Swingby Yield Farming Contract Security Audit

Table of Contents

Scope

The scope of the audit is the following:

Swingby PCS-LP Staking
Commit: d459a16e6a283c74b4b55171764f4bf52fba3574

Swingby UniswapV2 & Sushiswap LP staking
Commit: 020f49c4c8f095da643ab6bfd852fa3f6945b9d2

SbBTC LP Staking
Commit: 51bf0b3d708e7cc80d75eae682d99c7f09d07f24

Overview

Swingby PCS-LP Staking

Users can stake their SWINGBY/BNB Pancakeswap LP tokens in this contract, which will then be deposited into the Pancakeswap masterchef. This allows the earning of both SWINGBY rewards, as well as any CAKE rewards emitted in the underlying PCS Masterchef, if any. This contract will be deployed on Binance Smart Chain.

Swingby UniswapV2 & SushiSwap LP Staking

Users can stake their SWINGBY/WETH UniswapV2 and SushiSwap LP tokens in this contract to earn SWINGBY rewards. This contract will be deployed on Ethereum Mainnet.

sbBTC LP Staking

Users can stake their SbBTC LP tokens in this contract to earn SWINGBY rewards. The emissions for this contract are dynamic and based on the balance of the Skybridge contract's float. Emissions increase as the BTC/wBNB float of the Skybridge contract gets more balanced. There is an upper and lower cap for the emissions. This contract will be deployed on Ethereum Mainnet.

Summary of Findings

In performing a security audit of the Skybridge Yield Farming Contracts, several issues of concern were found. For each finding, a summary of the issue is documented, along with any other finer details regarding the issue. Security recommendations are also provided where applicable.

The table below shows a breakdown of security findings found categorized by severity or risk and impact. A finding that has been reported is listed as Unresolved, and if that finding is satisfactorily mitigated, it will be categorized as Resolved.

Severity	**Resolved**	**Unresolved**	**Total**
Critical	0	0	0
High	1	0	1
Medium	2	1	3
Low	2	4	6
Info	13	14	27

Issues

Swingby PCS-LP Staking

FDEV-001: Unrequired functionality for single pool only Masterchef

Severity: Low

Since there is only 1 pool (SWINGBY/BNB PCS LP), it is unnecessary for features such as the poolInfo array, and set and add pool. Removal of adding/setting new pools would also limit the amount of malicious actions a privileged role can do.

Recommendations

Pancakeswap's SmartChef (https://bscscan.com/address/0x97058cf9b36c9ef1622485cef22e72d6fea32a36#code) which has only a single stakeable token would be a more fitting base rather than the standard Masterchef. The one and only PoolInfo can be initialized in the init function.

Notes

Acknowledged with no change.

FDEV-002: Typo in rewardCoinsDept

Severity: Info

There is a typo in rewardCoinsDept.

Recommendations

rewardCoinsDept should be user.rewardCoinsDebt.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/bfd972d218376737289b3be4b4004ffa959cfe41.

FDEV-003: safeTransfer should be used for pool.farmCoin transfers

Severity: Low

Currently, IERC20(pool.farmCoin).transfer is used to transfer the farmCoin, even though safeTransfer is supported due to it using IERC20.

Recommendations

Like the pool.lpToken transfers, pool.farmCoin should also use safeTransfer.

Additionally, farmToken in the PoolInfo struct can be set as IERC20 instead of address.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/306ae585f952dec13c8f028ae3825713a03c6b5a.

FDEV-004: Conditional != 0 can be used instead of > 0 for unsigned integer comparison

Severity: Info

User.amount > 0 is used for the condition that user.amount is non-zero, even though user.amount is a uint256 type, and will never be negative.

Recommendations

Use if (user.amount != 0) instead.

Notes

Acknowledged but no change made.

FDEV-005: Do safeTransfer and safeTransferFrom only when transfer amount is non zero

Severity: Info

Users can deposit and withdraw 0 amounts, and the safeTransferFrom and safeTransfer functions for the LP token will be called unnecessarily if the amount is 0.

Recommendations

Add a conditional check to only call safeTransferFrom and safeTransfer if amount != 0.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/01fe7df1c77d695577afafd86310c62867be9c8d

---

FDEV-006: Unnecessary multiplication with BONUS_MULTIPLIER

Severity: Info

BONUS_MULTIPIER is a constant set at 1, and multiplying a number against it would result in the same number.

Recommendations

Remove the multiplication of BONUS_MULTIPLIER as it is an unnecessary use of gas. BONUS_MULTIPLIER can also be removed.

Notes

Acknowledged but no change made.

FDEV-007: Lack of max cap for swingbyPerBlock in modify

Severity: Low

The owner of the ChefLink contract can modify the swingbyPerBlock to an arbitrary amount. A malicious owner can set it to a high amount for a temporary period of time to increase emissions for his own benefit.

Recommendations

Add a max emission check in the modify function.

Notes

Acknowledged but no change made.

FDEV-008: safeSWINGBYTransfer does not check for bool return value of transfer

Severity: Info

safeSWINGBYTransfer uses transfer instead of safeTransfer, which returns a bool value. This return value is not checked.

Recommendations

Add a require statement to ensure that the return value is true.

Notes

Acknowledged but no change made.

FDEV-009: Add nonReentrant modifier to user facing functions

Severity: Info

Although both deposit and reward tokens are ERC20 tokens and do not have any behavior, it is recommended to the nonReentrant as there have been cases where a MasterChef fork has faced a reentrancy attack due to the underlying reward token having a callback functionality.

Recommendations

Add nonReentrant modifier to user facing functions such as deposit, withdraw and emergencyWithdraw.

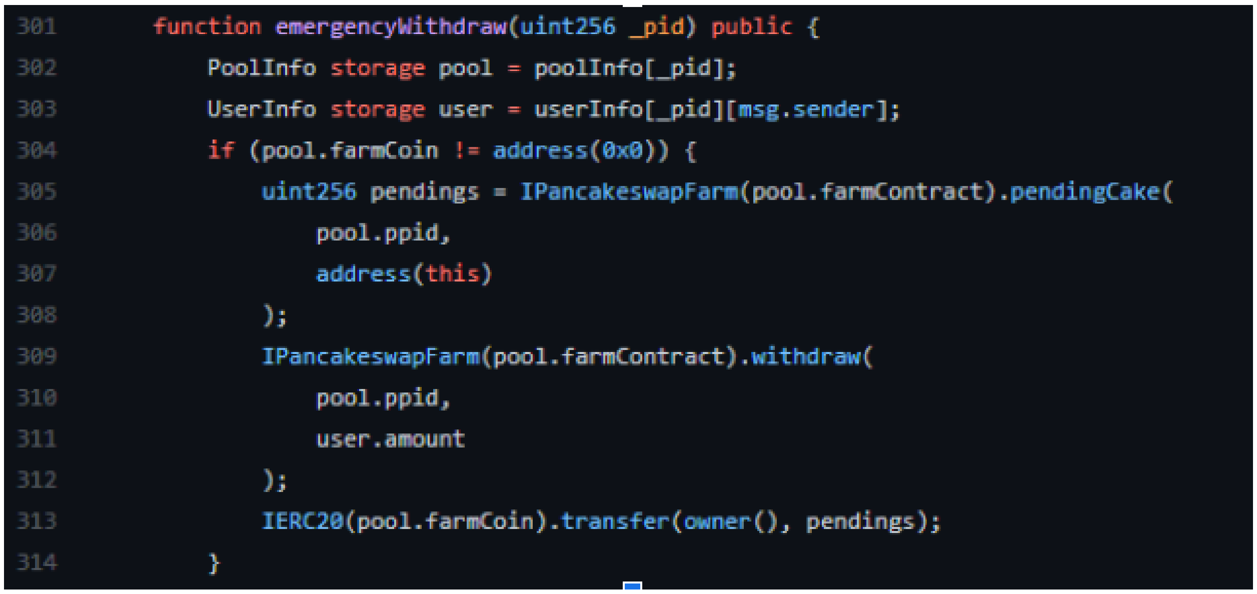
Notes

Acknowledged but no change made.

FDEV-010: Improper calculation of pending cakes sent to owner in emergencyWithdraw

Severity: High

Whenever a user does emergencyWithdraw, all the pending cakes will be sent to the owner of ChefLink, as seen in the calculation in L305.



```
301     function emergencyWithdraw(uint256 _pid) public {
302         PoolInfo storage pool = poolInfo[_pid];
303         UserInfo storage user = userInfo[_pid][msg.sender];
304         if (pool.farmCoin != address(0x0)) {
305             uint256 pendings = IPancakeswapFarm(pool.farmContract).pendingCake(
306                 pool.ppid,
307                 address(this)
308             );
309             IPancakeswapFarm(pool.farmContract).withdraw(
310                 pool.ppid,
311                 user.amount
312             );
313             IERC20(pool.farmCoin).transfer(owner(), pendings);
314         }
```

This is an incorrect calculation, and can be used by the owner to drain pending rewards belonging to all users by making frequent emergencyWithdraw calls.

Recommendation

The standard emergencyWithdraw behavior is that the user forfeits his share of the rewards, which will remain in the MasterChef. It would be recommended to not transfer the pendingCake to the owner.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/6f64adec2ae9bbe9761bc87478faac7b1b4300e6. Only the owner can call panic to emergencyWithdraw from Pancakeswap's Masterchef. The LPs will remain in ChefLink, and users cannot make normal deposit or withdraw calls. Users can still withdraw their deposited LPs by calling emergencyWithdraw.

FDEV-011: Checks-Effects-Interactions pattern not followed in emergencyWithdraw

Severity: Low

The following 3 instructions are done only after an external call (transfer) is done.

```
user.amount = 0;
user.rewardDebt = 0;
user.rewardCoinsDept = 0;
```

This can allow for reentrancy attacks if the external call is done to a contract (e.g. custom token) that allows such behavior.

Recommendations

As the user.amount is used, the value of it can be cached, followed by setting the user's data all to 0, and finally doing the external call (transfer).

```
uint256 _amount = user.amount

user.amount = 0;
user.rewardDebt = 0;
user.rewardCoinsDept = 0;
pool.totalStaked = pool.totalStaked.sub(_amount);

pool.lpToken.safeTransfer(address(msg.sender), _amount);
emit EmergencyWithdraw(msg.sender, _pid, _amount);
```

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/6ffc900f5a3a5861b29ab417efed81f4f92507fe and https://github.com/SwingbyProtocol/defi-contract/commit/306ae585f952dec13c8f028ae3825713a03c6b5a

FDEV-012: Public functions can be external

Severity: Info

The following functions are not called within the same contract, and thus can be set to external:

```
init
add
modify
set
deposit
withdraw
emergencyWithdraw
kynkyuJitaiSengen
kynkyuJitaiSengenKaijo
```

Recommendation

Change the visibility from public to external.

Notes

Acknowledged but no change made.

Swingby UniswapV2 & SushiSwap LP Staking

CHL-001: Lack of max cap for swingbyPerBlock in modify

Severity: Low

The owner of the ChefLink contract can modify the swingbyPerBlock to an arbitrary amount. A malicious owner can set it to a high amount for a temporary period of time to increase emissions for his own benefit.

Recommendations

Add a max emission check in the modify function.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-002: Checks-Effects-Interactions pattern not followed in emergencyWithdraw

Severity: Low

The following 3 instructions are done only after an external call (transfer) is done.

```
user.amount = 0;
user.rewardDebt = 0;
```

This can allow for reentrancy attacks if the external call is done to a contract (e.g. custom token) that allows such behavior.

Recommendations

As the user.amount is used, the value of it can be cached, followed by setting the user's data all to 0, and finally doing the external call (transfer).

```
uint256 _amount = user.amount

user.amount = 0;
user.rewardDebt = 0;

pool.lpToken.safeTransfer(address(msg.sender), _amount);
emit EmergencyWithdraw(msg.sender, _pid, _amount);
```

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-003: Lack of duplicate LP token check in add

Severity: Info

Adding the same LP token more than one can result in some complications with the rewards calculation, as stated in the comments of the Masterchef contract.

There is currently no check to prevent the same LP token from being added more than once.

Recommendations

Add a modifier for the add function to ensure that a duplicate token cannot be added.

```
mapping(IERC20 => bool) public poolExistence;
modifier nonDuplicated(IERC20 _lpToken) {
    require(!poolExistence[_lpToken], "nonDuplicated: duplicated");
    _;
}
```

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-004: Token balances are using the actual token balance of the contract instead of cached balances

Severity: Info

In updatePool, the lpSupply, which is used as the denominator for calculating the swingbyReward to be added to the current accSwingbyPerShare of a pool, is using the balanceOf function of the token to obtain the balances.

```
uint256 lpSupply = pool.lpToken.balanceOf(address(this));
[...]
pool.accSwingbyPerShare = pool.accSwingbyPerShare.add(
            swingbyReward.mul(1e12).div(lpSupply)
        );
```

In some cases where the balances of tokens could include token balances besides the amount deposited by users (e.g. reward tokens are also stored in the same MasterChef contract), this can affect the calculation of the rewards for users.

Recommendations

Like PCS LP Staking, use a cached token balance for each pool.

This cache balance should be incremented with the deposited amount in `deposit`, and decremented by the withdraw amount in both `withdraw` and `emergencyWithdraw`.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-005: Conditional != 0 can be used instead of > 0 for unsigned integer comparison

Severity: Info

`user.amount > 0` is used for the condition that user.amount is non-zero, even though user.amount is a uint256 type, and will never be negative.

Recommendations

Use `if (user.amount != 0)` instead.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-006: Unnecessary multiplication with BONUS_MULTIPLIER

Severity: Info

BONUS_MULTIPIER is a constant set at 1, and multiplying a number against it would result in the same number.

Recommendations

Remove the multiplication of BONUS_MULTIPLIER as it is an unnecessary use of gas. BONUS_MULTIPLIER can also be removed.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-007: safeSWINGBYTransfer does not check for bool return value of transfer

Severity: Info

safeSWINGBYTransfer uses transfer instead of safeTransfer, which returns a bool value. This return value is not checked.

Recommendations

Add a require statement to ensure that the return value is true.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-008: Do safeTransfer and safeTransferFrom only when transfer amount is non zero

Severity: Info

Users can deposit and withdraw 0 amounts, and the safeTransferFrom and safeTransfer functions for the LP token will be called unnecessarily if the amount is 0.

Recommendations

Add a conditional check to only call safeTransferFrom and safeTransfer if amount != 0.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-009: Public functions can be external

Severity: Info

The following functions are not called within the same contract, and thus can be set to external:

```
init
add
modify
set
deposit
withdraw
emergencyWithdraw
```

Recommendation

Change the visibility from public to external.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

CHL-010: Add nonReentrant modifier to user facing functions

Severity: Info

Although both deposit and reward tokens are ERC20 tokens and do not have any callback behavior, it is recommended to the nonReentrant as there have been cases where a MasterChef fork has faced a reentrancy attack due to the underlying reward token having a callback functionality.

Recommendations

Add nonReentrant modifier to user facing functions such as deposit, withdraw and emergencyWithdraw.

Notes

Acknowledged but no change made as the contract has already been deployed at point of review.

sbBTC LP Staking

CLM-001: Initialization can be moved into constructor

Severity: Info

All the state variables can be moved into the constructor, which is empty now. This allows the setting of some of the non-changing state variables immutable to save some gas.

Recommendations

Move the following into the constructor

```
stakedToken = _stakedToken;
rewardToken = _rewardToken;
rewardPerBlock = _rewardPerBlock;
defaultRewardPerBlock = rewardPerBlock;
startBlock = _startBlock;
bonusEndBlock = _bonusEndBlock;
// Set the lastRewardBlock as the startBlock
lastRewardBlock = startBlock;
swapContract = _swapContract;
BTCT_ADDR = _btct;
// Transfer ownership to the admin address who becomes owner of the contract
transferOwnership(_admin);
```

Also, as the contract is ownable, there is no need for transferOwnership(_admin) as the address that deploys the contract will be the owner.

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-002: Immutable state variables

Severity: Info

The following variables can be made immutable to save gas

```
IERC20 public rewardToken;
IERC20 public stakedToken;
address public swapContract;
address public BTCT_ADDR;
```

Recommendations

Make the above state variables immutable.

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-003: swapContract can be defined as ISwapContractMin

Severity: Info

Like how the reward and swap token are set as IERC20 instead of address, the swapContract can be defined as ISwapContractMin so that it does not have to be casted each time to be used.

Recommendations

Change from address to ISwapContractMin, and remove the casting in getExpectedPerBlock and _updateRewardPerBlock.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-004: Unnecessary use of SafeMath

Severity: Info

As Solidity 0.8.0 is used, there is no need for SafeMath as integer over/underflow checks are done in normal arithmetic.

Recommendations

Change all safemath to normal arithmetic to reduce the gas costs.

Notes

Acknowledged but no change made.

CLM-005: Unnecessary uint256 casting

Severity: Info

In getExpectedPerBlock

```
if (diff != uint256(0)) {
            diff = uint256(1);
        }
```

Recommendations

Remove the unnecessary casting.

E.g.

```
diff != 0
```

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-006: Division by 0

Severity: Info

diff can be 0, which will result in a revert due to division by 0.

```
rewardPerBlock = rewardPerBlock.add(moved.mul(1e10).div(diff)); // moved ==
decimals 8
```

Recommendations

It appears that there is a logic flaw in the check of diff to set it to 1 if it is 0. To be confirmed what diff should be if 0 or non-zero.

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-007: Unnecessary comparison with true

Severity: Info

isDynamicBTC is bool so there is no need to check against == true.

```
if (reserveBTC >= reserveBTCT && isDynamicBTC == true) {
```

Recommendations

Change to

```
if (reserveBTC >= reserveBTCT && isDynamicBTC) {
```

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-008: getExpectedPerBlock can be set to external

Severity: Info

getExpectedPerBlock is never used internally within the contract, so it can be changed from public to external

Recommendations

Change the visibility of getExpectedPerBlock.

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-009: > 0 can be != 0 if comparing for unsigned integers

Severity: Info

As uint can never be negative, != 0 can be used to save gas.

Recommendations

Change > 0 to != 0 for all non-zero comparisons for uint data types.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1 and https://github.com/SwingbyProtocol/defi-contract/commit/e49ddd4648f8e2a3cdaed4b1c30650a5fe0cfce4

CLM-010: defaultRewardPerBlock.mul(2) can be made as a state variable

Severity: Info

This is used as the max upper limit check for reward per block, so it can be set as a state variable in updateRewardPerBlock and constructor.

Recommendations

Declare maxRewardPerBlock as a state variable, and update it in the constructor and updateRewardPerBlock. This state variable can be used in place of defaultRewardPerBlock.mul(2).

```
uint256 maxRewardPerBlock;
[...]
function updateRewardPerBlock(uint256 _rewardPerBlock) external onlyOwner {
        require(block.number < startBlock, "Pool has started");
        rewardPerBlock = _rewardPerBlock;
        defaultRewardPerBlock = rewardPerBlock;
        maxRewardPerBlock = rewardPerBlock * 2;
        emit NewRewardPerBlock(_rewardPerBlock);
    }
```

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/7e3c2e54f56176eb4720d3dcf7c29159dda796f1

CLM-011: getExpectedPerBlock should be a view

Severity: Info

It appears that getExpectedPerBlock does not change any state.

Recommendations
Add view to getExpectedPerBlock.

Notes
Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/2a9e2670b3e15cf4060eb5bc82fc2937f97ae00b

---

CLM-012: _updateRewardPerBlock is done before accTokenPerShare is updated

Severity: Medium

In updatePool, the _updateRewardPerBlock is called before accTokenPerShare is updated. This results in the calculation of the update of accTokenPerShare using the post-updated rewardPerBlock, rather than the current rewardPerBlock.

Recommendations
_updateRewardPerBlock should be called after accTokenPerShare is updated.

Notes

Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/837cd1ee8afdb5e2c3d3ca6494e7ca705b8b96d1

CLM-013: updatePool is not done before rewardPerBlock is updated in updateRewardPerBlock

Severity: Medium

When the owner sets the new rewardPerBlock in updateRewardPerBlock, updatePool is not called. This would result in the rewards since the last time updatePool was called not being calculated with the previous rewardPerBlock.

Recommendations
updatePool should be called before the new rewardPerBlock is set.

Notes
Resolved in
https://github.com/SwingbyProtocol/defi-contract/commit/1899ef6269c349e3b8c5b71372d630acc878e25e

---

CLM-014: Token balances are using the actual token balance of the contract instead of cached balances

Severity: Info

In updatePool, the lpSupply, which is used as the denominator for calculating the swingbyReward to be added to the current accTokenPerShare of a pool, is using the balanceOf function of the token to obtain the balances.

In some cases where the balances of tokens could include token balances besides the amount deposited by users (e.g. reward tokens are also stored in the same MasterChef contract), this can affect the calculation of the rewards for users.

Recommendations

Like PCS LP Staking, use a cached token balance for each pool.

This cache balance should be incremented with the deposited amount in `deposit`, and decremented by the withdraw amount in both `withdraw` and `emergencyWithdraw`.

Notes

Resolved in https://github.com/SwingbyProtocol/defi-contract/commit/e49ddd4648f8e2a3cdaed4b1c30650a5fe0cfce4

CLM-015: emergencyRewardWithdraw allows owner to withdraw reward tokens

Severity: Medium

emergencyRewardWithdraw allows the owner of the contract to withdraw all reward tokens. If the owner address has been compromised (e.g. through leakage of private key), it would be possible to drain the entire reward token balance.

Additionally, if the rewards are drained and there are insufficient rewards in the pool for users to harvest, users will be unable to do subsequent deposits and withdrawals. The only way for a user to withdraw would be to use the emergencyWithdraw function, giving up all pending rewards.

Recommendations
The owner of the contract should be a multisig address.

Notes
Acknowledged. Owner of deployed contract is currently still an EOA, but will move to a multisig eventually.

Conclusion

For PCS-LP Staking and SbBTC LP staking are medium and above issues have been resolved, with the exception of CLM-015, which the Swingby team has stated will be resolved in the near future. Most of the low and info issues have also been resolved.

For Swingby UniswapV2 & SushiSwap LP Staking, as the contract was already deployed and in use, the Swingby team has decided to acknowledge the issues, but leave them as unresolved as resolving would require a redeployment of the contract.

Overall, the team was quick to respond to security issues raised and have implemented the security recommendations mentioned in the report.