QuillAudits

Audit Report
November, 2022

For

DARK POOL

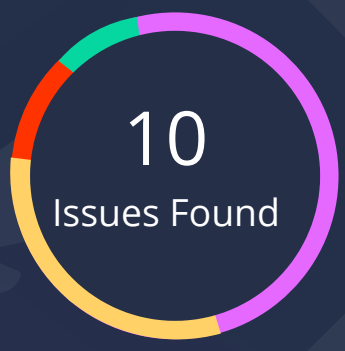
Table of Content

Table of Content

Executive Summary

Project Name	SuperTraffic
Overview	The Darkpool contract allows a hold and unhold functionality, basically to keep and release tokens in contract to either party when the deal is over.
Timeline	10 October, 2022 to 29 November, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze SuperTraffic codebase for quality, security, and correctness. *https://bitbucket.org/octopoli/dp_contract/src/dp-stage/* Branch Name: dp-stage Commit hash: 2c50b33
Fixed In	https://bitbucket.org/octopoli/dp_contract/src/dp-stage/contracts/DarkPool.sol Branch Name: dp-stage Commit Hash: d264216

Note: Extra Review has been carried out for new changes
Extra Review Audit Scope: *https://bitbucket.org/octopoli/dp_contract/commits/9980e02a3e1a5299595945f277dbeac29036d161*
Extra Review Date: 11 January, 2023 to 2nd February, 2023
Extra Review overview: For gas savings, this commit changes settleDeal signature and parameters to be passed in using indexes instead of MpKeys.



High Medium Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	0	1
Partially Resolved Issues	0	0	0	1
Resolved Issues	1	2	1	3

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - DarkPool.sol

High Severity Issues

A.1: Initialization issues

```
function initialize(address superadmin) public initializer {    infinite gas
    __AccessControl_init();
    require(superadmin != address(0), "Cannot set to zero address");
    _spr_admin = superadmin;
    _admin = superadmin;
     sys_fee = superadmin;
```

Description

The contract can be initialized with a zero address, doing this sets the superAdmin address to the dead address and renders the contract unusable.

Remediation

Include input validation/parity checks for initialized values in constructor to completely avoid mistakes and revert if major issues will be caused from the inputs given

Status

Resolved

Medium Severity Issues

A.2: Lack of input validation

Description

The contract can have multiple setter functions vulnerable to malicious input. There are no input validation checks and parity checks especially for critical functions such as:
setUSDTAddr
setSuperAdmin
setAdmin
setSystemFee
setExpTme
setBalance

Incorrect inputs here can lead to griefing. A wrong USDTAddr contract will render the withdraw function useless, a wrong superAdmin set could completely render the contract useless. Also, the new values to be set could be the same as the current values and so would use up all the gas per call, a require check for equality could break out of the function call early enough. The setBalance functions permits for the zero address to be passed in successfully which could alter token balances/allocations based on the dp_gateway which is out of scope.

Remediation

Include input validation in the functions as will be needed to reduce the surface of input validation risks. Use require checks as well.

Status

Resolved

A.3: Centralization issues

Description

The contract has the most functionality packed into one address, the superAdmin address. It controls the setup of new admins and the account to which fees go. The USDTAddr can also be set this way to a malicious contract designed to siphon funds. It provides a very wide surface as an attack vector if left unchecked

Remediation

We advise the client to carefully manage the superAdmin account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism (such as governance) or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets

Status

Acknowledged

SuperTraffic team removed the setUSDTAddr function and acknowledged the risk of centralization.

A.4: Consider using the SafeERC20 wrapper for USDT token

Description

Some ERC20 tokens have no return values in their transfer and transferFrom function. This could lead to unexpected functionality. The USDT token called here is IBEP40 and not IBEP20.

Remediation

Use the SafeERC20 wrapper to make the ‘safeTransfer’ function available, and

Status

Resolved

Low Severity Issues

A.5: Hold balance calculations

```
function initialize(address superadmin) public initializer {    infinite gas
    __AccessControl_init();
    require(superadmin != address(0), "Cannot set to zero address");
    _spr_admin = superadmin;
    _admin = superadmin;
     sys fee = superadmin;
```

Description

The require check here only takes the amount into consideration and not the fee. If a user has just enough for holding balance calculations but not for the fee it'll pass the require checks. It will cause an unhandled underflow.

Remediation

Include input validation/parity checks for initialized values in constructor to completely avoid mistakes and revert if major issues will be caused from the inputs given

Status

Resolved

Informational Issues

A.6: Renaming internal functions to fit best practice

Description

The require check here only takes the amount into consideration and not the fee. If a user has just enough for holding balance calculations but not for the fee it'll pass the require checks. It will cause an unhandled underflow.

```
function isUsrExstInt(bytes32 _id) internal pure returns(bool) {
function setBalInt(address user, uint amount) internal returns(bool) {
function sendUSDT(address to, uint256 amount) internal returns(bool){
function NtfWin(User memory user, string memory deal_id) internal {
function HldBal(address addr, uint amount, uint sf, string memory di) internal {
function UnHldBal(User memory usr, string memory di) internal {
```

Remediation

Rename all functions with visibility internal to improve code readability and align with best practice.

Status

Resolved

A.7: Rename variables and functions to match what they do

Description

It is advisable to name variables and functions according to what they do. This will improve readability and codebase understanding without needing to meet with the team.
The following functions/variables can be renamed:
getFeeBal returns the balance of the _sys_fee address, not a fee balance
expTme returns a time delta used in calculations with Deal struct objects, not a preset expiry time
setSystemFee sets the address for fees to be sent to, not the system fee

Remediation

Consider renaming the functions and abbreviated variables as well.

Status

Resolved

Informational Issues

A.8: Unused contract definitions

Description

Some contract defined variables are never used once declared. Consider removing them if they will not be implemented to reduce slots used and gas costs. The following can be removed:
Error InvalidUser (address user)
string[ ] public expDlsMpKs

Remediation

Remove the unused contract items or implement the desired functionality with them.

Status

Partially Resolved

SuperTraffic Team's Remark - for expDlsMpKs, "deprecated, do not remove !!!"

A.9: Unindexed events

Description

Indexing events helps for sorting and to make searches on the blockchain more efficient because of logs. The following events are not indexed:

```
event DealSettled(address from, string deal_id, uint block);
event DealCreated(address from,string deal_id,uint exptime,uint block);
event Withdrawal(address account, uint amount, uint block);
event Deposit(address account, uint amount, uint block);
event Hold(address account, uint amount, uint block, string deal_id);
event UnHold(address account, uint amount, uint block, string deal_id);
event Profit(address account, uint amount, uint block, string deal_id);
event Lose(address account, uint amount, uint block, string deal_id);
event Fee(address account, uint amount, uint block, string deal_id);
event SrvFee(address account, uint amount, uint block, string deal_id);
```

Remediation

To aid searches through logs, consider indexing arguments used in the event definitions

Status

Resolved

A.10: Usage of expensive data type (strings)

Description

Strings are used in various portions of the contract and strings are highly expensive to save and manipulate.

Remediation

It would be advisable to use a less costly data type, e.g. bytes32, to save gas costs.

Status

Acknowledged

SuperTraffic Team's Remark - suggestion about type was right, we can safe up to 40% of gas when use byte32 instead strings, but it is big move, we will plan to do it later, after launch in production and our first deals

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
IERC20 is re-used:
        - IERC20 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#9-82)
        - IERC20 (contracts/DarkPool.sol#11-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

DarkPool.sendUSDT(address,uint256) (contracts/DarkPool.sol#215-219) ignores return value by usdt.transfer(to,amount) (contracts/DarkPool.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Reentrancy in DarkPool.withdraw(uint256) (contracts/DarkPool.sol#226-237):
        External calls:
        - success = sendUSDT(msg.sender,amount) (contracts/DarkPool.sol#231)
                - usdt.transfer(to,amount) (contracts/DarkPool.sol#217)
        State variables written after the call(s):
        - usrActBal[_id] = usrActBal[_id] - amount (contracts/DarkPool.sol#233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

DarkPool.getMpKeys(uint256).mk (contracts/DarkPool.sol#335) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

```
DarkPool.initialize(address).superadmin (contracts/DarkPool.sol#138) lacks a zero-check on :
                - _spr_admin = superadmin (contracts/DarkPool.sol#140)
                - _admin = superadmin (contracts/DarkPool.sol#141)
                - _sys_fee = superadmin (contracts/DarkPool.sol#142)
DarkPool.setSuperAdmin(address).newSuperAdmin (contracts/DarkPool.sol#160) lacks a zero-check on :
                - _spr_admin = newSuperAdmin (contracts/DarkPool.sol#163)
DarkPool.setAdmin(address).newAdmin (contracts/DarkPool.sol#167) lacks a zero-check on :
                - _admin = newAdmin (contracts/DarkPool.sol#170)
DarkPool.setUSDTAddr(address).usdtContract (contracts/DarkPool.sol#221) lacks a zero-check on :
                - USDT_ADDR = usdtContract (contracts/DarkPool.sol#222)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in DarkPool.withdraw(uint256) (contracts/DarkPool.sol#226-237):
        External calls:
        - success = sendUSDT(msg.sender,amount) (contracts/DarkPool.sol#231)
                - usdt.transfer(to,amount) (contracts/DarkPool.sol#217)
        Event emitted after the call(s):
        - Withdrawal(msg.sender,amount,block.number) (contracts/DarkPool.sol#234)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

DarkPool.HldBal(address,uint256,uint256,string) (contracts/DarkPool.sol#258-267) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((usrActBal[id] >= amount),amnt exd) (contracts/DarkPool.sol#261)
DarkPool.UnHldBal(DarkPool.User,string) (contracts/DarkPool.sol#269-278) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((usrHldBal[id] >= usr.hld),amnt exd) (contracts/DarkPool.sol#272)
DarkPool.CreateDeal(address,address,uint256,DarkPool.Deal) (contracts/DarkPool.sol#280-303) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(deal.rRsk1 + sfee <= usrActBal[deal.usr1],rR1 gt usr bal) (contracts/DarkPool.sol#288)
        - require(bool,string)(deal.rRsk2 + sfee <= usrActBal[deal.usr2],rR2 gt usr bal) (contracts/DarkPool.sol#289)
        - require(bool,string)(deal.tWn2 <= deal.rRsk1,tW2 gt rR1) (contracts/DarkPool.sol#290)
        - require(bool,string)(deal.tWn1 <= deal.rRsk2,tW1 gt rR2) (contracts/DarkPool.sol#291)
        - require(bool,string)(dlsMp[deal.deal_id].exptme == 0,deal exst) (contracts/DarkPool.sol#292)
DarkPool.SettleDeal(DarkPool.Deal,DarkPool.Settle,DarkPool.MpKeys) (contracts/DarkPool.sol#305-332) uses timestamp for comparisons
```

```
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) u
assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
console._sendLogPayload(bytes) (node_modules/hardhat/console.sol#7-14) uses assembly
        - INLINE ASM (node_modules/hardhat/console.sol#10-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['>=0.4.22<0.9.0', '^0.8.0', '^0.8.1']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (contracts/DarkPool.sol#2)
        - ^0.8.0 (contracts/USDT.sol#2)
        - >=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions
```

```
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65
):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/Add
ressUpgradeable.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgrade
able.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52
) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable
.sol#54-55) is not in mixedCase
Variable AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#235) is not in mixedCa
se
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCa
se
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not
in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is not
 in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-
28) is not in mixedCase
Variable ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#41) is not in mixedCase
Parameter DarkPool.isUsrExstInt(bytes32)._id (contracts/DarkPool.sol#189) is not in mixedCase
Function DarkPool.NtfWin(DarkPool.User,string) (contracts/DarkPool.sol#250-256) is not in mixedCase
Parameter DarkPool.NtfWin(DarkPool.User,string).deal_id (contracts/DarkPool.sol#250) is not in mixedCase
Function DarkPool.HldBal(address,uint256,uint256,string) (contracts/DarkPool.sol#258-267) is not in mixedCase
Function DarkPool.UnHldBal(DarkPool.User,string) (contracts/DarkPool.sol#269-278) is not in mixedCase
Function DarkPool.CreateDeal(address,address,uint256,DarkPool.Deal) (contracts/DarkPool.sol#280-303) is not in mixedCase
Function DarkPool.SettleDeal(DarkPool.Deal,DarkPool.Settle,DarkPool.MpKeys) (contracts/DarkPool.sol#305-332) is not in mixedCase
```

Important Note to User

The protocol uses a go-application (dp-gateway) to handle some functions internally. dp-gateway is out of the scope of this audit and functions interlinked [withdraw(), setBalance()] with this were not fully encompassed in this audit.

Important Note to User

The protocol uses a go-application (dp-gateway) to handle some functions internally. dp-gateway is out of the scope of this audit and functions interlinked [withdraw(), setBalance()] with this were not fully encompassed in this audit.

Some Key Questions:

Can a smart contract owner or admin steal money from the clients, who made deposits to it?

- The withdraw function is callable by any user and funds get sent directly to the caller with no possibility of mismanaged transfers to a wrongly specified address.

- There exists a scenario of possible exploit, via the linked address. A linked address will have the same balance as the main address and can be used to place withdrawals without limits.

- Addresses can be linked using the linkAdr() function that can only be called by the admin. If the admin is compromised or malicious, there exists a high chance to access other users' funds if this function is called.

- The team has acknowledged this risk and responded thus"Yes, the only possible exploit is to link the scam address to the existing normal address, but to do that, the scammer should break our software that integrates with the smart contract, where we call all functions, and get access (private keys) to the admin account. But it is a question to another company, who is doing an audit and pentest of the rest of our software

Can a third-party person steal money from the smart contract?

- A third party without admin privileges and not passed in to be linked by a malicious admin should not constitute risk to theft of funds.

- Carefully manage admin accounts and ensure that adequate checks happen for addresses to be linked using the dp-gateway.

Closing Summary

In this report, we have considered the security of the SuperTraffic codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the SuperTraffic Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the SuperTraffic Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$15B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2022

For

DARK POOL

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com