SpartaDex Security Review

A security review of the SpartaDex. The first DEX with gamified yeild.

Author: gkrastenov Independent Security Researcher

This audit report includes all the vulnerabilities, issues and code improvements found during the security review.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behaviour with some of the protocol's functionalities that's not so critical.

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions and the cost of the attack is relatively low to the amount of funds that can be stolen or lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a huge stake by the attacker with little or no incentive.

Executive summary

Overview

Project Name	SpartaDex
Review Commit hash	18941ef176e74c81da9d7a98ff90ffd1c4ab9d80
Fixes Review Commit hash	5cf7351353e3989c6c8272c3afa9dfdeb4bddccb

Scope

File
Contracts (15)
contracts/tokens/StakedSparta.sol
contracts/tokens/PolisMinter.sol
contracts/tokens/Polis.sol
contracts/tokens/Sparta.sol
contracts/PolisManager.sol

File
contracts/dex/periphery/SpartaDexRouter.sol
contracts/dex/core/SpartaDexFactory.sol
contracts/dex/core/SpartaDexERC20.sol
contracts/dex/core/SpartaDexPair.sol
contracts/staking/LinearStaking.sol
contracts/staking/SpartaStaking.sol
contracts/staking/LPLinearStaking.sol
contracts/SpartaRewardLpLinearStaking.sol
contracts/staking/LPLinearStaking.sol
contracts/WithFees.sol
Interfaces (9)
ccontracts/IPolisManager.sol
contracts/tokens/interfaces/IPolisMinter.sol
contracts/tokens/interfaces/IStakedSparta.sol
contracts/tokens/interfaces/IPolis.sol
contracts/tokens/interfaces/ISparta.sol
contracts/IWithFees.sol
contracts/staking/interfaces/ILinearStaking.sol

File
contracts/staking/interfaces/ISpartaStaking.sol
contracts/payment-receiver/IPaymentReceiver.sol
Total (24)
nSLOC (1966)

Issues found

Severity	Count	Fixed	Acknowledged
High risk	2	2	0
Medium risk	6	6	0
Low risk	8	8	0
Informational	18	16	2
Gas	6	5	1
Total	**40**	**37**	**3**

Findings

ID	Title	Severity
H-1	Token can be stucked in the staking contract	High
H-2	Wrong implementation of swapTokensForExactTokens	High

ID	Title	Severity
M-1	weiRewardRatioPerTokenStored() will not work most of the time	Medium
M-2	The owner can not lock tokens without granting approval rights to another user	Medium
M-3	Signature replay attack is possible	Medium
M-4	ChainId is missed in EIP712 Type hash	Medium
M-5	withdrawTokensToClaimFromRounds will not work as expected	Medium
M-6	Use safeTransferFrom() instead of transferFrom()	Medium
L-1	Double approving to same address is possible	Low
L-2	Hard-coded chainId in DOMAIN_SEPARATOR	Low
L-3	The tokenId is not checked for existence during the buying of gems	Low
L-4	The wallet can be equal to address(0) during the buying of gems	Low
L-5	Typos in EIP712Domain separator and UPGRADE_TYPE	Low
L-6	Transferring ownership in the constructor is unnecessary	Low
L-7	Upgrading of not existing token is possible	Low
L-8	UPGRADE_TYPE and BUY_TYPE contain unnecessary parametars	Low
I-1	Missed license	Informational
I-2	Redundant events or errors	Informational
I-3	Library ECDSA is never used in Polis contract	Informational
I-4	Argument in modifier is redundant	Informational

ID	Title	Severity
I-5	Approving to zero address is possible	Informational
I-6	updateReward modifier is unnecessary in initialize function	Informational
I-7	Redundant function	Informational
I-8	Naming collision in WithFees contract	Informational
I-9	Emit event in crucial places	Informational
I-10	Use custom error instead of require statement	Informational
I-11	Rename function setup to initialize	Informational
I-12	Modifiers should be declared before functions	Informational
I-13	Add NatSpec documentation	Informational
I-14	Missing non-zero address checks	Informational
I-15	Import declarations should import specific identifiers, rather than the whole file	Informational
I-16	Remove redundant import	Informational
I-17	Add additional checks for the variables duration and start	Informational
I-18	Override keyword is missed in some functions	Informational
G-1	Using private rather than public for constants, saves gas	Gas
G-2	Constructors can be marked payable	Gas
G-3	++i/i++ should be unchecked{++i}/unchecked{i++} when it is not possible for them to overflow	Gas
G-4	Don’t initialize variables with default value	Gas

ID	Title	Severity
G-5	Cache storage values in memory to minimize SLOADs	Gas
G-6	Use calldata instead of memory	Gas

High

[H-01] Token can be stucked in staking contract

Impact

If the owner grants rights to another user to lock his token but later decides to remove those rights, the token will become stuck in the contract because only the locker can unlock.

```
    function unlock(uint256 _tokenId) external override {
        if (lockerOf(_tokenId) != msg.sender) {
            //@audit can be stucked if approved operator is removed
            revert CannotUnlock();
        }

        delete _lockers[_tokenId];

        emit Unlock(_tokenId, msg.sender);
    }
```

Recommended Mitigation Steps

Owner should be able to unlock his token.

Fixes Review

Fixed.

[H-02] Wrong implementation of swapTokensForExactTokens

Impact

The check for excessive input amount is wrongly implemented in the `swapTokensForExactTokens` function. The swap will be successful only when `amounts[0]` is greater than or equal to `amountInMax`

`amounts[0]` represents the actual amount of input tokens that will be spent during the token swap. It is calculated based on the token exchange rates and liquidity available in the SpartaDex pairs. `amountInMax` is the maximum amount of input tokens that you are willing to spend.

The actual amount of input tokens should not be bigger than the maximum amount that the user is willing to spend.

For reference: https://github.com/Uniswap/v2-periphery/blob/master/contracts/UniswapV2Router02.sol#L246

```
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
        //@audit wrong if condition
```

```
        if (amounts[0] < amountInMax) {
            revert ExcessiveInputAmount();
        }

        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            UniswapV2Library.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        _swap(amounts, path, to);
    }
```

Recommended Mitigation Steps

Make the following changes:

```
- if (amounts[0] < amountInMax)
+ if (amounts[0] > amountInMax)
```

Fixes Review

Fixed.

Medium

[M-01] weiRewardRatioPerTokenStored() will not work most of the time

Impact

This function calculates the ratio between the reward wei and the stored tokens. If the `totalSupply` is zero, it returns 0. When the `totalSupply` is not equal to 0, the return value is the division of the `rewardRate` and the `totalSupply`. If the `totalSupply` is greater than the `rewardRate`, precision loss may occur.

```
function weiRewardRatioPerTokenStored() external view returns (uint256) {
        //@audit precision loss if totalSupply > rewardRate
        return totalSupply == 0 ? 0 : rewardRate / totalSupply;
    }
```

PoC

```
  it.only("Medium 1: weiRewardRatioPerTokenStored() -> precision loss", async function () {
    const initialAmount = ethers.utils.parseEther("30000000");
    await mintTokens(instance.address, initialAmount);

    await mintTokensToContract();

    const stakerAmount = ethers.utils.parseEther("100");
    await mintTokens(staker.address, stakerAmount);
    await sparta.connect(staker).approve(instance.address, stakerAmount);

    const aliceAmount = ethers.utils.parseEther("200");
    await mintTokens(alice.address, aliceAmount);
    await sparta.connect(alice).approve(instance.address, aliceAmount);

    const bobAmount = ethers.utils.parseEther("1200");
    await mintTokens(bob.address, bobAmount);
    await sparta.connect(bob).approve(instance.address, bobAmount);

    const duration = time.duration.seconds(2500000); // around 30 days (28.93)
    await instance
      .connect(stakingOwner)
      .initialize(initialAmount, stakingStart, duration);
```

```
    // rewardRate = 12*1e18
    await time.setNextBlockTimestamp(stakingStart);

    await instance.connect(staker).stake(stakerAmount); // 100*1e18
     // totalSupply = 100*1e18 which is bigger than rewardRate
    let ratio =  await instance.weiRewardRatioPerTokenStored(); // 100*1e18 > rewardRate
    expect(ratio).to.be.eq('0');

    await instance.connect(alice).stake(aliceAmount); // 200*1e18
     // totalSupply = 300*1e18
    ratio =  await instance.weiRewardRatioPerTokenStored();  // 300*1e18 > rewardRate
    expect(ratio).to.be.eq('0');


    await instance.connect(bob).stake(bobAmount); // 1200*1e18
    // totalSupply = 1500*1e18
    ratio =  await instance.weiRewardRatioPerTokenStored(); // 1500*1e18 > rewardRate
    expect(ratio).to.be.eq('0');
  });
```

Recommended Mitigation Steps

Find a different way to calculate this ratio.

Fixes Review

Fixed.

[M-02] The owner can not lock tokens without granting approval rights to another user

Impact

The owner is not able to lock his own token because first he has to approve the operator/locker. He is forced to always grant permissions to someone else which may not be desirable in some scenarios. For example, he may not trust another user.

PoC

```
it.only("Medium 2: The owner can not lock their tokens", async function () {
     await instance.connect(tokenOwner).mint();
     await instance.connect(tokenOwner).lock(1); // revert with CannotLock()
     //
     expect(await instance.lockerOf(1)).to.be.equal(tokenOwner.address);
     expect(await instance.totalSupply()).to.be.equal(1);
 });
```

Recommended Mitigation Steps

If `msg.sender` is the owner of the token, then allow them to lock their token

Fixes Review

Fixed.

[M-03] Signature replay attack is possible

Impact

A signature replay attack is possible during token upgrades. If the deadline is set too far in the future or has not yet expired, a previously used signature for upgrading can be reused.

PoC

```
it.only("Signature replay attack", async () => {
  await polis.mintAsMinter(owner1.address);
  const levelBefore = await polis.senateLevel(1);
  await polis.connect(owner1).setLockApprovalForAll(instance.address, true);
  await instance.connect(owner1).lock(1);
  const hash = await instance.upgradeHash(1, 1);
  const signature = deployer.signMessage(ethers.utils.arrayify(hash));

  await instance.upgradeWithSignature(1, 1, signature);
  let lv = await polis.senateLevel(1);
  console.log(lv); // senate = 1

  expect(await polis.senateLevel(1)).to.be.equal(levelBefore + 1);

  // Attack:
  const maxLevel = 254;
  for (let i = 0; i < maxLevel; i++) {
    await instance.upgradeWithSignature(1, 1, signature);
  }
  lv = await polis.senateLevel(1);
  console.log(lv); // senate = 255

  // Reached max level:
  expect(await polis.senateLevel(1)).to.be.equal(levelBefore + 255); // => true
});
```

Recommended Mitigation Steps

Add a nonce for every signature that the user signs.

Fixes Review

Fixed.

[M-04] ChainId is missed in EIP712 Type hash

Impact

ChainId is missed in EIP712 Type hash in `PolisManager` and `PaymentReceiver` contracts.

```
    function domainSeperator() internal view returns (bytes32) {
        return
            keccak256(
                abi.encode(
                    keccak256( //@audit Wrong domain separator, missed chainId
                        "EIP712Domain(string name, string version, address verifyingContract, uint256 signedAt)"
                    ),
                    keccak256(bytes("Sparta")),
                    keccak256(bytes("1")),
                    _chainId(),
                    address(this),
                    keccak256(bytes("Sparta"))
                )
            );
    }
```

Recommended Mitigation Steps

Change type hash to:

```
- "EIP712Domain(string name, string version, address verifyingContract, uint256 signedAt)"
+ "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract,uint256 signedAt)"
```

Fixes Review

Fixed.

[M-05] withdrawTokensToClaimFromRounds will not work as expected

Impact

`withdrawTokensToClaimFromRounds` function will not work as expected because every time `roundsLength` is equal to 0 and the for loop will not cycle through all the given rounds.

```
    function withdrawTokensToClaimFromRounds(uint256[] memory rounds) external {
        uint256 roundsLength = 0; //@audit M: roundsLength = rounds.length
        for (uint roundIndex = 0; roundIndex < roundsLength; ++roundIndex) {
            withdrawTokensToClaim(rounds[roundIndex]);
        }
    }
```

Recommended Mitigation Steps

Make the following changes:

```
- uint256 roundsLength = 0;
+ uint256 roundsLength = rounds.length;
```

Fixes Review

Fixed.

[M-06] Use safeTransferFrom() instead of transferFrom()

It is more preferable to use safeTransferFrom instead of transferFrom because if '_to ' is a contract address that does not support ERC721, the NFT can be frozen in that contract.

Also, OpenZeppelin's documentation discourages the use of transferFrom(). Use safeTransferFrom() whenever possible because transferFrom() can not check whether the receiving address know how to handle ERC721 tokens.

Fixes Review

Fixed.

Low severity

[L-01] Double approving to same address is possible

Impact

It is possible to double-approve the same address in the `lockApprove()` function of the `Polis` contract. This behavior is not expected, and the `LockApproval` event with the same `owner`, `to`, and `tokenId` will be emitted again.

```
    function lockApprove(
        address _to,
        uint256 _tokenId
    ) public virtual isNotLocked(_tokenId) {
        //@audit double approving to same address is possible
        address owner = ERC721A.ownerOf(_tokenId);
        if (_to == owner) {
            revert SelfApproval();
        }
        bool canApprove = msg.sender == owner ||
            _lockOperatorApprovals[owner][msg.sender];
```

```
        if (!canApprove) {
            revert CannotApprove();
        }

        _lockApprove(owner, _to, _tokenId);
    }
```

Recommended Mitigation Steps

Add additional check for that.

```
if (lockApprovals[tokenId] == _to) {
   revert  AlreadyApproved();
}
```

Fixes Review

Fixed.

[L-02] Hard-coded chainId in DOMAIN_SEPARATOR

Impact

Chain ID should be computed dynamically rather than being hard-coded into the DOMAIN_SEPARATOR during initialization.

```
constructor() {
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256(
```

```
                "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
            ),
            keccak256(bytes(name)),
            keccak256(bytes("1")),
            1729, //@audit hard-coded chainId
            address(this)
        )
    );
}
```

Recommended Mitigation Steps

Use similar approach as in the `PaymentReceiver` contract.

Fixes Review

Fixed.

[L-03] The tokenId is not checked for existence during the buying of gems

Impact

Gems can be bought and a `GemsPurchased` event can be emitted when the `tokenId` does not exist.

```
function buyGems(
    address _wallet,
    uint256 _tokenId, // polis nft
    uint256 _amount,
    uint256 _price,
    uint256 _deadline,
    bytes calldata _signature
) external override deadlineIsNotMissed(_deadline) {
```

```
        //@audit _tokenId is not checked if exist

        bytes32 hash = buyGemsnHash(
            _wallet,
            _tokenId,
            _amount,
            _price,
            _deadline
        );

        address signer = hash.toEthSignedMessageHash().recover(_signature);
        _esnureHasGemsTraderRole(signer);

        if (!paymentToken.transferFrom(msg.sender, treasury, _price)) {
            revert TransferFailed();
        }

        emit GemsPurchased(
            _wallet,
            collection,
            _tokenId,
            msg.sender,
            _amount,
            _price
        );
    }
```

Recommended Mitigation Steps

Add an additional check if the `tokenId` exists.

```
  if (collection.ownerOf(_tokenId) == address(0)) {
      revert TokenNotExists();
```

```
}
```

Fixes Review

Fixed.

[L-04] The wallet can be equal to address(0) during the buying of gems

Impact

Gems can be bought and a `GemsPurchased` event can be emitted when the wallet is equal to address(0).

```
function buyGems(
    address _wallet,
    uint256 _tokenId, // polis nft
    uint256 _amount,
    uint256 _price,
    uint256 _deadline,
    bytes calldata _signature
) external override deadlineIsNotMissed(_deadline) {
    //@audit _wallet can be eqaul to zero

    bytes32 hash = buyGemsnHash(
        _wallet,
        _tokenId,
        _amount,
        _price,
        _deadline
    );

    address signer = hash.toEthSignedMessageHash().recover(_signature);
    _esnureHasGemsTraderRole(signer);
```

```
        if (!paymentToken.transferFrom(msg.sender, treasury, _price)) {
            revert TransferFailed();
        }

        emit GemsPurchased(
            _wallet,
            collection,
            _tokenId,
            msg.sender,
            _amount,
            _price
        );
    }
```

Recommended Mitigation Steps

Add an additional check if wallet is eqaul to zero address.

```
if (_wallet_ == address(0)) {
    revert ZeroAddress();
}
```

Fixes Review

Fixed.

[L-05] Typos in EIP712Domain separator and UPGRADE_TYPE

Impact

Typos occur in the EIP712 Domain separator and the UPGRADE_TYPE constant variable.

For additional information: https://eips.ethereum.org/EIPS/eip-712

Recommended Mitigation Steps

Make the following changes:

```
- string constant UPGRADE_TYPE = "upgrade(uint256 tokenId, uint8 level, bytes signature)";
+ string constant UPGRADE_TYPE = "upgrade(uint256 tokenId,uint8 level,bytes signature)";


- "EIP712Domain(string name, string version, address verifyingContract, uint256 signedAt)"
+ "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract,uint256 signedAt)"
```

[L-06] Transferring ownership in the constructor is unnecessary

Impact

Transferring ownership in the constructor is unnecessary because it is already done in the base class.

```
    constructor(
        IERC20 sparta_,
        IStakedSparta stakedSparta_,
        IAccessControl _acl,
        address _treasury,
        uint256 _value
    ) WithFees(_acl, _treasury, _value) {
        sparta = sparta_;
        stakedSparta = stakedSparta_;
```

```
        // @audit unnecessary
        _transferOwnership(msg.sender);
    }
```

Fixes Review

Fixed.

[L-07] Upgrading of not existing token is possible

Impact

Upgrading a non-existing token is possible in the `Polis` contract.

```
    function upgrade(uint256 tokenId) external onlyUpgradeRoleAccess {
        //@audit Non-existing tokens can be upgraded.
        _upgrade(tokenId);
    }
```

Recommended Mitigation Steps

Add an additional check to verify if the token exists.

Fixes Review

Fixed.

[L-08] UPGRADE_TYPE and BUY_TYPE contain unnecessary parametars

The parameter `signature` is unnecessary because it is never used during the creation of the hash. UPGRADE_TYPE BUY_TYPE

If you are not sure how to implement EIP712 in your smart contract.

Than you can follow this article: ref

Fixes Review

Fixed.

Information

[I-01] Missed license in SpartaDexRouter

The license `SPDX-License-Identifier: Unlicense` is missing in `SpartaDexRouter` and most of the files in the .../dex/periphery folder.

Fixes Review

Fixed.

[I-02] Redundant events or errors

Following erros and events are redundant `error OnlyLockerRole()`, `error SignatureExceeded(uint256 deadline)`, `error AlreadyLocked()` in `PolisMinter` contract

Fixes Review

Fixed.

[I-03] Library ECDSA is never used in Polis contract

Library ECDSA is never used in `Polis` contract.

```
using ECDSA for bytes32; //@audit never used in this contract
```

Fixes Review

Fixed.

[I-04] Argument in modifier is redundant

Argument `tokenId` is redundant in `onlyOneTokenLockedByUser` modifier.

```
modifier onlyOneTokenLockedByUser(uint256 tokenId) { //@audit tokenId argument is not used
        if (lockedTokens[msg.sender] != 0) {
            revert OnlyOneTokenLocked();
        }
        _;
    }
```

Fixes Review

Fixed.

[I-05] Approving to zero address is possible

The owner can approve address(0) to have rights to lock tokens. This action will not have any negative impact but this is not expected behavior

```
function setLockApprovalForAll(
    address operator,
    bool approved
) public virtual {
    //@audit approving to zero address is possible
    if (msg.sender == operator) {
        revert SelfApproval();
    }
    _setLockApprovalForAll(msg.sender, operator, approved);
}
```

Fixes Review

Fixed.

[I-06] updateReward modifier is unnecessary in initialize function

The `updateReward` modifier is unnecessary in the `initialize` function because the variable `updatedAt` is set to block.timestamp within the `initialize` function and `rewardPerTokenStored` is set to zero by default.

Fixes Review

Fixed.

[I-07] Redundant function

Remove the redundant function `foo` in the `SpartaDexFactory` contract.

```
function foo() external pure {}
```

Fixes Review

Fixed.

[I-08] Naming collision in WithFees contract

A naming collision occurs in the `WithFees` contract between the storage variable `value` and the transfer of ether using the `call` function. This will not have any negative impact while sending money.

```
(bool sent, ) = treasury.call{value: address(this).balance}("");
 //@audit naming collision with storage variable value
```

Recommended Mitigation Steps

Make the following changes:

```
- uint256 public immutable override value;
+ uint256 public immutable override fee;
```

Fixes Review

Fixed.

[I-09] Emit event in crucial places

Emit event in crucial place like in `setup()` function in `LinearStaking` contract.

```
function setup(
       uint256 _amount,
```

```
        uint256 _duration,
        uint256 _start
    ) external notInitialized onlyOwner updateReward(address(0)) {
        require(
            _amount <= rewardToken.balanceOf(address(this)),
            "reward amount > balance"
            //@fix: use error
        );

        duration = _duration;
        rewardRate = _amount / duration; //gas: use _duration
        start = _start;
        updatedAt = block.timestamp;

        initialized = true;

+       emit Initialized(_start, _duration, _amount);
    }
```

Fixes Review

Fixed.

[I-10] Use custom error instead of require statement

Use custom error instead of a require statement in `setup` function in `LinearStaking` contract.

```
  function setup( //@audit follow good standard should be initialize
        uint256 _amount,
        uint256 _duration,
        uint256 _start
    ) external notInitialized onlyOwner updateReward(address(0)) {
        require(
```

```
            _amount <= rewardToken.balanceOf(address(this)),
            "reward amount > balance"
            //@audit nc: use custom error
        );
}
```

Fixes Review

Fixed.

[I-11] Rename function setup to initialize

Rename function `setup` to `initialize` in `LinearStaking` contract. It is recommended to follow established design standards.

```
   function setup( //@audit follow good standards
          uint256 _amount,
          uint256 _duration,
          uint256 _start
      ) external notInitialized onlyOwner updateReward(address(0))
```

Fixes Review

Fixed.

[I-12] Modifiers should be declared before functions

It is good practice modifiers to be declared before function. Currently, they are below the functios.

Fixes Review

Fixed.

[I-13] Add NatSpec documentation

NatSpec documentation to all public methods and variables is essential for better understanding of the code by developers and auditors and is strongly recommended.

Fixes Review

Acknowledged.

[I-14] Missing non-zero address checks

Add non-zero address checks for all address type arguments.

Fixes Review

Fixed.

[I-15] Import declarations should import specific identifiers, rather than the whole file

Using import declarations of the form `import {<identifier_name>} from "some/file.sol"` avoids polluting the symbol namespace making flattened files smaller and speeds up compilation

Fixes Review

Acknowledged.

[I-16] Remove redundant import

The interface IUniswapV2Pair is not used in the `Polis` contract.

```
import "../dex/core/interfaces/IUniswapV2Pair.sol";
```

Fixes Review

Fixed.

[I-17] Add additional checks for the variables duration and start

Add additional checks for the variables `duration` and `start` in the `initialize` function. The `duration` should be greater than zero, and the `start` variable should be greater than or equal to block.timestamp

Fixes Review

Fixed.

[I-18] Override keyword is missed in some functions

The override keyword is missed in the `mintTo` and `burnFrom` functions in the `StakedSparta` contract.

Fixes Review

Fixed.

Gas

[G-01] Using private rather than public for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves 3406-3606 gas in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table.

Instances: WithFees: 12; PolisManager: 11, 14; PaymentReceiver: 17; Polis: 14-15; PolisMinter: 13; Sparta: 8; StakedSparta: 17;

Fixes Review

Fixed.

[G-02] Constructors can be marked payable

Payable functions cost less gas to execute, since the compiler does not have to add extra checks to ensure that a payment wasn't provided. A constructor can safely be marked as payable, since only the deployer would be able to pass funds, and the project itself would not pass any funds.

Fixes Review

Acknowledged.

[G-03] ++i/i++ should be unchecked{++i}/unchecked{i++} when it is not possible for them to overflow

The unchecked keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves 30-40 gas per loop

Instances: SpartaStaking: 224

Fixes Review

Fixed.

[G-04] Do not initialize variables with default value

Uninitialized variables are assigned with the types default value. Explicitly initializing a variable with it's default value costs unnecesary gas.

Instances: ToInitialize: 8; SpartaStaking: 194, 195, 201, 203, 224,

Fixes Review

Fixed.

[G-05] Cache storage values in memory to minimize SLOADs

Use directly `_duration` instead of storage variable `duration` : 1, 2

Fixes Review

Fixed.

[G-08] Use calldata instead of memory

Use calldata instead of memory for function parameters saves gas if the function argument is only read. Instances: SpartaStaking: 193

Fixes Review

Fixed.