SMART CONTRACT AUDIT REPORT

for

HYDRO FOUNDATION

Prepared By: Shuxiao Wang

Hangzhou, China

Aug 21, 2019

Document Properties

Client	Hydro Foundation
Title	Smart Contract Audit Report
Target	Hydro Protocol v2.0
Version	1.0
Author	Dr. Chiachih Wu
Auditors	Dr. Chiachih Wu, Dr. Xuxian Jiang
Reviewed by	Dr. Chiachih Wu
Approved by	Dr. Chiachih Wu
Classification	Confidential

Version Info

Version	Date	Author	Description
1.0	Aug 21, 2019	Dr. Chiachih Wu	Final Report
0.3	Aug 8, 2019	Dr. Chiachih Wu	More Findings Added
0.2	Aug 2, 2019	Dr. Chiachih Wu	Findings Added
0.1	Jul 31, 2019	Dr. Chiachih Wu	Initial Draft

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the **Hydro Protocol v2.0** design document and related smart contract source code, we in the report outline our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the white paper, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Hydro Protocol v2.0

Hydro [3] is an open source framework for building high performance decentralized exchanges. The basic information of Hydro is as follows:

Table 1.1: Basic Information of Hydro Protocol v2.0

Item	Description
Issuer	Hydro Foundation
Website	https://hydroprotocol.io
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Audit Completion Date	Aug 21, 2019

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit:

- https://github.com/HydroProtocol/protocol.git
- commit: f1245df

1.2 About PeckShield

PeckShield Inc. [10] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [5]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as an investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Short Address Bug
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

2 Findings

2.1 Summary

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	2
Informational	8
Total	10

2.2 Key Findings

Overall, the smart contract implementation can be improved because of the existence of 10 issues, including 2 low severity vulnerability and 8 informational recommendations, as shown in Table 2.1.

Table 2.1: Key Audit Findings

ID	Severity	Title	Type	Status
PVE-001	Low	Unchecked Return Value	Vulnerability	Fixed
PVE-002	Informational	Lack of Emitting Events	Recommendation	Fixed
PVE-003	Informational	Performance Enhancement	Recommendation	Fixed
PVE-004	Informational	Gas Consumption Enhancement	Recommendation	Confirmed
PVE-005	Informational	Abusing Non-exist auctionID	Recommendation	Fixed
PVE-006	Informational	Lack of Kill-Switch Implementation	Recommendation	Confirmed
PVE-007	Low	Approve/TransferFrom Race Condition	Vulnerability	Confirmed
PVE-008	Informational	Lack of borrowInterestRate Sanity Check	Recommendation	Fixed
PVE-009	Informational	Lack of Token Parameter Initialization	Recommendation	Fixed
PVE-010	Informational	Inconsistent Function Declaration	Recommendation	Confirmed

Please refer to Chapter 3 for details.

3 Detailed Results

3.1 Unchecked Return Value

- ID: PVE-001
- Severity: Low
- Description: The return value of calling `makerDaoOracle.peek()` is not checked. In the unlikely case, suppose makerDao's oracle experiences an unknown problem that leads to irrational price feeds, Hydro will cascadingly be affected without any protection.
- Details: oracle/DaiPriceOracle.sol:121

```
121 function getMakerDaoPrice()
122     public
123     view
124     returns (uint256)
125 {
126     (bytes32 value, ) = makerDaoOracle.peek();
127     return uint256(value);
128 }
```

Listing 3.1: oracle/DaiPriceOracle.sol

oracle/EthPriceOracle.sol:30

```
30 function getPrice(
31     address _asset
32 )
33     external
34     view
35     returns (uint256)
36 {
37     require(_asset == address(0), "ASSET_NOT_MATCH");
38     (bytes32 value, ) = makerDaoOracle.peek();
39     return uint256(value);
40 }
```

Listing 3.2: oracle/EthPriceOracle.sol

- Recommendation: Validate the values returned by `makerDaoOracle.peek()`.

3.2 Lack of Emitting Events

- ID: PVE-002
- Severity: Informational
- Description: For better book-keeping or offline analytics purposes, it is suggested to emit certain events whenever `borrowIndex` or `supplyIndex` are updated or user funds are transferred.
- Details: lib/Transfer.sol:93

```
93  function transfer(
94    Store.State storage state,
95    address asset,
96    Types.BalancePath memory fromBalancePath,
97    Types.BalancePath memory toBalancePath,
98    uint256 amount
99  )
100   internal
101 {
102   if (toBalancePath.category == Types.BalanceCategory.CollateralAccount) {
103     Requires.requireMarketIDAndAssetMatch(state, toBalancePath.marketID, asset);
104   }
105
106   mapping(address => uint256) storage fromBalances = fromBalancePath.getBalances(
        state);
107   mapping(address => uint256) storage toBalances = toBalancePath.getBalances(state)
        ;
108
109   require(fromBalances[asset] >= amount, "TRANSFER_BALANCE_NOT_ENOUGH");
110
111   fromBalances[asset] = fromBalances[asset] - amount;
112   toBalances[asset] = toBalances[asset].add(amount);
113 }
```

Listing 3.3: lib/Transfer.sol

funding/LendingPool.sol:348

```
348 function updateIndex(
349   Store.State storage state,
350   address asset
351 )
352   private
353 {
354   (uint256 currentSupplyIndex, uint256 currentBorrowIndex) = getCurrentIndex(state,
        asset);
355
356   // get the total equity value
```

```
    uint256 normalizedBorrow = state.pool.normalizedTotalBorrow[asset];
    uint256 normalizedSupply = getTotalNormalizedSupply(state, asset);

    // interest = equity value * (current index value - starting index value)
    uint256 recentBorrowInterest = Decimal.mulCeil(
      normalizedBorrow,
      currentBorrowIndex.sub(state.pool.borrowIndex[asset])
    );

    uint256 recentSupplyInterest = Decimal.mulFloor(
      normalizedSupply,
      currentSupplyIndex.sub(state.pool.supplyIndex[asset])
    );

    // the interest rate spread goes into the insurance pool
    state.pool.insuranceBalances[asset] = state.pool.insuranceBalances[asset].add(
        recentBorrowInterest.sub(recentSupplyInterest));

    // update the indexes
    state.pool.supplyIndex[asset] = currentSupplyIndex;
    state.pool.borrowIndex[asset] = currentBorrowIndex;
    state.pool.indexStartTime[asset] = block.timestamp;
}
```

Listing 3.4: funding/LendingPool.sol

- Recommendation: Emit corresponding events.

3.3 Performance Enhancement

- ID: PVE-003
- Severity: Informational
- Description: When an auction is finished, the `endAuction()` function will look up the auction in the internal `currentAuctions` array and then remove it. The loop up process can be terminated earlier to save gas.
- Details: funding/Auctions.sol:304

```
304 function endAuction(
305     Store.State storage state,
306     Types.Auction storage auction
307 )
308     private
309 {
310     auction.status = Types.AuctionStatus.Finished;
311 
312     state.accounts[auction.borrower][auction.marketID].status = Types.
        CollateralAccountStatus.Normal;
```

```
    for (uint i = 0; i < state.auction.currentAuctions.length; i++) {
      if (state.auction.currentAuctions[i] == auction.id) {
          state.auction.currentAuctions[i] = state.auction.currentAuctions[state.
              auction.currentAuctions.length -1];
          state.auction.currentAuctions.length--;
      }
    }
}
```

Listing 3.5: funding/Auctions.sol

- Recommendation: Exit the loop when `auction.id` is found.

3.4 Gas Consumption Enhancement

- ID: PVE-004
- Severity: Informational
- Description: Related to PVE-006, the `endAuction()` function does not delete or release the stale auction storage. However, by releasing unused storage spaces, Ethereum protocol is designed to refund some gas [12].
- Details: funding/Auctions.sol:304

```
304 function endAuction(
305     Store.State storage state,
306     Types.Auction storage auction
307 )
308     private
309 {
310     auction.status = Types.AuctionStatus.Finished;
311 
312     state.accounts[auction.borrower][auction.marketID].status = Types.
            CollateralAccountStatus.Normal;
313 
314     for (uint i = 0; i < state.auction.currentAuctions.length; i++) {
315         if (state.auction.currentAuctions[i] == auction.id) {
316             state.auction.currentAuctions[i] = state.auction.currentAuctions[state.
                    auction.currentAuctions.length -1];
317             state.auction.currentAuctions.length--;
318         }
319     }
320 }
```

Listing 3.6: funding/Auctions.sol

- Recommendation: Delete stale auction storage.

3.5 Abusing Non-exist auctionID

- ID: PVE-005
- Severity: Informational
- Description: Related to PVE-006, the exported function `fillAuctionWithAmount()` does not validate the `auctionID` and can be leveraged to retrieve an uninitialized auction from the storage. In a later commit, a sanity check function `requireAuctionNotFinished()` is implemented. However, it can still be improved as both the `status` of an uninitialized auction and `Types.AuctionStatus.InProgress` are 0.

  Inside `fillAuctionWithAmount()`, `leftDebtAmount` is calculated for the uninitialized auction, which equals 0 in this case. As a result, `endAuction()` is invoked, which leads to data corruption on `state.accounts[0][0]` and `state.auction.currentAuctions` (the very first slot with `auction.id`=0).

  Fortunately, the SafeMath division in used in both `fillHealthyAuction()` and `fillBadAuction()` accidently prevents this vulnerability from being exploited. However, we still strongly recommend to validate auctionID, in addition to checking the auction status, before entering `fillAuctionWithAmount()`.

- Details: ExternalFunctions.sol:159

```
function fillAuctionWithAmount(
    uint32 auctionID,
    uint256 amount
)
    external
{
    Auctions.fillAuctionWithAmount(state, auctionID, amount);
}
```

Listing 3.7: ExternalFunctions.sol

funding/Auctions.sol:266

```
function fillAuctionWithAmount(
    Store.State storage state,
    uint32 auctionID,
    uint256 repayAmount
)
    external
{
    Types.Auction storage auction = state.auction.auctions[auctionID];
    uint256 ratio = auction.ratio(state);

    uint256 actualRepayAmount;
```

```
    uint256 collateralForBidder;

    if (ratio <= Decimal.one()) {
        (actualRepayAmount, collateralForBidder) = fillHealthyAuction(state,
            auction, ratio, repayAmount);
    } else {
        (actualRepayAmount, collateralForBidder) = fillBadAuction(state, auction,
            ratio, repayAmount);
    }

    // reset account state if all debts are paid
    uint256 leftDebtAmount = LendingPool.getAmountBorrowed(
        state,
        auction.debtAsset,
        auction.borrower,
        auction.marketID
    );

    Events.logFillAuction(auction.id, msg.sender, actualRepayAmount,
        collateralForBidder, leftDebtAmount);

    if (leftDebtAmount == 0) {
        endAuction(state, auction);
    }
}
```

Listing 3.8: funding/Auctions.sol

funding/Auctions.sol:304

```
function endAuction(
    Store.State storage state,
    Types.Auction storage auction
)
    private
{
    auction.status = Types.AuctionStatus.Finished;

    state.accounts[auction.borrower][auction.marketID].status = Types.
        CollateralAccountStatus.Normal;

    for (uint i = 0; i < state.auction.currentAuctions.length; i++) {
        if (state.auction.currentAuctions[i] == auction.id) {
            state.auction.currentAuctions[i] = state.auction.currentAuctions[state.
                auction.currentAuctions.length-1];
            state.auction.currentAuctions.length--;
        }
    }
}
```

Listing 3.9: funding/Auctions.sol

- Recommendation: Validate the existence of `auctionID`.

3.6 Lack of Kill-Switch Implementation

- ID: PVE-006
- Severity: Informational
- Description: A kill-switch or a similar mechanism that allows the admin to temporarily pause the system can help the development team to better handle security accidents if any. In many security response cases, the admin can leverage the kill-switch mechanism to pause the system operations, take a snapshot of user accounts or funds, and then migrate the states to a newly deployed system.
- Details: Not Implemented
- Recommendation: Add kill switch mechanism into the design and implementation. One example of implementation will be adding a `require(live == 1)` check when entering various critical functions. Certainly, there exists an admin-only interface for setting the `live` variable.

3.7 Approve/TransferFrom Race Condition

- ID: PVE-007
- Severity: Low
- Description: The ERC20 standard defines a number of required interfaces of generic token contracts. In particular, `approve(spender, value)` sets the allowance a spender can spend on the user's tokens via `transferFrom()`. The current implementation is vulnerable to a known race condition. Specifically, when observing a pending transaction with the `approve()` call, the spender immediately can call `transferFrom()` to transfer the previously allowed value and still receive new allowance (thanks to new `approve()` call) for next transfer [4]. Note that the spender needs to get `transferFrom()` mined before the observed `approve()` call.
- Result: helper/StandardToken.sol:71

```
71 function transferFrom(
72     address from,
73     address to,
74     uint256 amount
75 )
76     public
77     returns (bool)
78 {
79     require(to != address(0), "TO_ADDRESS_IS_EMPTY");
80     require(amount <= balances[from], "BALANCE_NOT_ENOUGH");
```

```
    require(amount <= allowed[from][msg.sender], "ALLOWANCE_NOT_ENOUGH");

    balances[from] = balances[from].sub(amount);
    balances[to] = balances[to].add(amount);
    allowed[from][msg.sender] = allowed[from][msg.sender].sub(amount);
    emit Transfer(from, to, amount);
    return true;
}
```

Listing 3.10: helper/StandardToken.sol

- Recommendation: Add correct sanity checks as suggested in [4].

3.8 Lack of borrowInterestRate Sanity Check

- ID: PVE-008
- Severity: Informational
- Description: It is suggested to ensure the externally-influenced `borrowInterestRate` risk parameter always stays within the allowed range. Otherwise, the risk of depending on external admin operation might go unchecked.
- Result: funding/LendingPool.sol:299

```
function updateInterestRate(
    Store.State storage state,
    address asset
)
    private
{
    (uint256 borrowInterestRate, uint256 supplyInterestRate) = getInterestRates(
        state, asset, 0);
    state.pool.borrowAnnualInterestRate[asset] = borrowInterestRate;
    state.pool.supplyAnnualInterestRate[asset] = supplyInterestRate;
}
```

Listing 3.11: funding/LendingPool.sol

- Recommendation: Ensure the `borrowInterestRate` risk parameter stays in a normal range, instead of leaving the monitoring task to the admin.

3.9 Lack of Token Parameter Initialization

- ID: PVE-009
- Severity: Informational

- Description: The constructor of `LendingPoolToken()` does not take `totalSupply` as an input parameter but references it for initializing the owner's balance.

- Result: funding/LendingPoolToken.sol:39

```
constructor (
    string memory tokenName,
    string memory tokenSymbol,
    uint8 tokenDecimals
)
    public
{
    name = tokenName;
    symbol = tokenSymbol;
    decimals = tokenDecimals;
    balances[msg.sender] = totalSupply;
}
```

Listing 3.12: funding/LendingPoolToken.sol

- Recommendation: Pass `totalSupply` into the constructor and initialize it.

3.10 Inconsistent Function Declaration

- ID: PVE-010

- Severity: Informational

- Description: The `createMarket()` function does not need to be `public` as it is designed to be externally called. Also, for consistency, it is suggested to change from `public` to `external`.

- Result: components/OperationsComponent.sol

```
function createMarket(
    Store.State storage state,
    Types.Market memory market
)
    public
{
    Requires.requireMarketAssetsValid(state, market);
    Requires.requireMarketNotExist(state, market);
    Requires.requireDecimalLessOrEquanThanOne(market.auctionRatioStart);
    Requires.requireDecimalLessOrEquanThanOne(market.auctionRatioPerBlock);
    Requires.requireDecimalGreaterThanOne(market.liquidateRate);
    Requires.requireDecimalGreaterThanOne(market.withdrawRate);

    state.markets[state.marketsCount++] = market;
    Events.logCreateMarket(market);
}
```

Listing 3.13: components/OperationsComponent.sol

- Recommendation: Modify the function declaration from `public` to `external`.

4 Conclusion

In this audit, we have analyzed the Hydro smart contact implementation. During our auditing process, we are constantly impressed by the thinkings behind the Hydro protocol. It is indeed a rather complex system with various functionalities (e.g., margin trading, lending, and auctions), but the entire protocol is cleanly designed and engineered. The related smart contracts are also neatly organized and elegantly implemented. Those identified issues are promptly confirmed and fixed. Meanwhile, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions.

5 Appendix

5.1 Basic Coding Bugs

5.1.1 Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

5.1.2 Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

5.1.3 Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

5.1.4 Overflows & Underflows

- Description: Whether the contract has general overflow or underflow vulnerabilities [6, 7, 8, 9, 11].
- Result: Not found
- Severity: Critical

5.1.5 Reentrancy

- Description: Reentrancy [13] is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: Not found
- Severity: Critical

5.1.6 Money-Giving Bug

- Description: Whether the contract returns funds to an arbitrary address.
- Result: Not found
- Severity: High

5.1.7 Blackhole

- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: Not found
- Severity: High

5.1.8 Short Address Bug

- Description: Whether the contract checks the length of the address argument [2].
- Result: Not found
- Severity: Medium

5.1.9 Unauthorized Self-Destruct

- Description: Whether the contract can be killed by any arbitrary address.
- Result: Not found
- Severity: Medium

5.1.10 Revert DoS

- Description: Whether the contract is vulnerable to DoS attack because of unexpected `revert`.
- Result: Not found
- Severity: Medium

5.1.11 Unchecked External `Call`

- Description: Whether the contract has any external `call` without checking the return value.
- Result: Not found
- Severity: Medium

5.1.12 Gasless `Send`

- Description: Whether the contract is vulnerable to gasless `send`.
- Result: Not found
- Severity: Medium

5.1.13 `Send` Instead of `Transfer`

- Description: Whether the contract uses `send` instead of `transfer`.
- Result: Not found
- Severity: Medium

5.1.14 Costly Loop

- Description: Whether the contract has any costly loop which may lead to `Out-Of-Gas` exception.
- Result: Not found
- Severity: Medium

5.1.15 (Unsafe) Use of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: Not found
- Severity: Medium

5.1.16 (Unsafe) Use of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: Not found
- Severity: Medium

5.1.17 Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

5.1.18 Deprecated Uses

- Description: Whether the contract use the deprecated `tx.origin` to perform the authorization.
- Result: Not found
- Severity: Medium

5.2 Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

5.3 Additional Recommendations

5.3.1 Avoid Use of Variadic Byte Array

- Description: Use fixed-size byte array is better than that of `byte[]`, as the latter is a waste of space.
- Result: Not found
- Severity: Low

5.3.2 Use Fixed Compiler Version

- Description: Use fixed compiler version is better.
- Result: Not found
- Severity: Low

5.3.3 Make Visibility Level Explicit

- Description: Assign explicit visibility specifiers for functions and state variables.
- Result: Not found
- Severity: Low

5.3.4 Make Type Inference Explicit

- Description: Do not use keyword `var` to specify the type, i.e., it asks the compiler to deduce the type, which is not safe especially in a loop.
- Result: Not found
- Severity: Low

5.3.5 Adhere To Function Declaration Strictly

- Description: Solidity compiler (version 0.4.23) enforces strict ABI length checks for return data from `calls()` [1], which may break the the execution if the function implementation does NOT follow its declaration (e.g., no return in implementing `transfer()` of ERC20 tokens).
- Result: Not found
- Severity: Low

References

[1] axic. Enforcing ABI length checks for return data from calls can be breaking. https://github.com/ethereum/solidity/issues/4116.

[2] CoinFabrik Blog. Smart Contract Short Address Attack Mitigation Failure. https://blog.coinfabrik.com/smart-contract-short-address-attack-mitigation-failure/.

[3] Hydro Foundation. Hydro Protocol. https://hydroprotocol.io/.

[4] HaleTom. Resolution on the EIP20 API Approve / TransferFrom multiple withdrawal attack. https://github.com/ethereum/EIPs/issues/738.

[5] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[6] PeckShield. ALERT: New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10299). https://www.peckshield.com/2018/04/22/batchOverflow/.

[7] PeckShield. New burnOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-11239). https://www.peckshield.com/2018/05/18/burnOverflow/.

[8] PeckShield. New multiOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-10706). https://www.peckshield.com/2018/05/10/multiOverflow/.

[9] PeckShield. New proxyOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10376). https://www.peckshield.com/2018/04/25/proxyOverflow/.

[10] PeckShield. PeckShield Inc. https://www.peckshield.com.

[11] PeckShield. Your Tokens Are Mine: A Suspicious Scam Token in A Top Exchange. https://www.peckshield.com/2018/04/28/transferFlaw/.

[12] StackExchange Q&A. Is refunded gas for "freed" storage given to the contract, the "allocator", or the "deleter"? https://ethereum.stackexchange.com/questions/32419/is-refunded-gas-for-freed-storage-given-to-the-contract-the-allocator-or-t?rq=1.

[13] Solidity. Warnings of Expressions and Control Structures. http://solidity.readthedocs.io/en/develop/control-structures.html.