable State Mutability

Type	Severity	Location
Optimization	Informational	Pool L373, L374, L375

Description:

The `BASE`, `WBNB` and `DEPLOYER` variables are only assigned to once during the contract's creation.

Recommendation:

As their naming convention implies, they are meant to remain unchanged throughout the lifetime of the contract apart from their original definition during the contract's creation.

As such, we advise that the `immutable` mutability specifier is set on both variables to firstly ensure that their naming convention is properly reflected in code and lastly to greatly optimize the gas cost involved in utilizing them as they are hot-swapped directly in the code as `memory` reads rather than the `storage` reads they currently do.

SPA-07: BEP-20 `approve` Race Condition

Type	Severity	Location
Implementation	Informational	Pool L145-L153

Description:

The `approve` function as built by both the ERC-20 and BEP-20 standard is inherently flawed in that it provides a race-condition attack vector as defined in the Smart Contract Weakness Classification registry under no. 114.

Recommendation:

There are multiple approaches to alleviating this attack vector that are entirely up to the developers. Potential approaches include `require`ing that the previously set approval value was `0` or that new functions that increment and decrement the allowance respectively are utilized such as `increaseAllowance` and `decreaseAllowance`. We list this finding as `Informational` despite its severity as it is a widely known issue in the Ethereum community and oft unconsidered.

SPA-08: Contract Bytecode Optimization

Type	Severity	Location
Optimization	Informational	Pool L265-L282, L283-L302

Description:

The linked function pairs `_getAddedBaseAmount` & `_getAddedTokenAmount` and `_swapBaseToToken` & `_swapTokenToBase` contain similar statements and can be optimized.

Recommendation:

We advise that a single function implementation is utilized instead that accepts an `iBEP20` argument denoting the contract to query the new balance from as well as a `uint256` argument that denotes the old balance which is either `_baseBalance` or `_tokenBalance` respectively.

Additionally, the `else` branch as well as explicit `return` statement can be omitted from the function as Solidity data types begin with their default zeroed out value and named return variables are automatically returned at the end of the function they are declared in.

The `_swapBaseToToken` and `_swapTokenToBase` functions can be merged into a single function and optimized identically.

SPA-09: Redundant Casting

Type	Severity	Location
Optimization	Informational	Router L385-L392

Description:

The linked function `migrateRouterData` accepts an `address payable` that is subsequently cast to the `Router` interface on each line assignment.

Recommendation:

Depending on where the function is called from, the function can directly accept a `Router` type variable (if called from an EOA) or create an in-memory variable of the input `address payable` casted to a `Router` that is subsequently utilized.

SPA-10: Migration Guard

Type	Severity	Location
Implementation	Informational	Router L384-L403

Description:

The functions `migrateRouterData` and `migrateTokenData` can be called multiple times overwriting sensitive variables of the contract.

Recommendation:

We advise that a safety guard is set in place that ensures the `Router` is empty before migrating the data i.e. `swapTx == 0` and `arrayTokens.length == 0` respectively for `require` checks.

SPA-11: Function Re-use

Type	Severity	Location
Optimization	Informational	Router L418-L424

Description:

The function `createPool` contains duplicate code from the `addLiquidityForMember` function.

Recommendation:

We advise that the `addLiquidityForMember` or `addLiquidity` function is utilized directly replacing the L418, L419, L422, L423 and L424 statements to reduce the bytecode of the contract and render its maintainability much easier.

SPA-12: Function Re-use

Type	Severity	Location
Implementation	Informational	Router L438-L447, L475-L499

Description:

The function `removeLiquidityExactAndSwap` contains duplicate code from the `removeLiquidityExact` function.

Recommendation:

We advise that the functions are revised to utilize a single `internal` function reducing the bytecode of the contract and rendering its maintainability much easier.

SPA-13: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Optimization	Informational	Router L412, L451, L568, L584

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

SPA-14: Unused Fallback Function

Type	Severity	Location
Optimization	Informational	Router L412, L451, L568, L584

Description:

The only outgoing `ether` transfers that occur in the contract are done so within the `_handleTransferIn` and `_handleTransferOut` functions and those are done with accounted-for funds from either withdrawing `WBNB` or accepting and utilizing the transaction's `msg.value`. As such, any raw funds sent to the contract are locked forever.

Recommendation:

We advise that the fallback function is omitted as it does not affect the functionality of `payable` functions and would ensure no funds remain locked in the contract.

SPA-15: Variable State Mutability

Type	Severity	Location
Optimization	Informational	Utils.sol L147, L148

Description:

The `BASE` and `DEPLOYER` variables are only assigned to once during the contract's creation.

Recommendation:

As their naming convention implies, they are meant to remain unchanged throughout the lifetime of the contract apart from their original definition during the contract's creation.

As such, we advise that the `immutable` mutability specifier is set on both variables to firstly ensure that their naming convention is properly reflected in code and lastly to greatly optimize the gas cost involved in utilizing them as they are hot-swapped directly in the code as `memory` reads rather than the `storage` reads they currently do.

SPA-16: External Call Optimization

Type	Severity	Location
Optimization	Informational	Utils.sol L190-L199

Description:

The `getGlobalDetails` function invokes and casts the result of `dao.ROUTER()` 6 times.

Recommendation:

As the result of the `dao.ROUTER()` function will not change during this function's execution since it is `view`, we advise that its result is casted and stored to an in-memory `iROUTER` variable.

SPA-17: Redundant Casting

Type	Severity	Location
Optimization	Informational	Utils.sol L238-L245

Description:

The linked function `getPoolData` resolves a pool `address` that is subsequently cast to the `iPOOL` interface on almost each line assignment.

Recommendation:

The function can create an in-memory variable of the resolved pool `address` casted to an `iPOOL` that is subsequently utilized.

SPA-18: Incorrect Pool Age Measurement

Type	Severity	Location
Implementation	Informational	Utils.sol L288-L296

Description:

The linked function `getPoolAge` contains an invalid calculation of age for the first "day" of a pool's age.

Recommendation:

The default returned variable of the function is `1`, unless the pool's `genesis` plus a single "day" (`86400`) is greater-than `now` in which case the calculation `(now.sub(genesis)).div(86400)` takes place. As Solidity rounds down, a pool's "age" will be equal to a single day for the duration between `0` and `86400 * 2 - 1` which is incorrect, as the default value of the function alludes that "age" should be measured non-inclusive i.e. `0 - 86399` is "age" `1`, `86400 - 172799` is "age" `2` etc.

We advise that instead of setting an `if` branch, the "age" is normally calculated as done so in L294 with an addition of the literal `1` i.e. `(now.sub(genesis)).div(86400) + 1`.

SPA-19: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Optimization	Informational	Utils.sol L317, L374

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

SPA-20: Supplementary Documentation Request

Type	Severity	Location
Documentation	Informational	Utils.sol L373-L447

Description:

The linked functions are part of the core math implementations of the Spartan Protocol as detailed in their whitepaper.

Recommendation:

We advise that the whitepaper page is directly linked on each formula as well as a direct link to it is provided. We would like to note that we were unable to find the formulas defined in `calcLiquidityUnits`, `getSlipAdustment` and `calcAsymmetricShare` in the whitepaper defined on their `resources` GitHub repository as linked by their official website.