Tenderize - Livepeer Adapter

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **September 11th, 2023 - September 13th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 4
CONTACTS 4
1 EXECUTIVE OVERVIEW 5
1.1 INTRODUCTION 6
1.2 ASSESSMENT SUMMARY 6
1.3 TEST APPROACH & METHODOLOGY 7
2 RISK METHODOLOGY 8
2.1 EXPLOITABILITY 9
2.2 IMPACT 10
2.3 SEVERITY COEFFICIENT 12
2.4 SCOPE 14
3 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 15
4 FINDINGS & TECH DETAILS 16
4.1 (HAL-01) CLAIM FEES STUCK IN CONTRACT - CRITICAL(10) 18
Description 18
Code Location 18
BVSS 19
Recommendation 19
Remediation Plan 19
4.2 (HAL-02) LACK OF SLIPPAGE PROTECTION IN SWAPS - HIGH(8.1) 20
Description 20
Code Location 20
BVSS 21
Recommendation 21

Remediation Plan 21

4.3 (HAL-03) STATICCALL TO UNISWAP V3 DOES NOT WORK - HIGH(8.1) 22

Description 22

Proof of Concept 22

Code Location 23

BVSS 24

Recommendation 24

Remediation Plan 24

4.4 (HAL-04) APPROVAL NOT REVOKED IF STATICCALL FAILS - LOW(3.1) 25

Description 25

Code Location 25

BVSS 26

Recommendation 26

Remediation Plan 26

4.5 (HAL-05) CONTRACT PAUSE FEATURE MISSING - LOW(2.5) 27

Description 27

Recommendation 27

BVSS 27

Remediation Plan 27

4.6 (HAL-06) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL(0.0) 28

Description 28

BVSS 28

Recommendation 28

	Remediation Plan	28
4.7	(HAL-07) OPEN TO-DO - INFORMATIONAL(0.0)	29
	Description	29
	Code Location	29
	BVSS	29
	Recommendation	29
	Remediation Plan	29
5	AUTOMATED TESTING	30
5.1	STATIC ANALYSIS REPORT	31
	Description	31
	Results	31
5.2	AUTOMATED SECURITY SCAN	32
	Description	32
	Results	32

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	09/12/2023
0.2	Document Updates	09/13/2023
0.3	Draft Review	09/13/2023
0.4	Draft Review	09/14/2023
1.0	Remediation Plan	09/29/2023
1.1	Remediation Plan Review	09/29/2023
1.2	Remediation Plan Review	09/29/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Tenderize engaged Halborn to conduct a security assessment on their smart contracts beginning on September 11th, 2023 and ending on September 13th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided three days for the engagement and assigned a full-time security engineer to verify the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some high security risks that were addressed/partially addressed by the Tenderize team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions. (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment. (Brownie, Anvil, Foundry)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

IN-SCOPE CODE & COMMITS:

- Repository: Tenderize/staking
    - Commit ID: 634c65411caf53d5a4b3350e3e6358b5c531a97b
    - Smart contracts **in scope**:
        - src/adapters/LivepeerAdapter.sol

OUT-OF-SCOPE:

- Third-party libraries and dependencies.
- Economic attacks.

REMEDIATION COMMITS:

- Repository: Tenderize/staking
    - Commit IDs:
        - 4073c1f5b387ef7b4ce16c8e979ace68183e9226
        - ed8d90e0c2cec81fd5acae209225fb8d1128c3a0
        - e311dff2ee5d427840fa1b900365a678b63f08d4
        - 3c2eefa8c6ec35d0e3a19bf0c8f630d91e07c056

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	0	2	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) CLAIM FEES STUCK IN CONTRACT	Critical (10)	SOLVED - 09/29/2023
(HAL-02) LACK OF SLIPPAGE PROTECTION IN SWAPS	High (8.1)	PARTIALLY SOLVED - 09/29/2023
(HAL-03) STATICCALL TO UNISWAP V3 DOES NOT WORK	High (8.1)	SOLVED - 09/29/2023
(HAL-04) APPROVAL NOT REVOKED IF STATICCALL FAILS	Low (3.1)	SOLVED - 09/29/2023
(HAL-05) CONTRACT PAUSE FEATURE MISSING	Low (2.5)	RISK ACCEPTED
(HAL-06) INCOMPLETE NATSPEC DOCUMENTATION	Informational (0.0)	ACKNOWLEDGED
(HAL-07) OPEN TO-DO	Informational (0.0)	SOLVED - 09/29/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) CLAIM FEES STUCK IN CONTRACT - CRITICAL(10)

Description:

The LivepeerAdapter contract uses collected fees during the staking process to stake them back into the protocol through the rebase function, thus increasing its staked value in the time. Internally, the _livepeerClaimFees function is executed.

During the process of claiming yielded fees, the contract withdraws these fees from the protocol in ether. Consequently, in order to re-stake these fees, the contract swaps ether to LPT through Uniswap V3 which implements a WETH/LPT pool. Therefore, the contract deposits ether into WETH contract to get the corresponding amount of ether in WETH tokens to swap these tokens later.

However, since the contract deposits its entire balance into WETH by setting the value argument to address(this).balance, the following calls must use the ERC20.balanceOf function from WETH instead of address.balance if it is required to use the actual WETH balance as argument. Therefore, the following safeApprove argument and ExactInputSingleParams.amountIn value will contain a 0 because right after the aforementioned deposit, the contract's balance will be 0.

Code Location:

Listing 1: src/adapters/LivepeerAdapter.sol (Lines 128,129,137)

```
126     LIVEPEER.withdrawFees(payable(address(this)), pendingFees);
127     // convert fees to WETH
128     WETH.deposit{ value: address(this).balance }();
129     ERC20(address(WETH)).safeApprove(address(UNISWAP_ROUTER),
  ↳ address(this).balance);
130     // Create initial params for swap
131     ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
  ↳ .ExactInputSingleParams({
132         tokenIn: address(WETH),
            tokenOut: address(address(LPT)),
            fee: UNISWAP_POOL_FEE,
            recipient: address(this),
            deadline: block.timestamp,
            amountIn: address(this).balance,
            amountOutMinimum: 0,
            sqrtPriceLimitX96: 0
        });
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:L/Y:C/R:N/S:U (10)

Recommendation:

After making the deposit into WETH contract, instead of checking the account's balance, the ERC20.balanceOf function must be used in order to get the actual amount of ether.

Remediation Plan:

SOLVED: The Tenderize team has solved this issue by tracking the current balance before depositing to WETH contract in the following commits:

- 4073c1f5b387ef7b4ce16c8e979ace68183e9226
- ed8d90e0c2cec81fd5acae209225fb8d1128c3a0

4.2 (HAL-02) LACK OF SLIPPAGE PROTECTION IN SWAPS - HIGH (8.1)

Description:

The _livepeerClaimFees function, in order to swap WETH to LPT, calculates the amountOutMinimum through the execution of the same function that is being used for swapping tokens, but this time, executed as a staticcall. This execution would return the current amount of LPT tokens in return for the specified amount in the amountIn parameter.

However, it calculates the slippage parameters itself, which doesn't work. Slippage calculations (amountOutMinimum) have to be calculated outside the swap transaction. Otherwise, it uses the already modified pool values to calculate the amountOutMinimum value.

Code Location:

Listing 2: src/adapters/LivepeerAdapter.sol (Lines 138,142-143,148,151)

```
131     ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
  ↳ .ExactInputSingleParams({
132         tokenIn: address(WETH),
133         tokenOut: address(address(LPT)),
134         fee: UNISWAP_POOL_FEE,
135         recipient: address(this),
136         deadline: block.timestamp,
137         amountIn: address(this).balance,
138         amountOutMinimum: 0,
139         sqrtPriceLimitX96: 0
140     });
141
142     (bool success, bytes memory returnData) =
143         address(UNISWAP_ROUTER).staticcall(abi.encodeCall(
  ↳ UNISWAP_ROUTER.exactInputSingle, (params)));
144
145     if (!success) return;
146
```

```
    // set return value of staticcall to minimum LPT value to
↳ receive from swap
    params.amountOutMinimum = abi.decode(returnData, (uint256));

    // execute swap
    UNISWAP_ROUTER.exactInputSingle(params);
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:H/R:N/S:U (8.1)

Recommendation:

It is recommended to calculate the slippage out of the transaction that is going to execute the swap. Also, an external oracle could be used to fetch prices of involved tokens, as long as the oracle's source is different from the pool being used in Uniswap.

Remediation Plan:

PARTIALLY SOLVED: Although the Tenderize team has implemented a TWAP in order to fetch prices from an Uniswap V3 pool which would make it significantly harder to manipulate prices in the pool by performing a front-run or sandwich attack, it's not a good practice to fetch prices from the same pool that is going to be use later in a swap, all in the same transaction.

On the other hand, the TWAP interval should vary according to the liquidity of the pool, it is not the same to fetch prices in a 1-hour range in a huge pool as it is to do so in a smaller one that will tend to be more volatile.

Additionally, it has been set a 10% of allowed slippage based on previously fetched price in the following commits:

- e311dff2ee5d427840fa1b900365a678b63f08d4
- 4073c1f5b387ef7b4ce16c8e979ace68183e9226
- 3c2eefa8c6ec35d0e3a19bf0c8f630d91e07c056

4.3 (HAL-03) STATICCALL TO UNISWAP V3 DOES NOT WORK - HIGH (8.1)

Description:

The _livepeerClaimFees function tries to fetch current prices from Uniswap V3 pool through the usage of staticcall in order to avoid modifying the state during call execution. Therefore, it is expected to return amountOutMinimum value after its execution.

However, this method to fetch prices from Uniswap pools does not work, and it always reverts returning 0 value which, consequently, will be assigned to amountOutMinimum variable.

Proof of Concept:

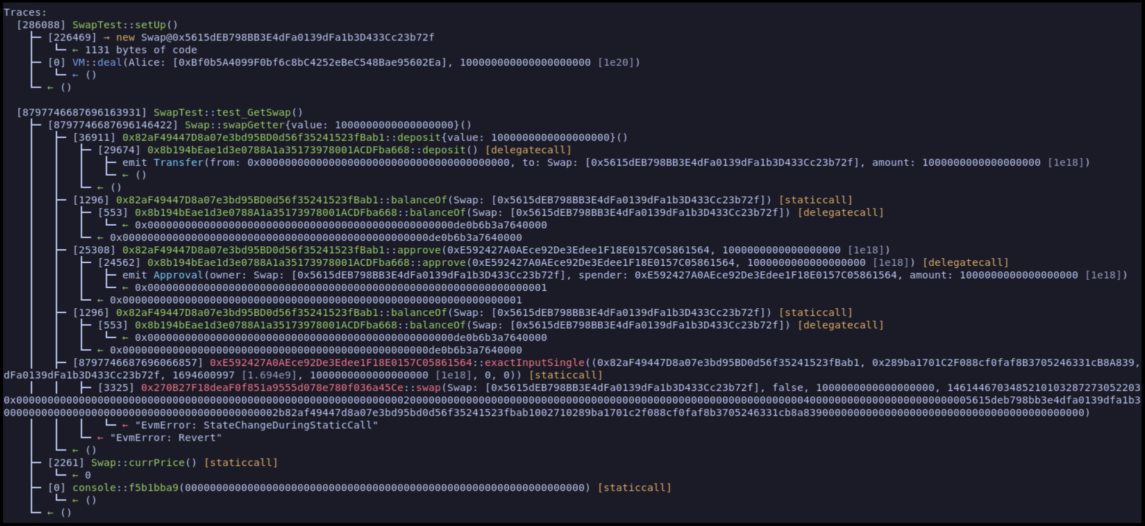
The following function has been used to prove this issue:

Listing 3: Price fetch from Uniswap

```
function swapGetter() public payable {
    WETH.deposit{ value: address(this).balance }();
    ERC20(address(WETH)).safeApprove(address(UNISWAP_ROUTER), WETH
↳ .balanceOf(address(this)));
    // Create initial params for swap
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
↳ .ExactInputSingleParams({
        tokenIn: address(WETH),
        tokenOut: address(address(LPT)),
        fee: UNISWAP_POOL_FEE,
        recipient: address(this),
        deadline: block.timestamp,
        amountIn: WETH.balanceOf(address(this)),
        amountOutMinimum: 0,
        sqrtPriceLimitX96: 0
    });

    (bool success, bytes memory returnData) =
        address(UNISWAP_ROUTER).staticcall(abi.encodeCall(
↳ UNISWAP_ROUTER.exactInputSingle, (params)));
```

```

    if (!success) return;

    // set return value of staticcall to minimum LPT value to receive from swap
    currPrice = abi.decode(returnData, (uint256));
}
```

The execution of the aforementioned function always returns `0` as it is specified in the following execution trace:



```
Traces:
  [286088] SwapTest::setUp()
    ├─ [226469] → new Swap@0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
    │   └─ ← 1131 bytes of code
    ├─ [0] VM::deal(Alice: [0xBf0b5A4099F0bf6c8bC4252eBeC548Bae95602Ea], 100000000000000000000 [1e20])
    │   └─ ← ()
    └─ ← ()

  [879774668769616393 1] SwapTest::test_GetSwap()
    ├─ [8797746687696146422] Swap::swapGetter{value: 1000000000000000000}()
    │   ├─ [36911] 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1::deposit{value: 1000000000000000000}()
    │   │   ├─ [29674] 0x8b194bEae1d3e0788A1a35173978001ACDFba668::deposit() [delegatecall]
    │   │   │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], amount: 1000000000000000000 [1e18])
    │   │   │   └─ ← ()
    │   │   └─ ← ()
    │   ├─ [1296] 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1::balanceOf(Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f]) [staticcall]
    │   │   ├─ [553] 0x8b194bEae1d3e0788A1a35173978001ACDFba668::balanceOf(Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f]) [delegatecall]
    │   │   │   └─ ← 0x0000000000000000000000000000000000000000000000000de0b6b3a7640000
    │   │   └─ ← 0x0000000000000000000000000000000000000000000000000de0b6b3a7640000
    │   ├─ [25308] 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1::approve(0xE592427A0AEce92De3Edee1F18E0157C05861564, 1000000000000000000 [1e18])
    │   │   ├─ [24562] 0x8b194bEae1d3e0788A1a35173978001ACDFba668::approve(0xE592427A0AEce92De3Edee1F18E0157C05861564, 1000000000000000000 [1e18]) [delegatecall]
    │   │   │   ├─ emit Approval(owner: Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], spender: 0xE592427A0AEce92De3Edee1F18E0157C05861564, amount: 1000000000000000000 [1e18])
    │   │   │   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    │   │   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    │   ├─ [1296] 0x82aF49447D8a07e3bd95BD0d56f35241523fBab1::balanceOf(Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f]) [staticcall]
    │   │   ├─ [553] 0x8b194bEae1d3e0788A1a35173978001ACDFba668::balanceOf(Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f]) [delegatecall]
    │   │   │   └─ ← 0x0000000000000000000000000000000000000000000000000de0b6b3a7640000
    │   │   └─ ← 0x0000000000000000000000000000000000000000000000000de0b6b3a7640000
    │   ├─ [8797746687696066857] 0xE592427A0AEce92De3Edee1F18E0157C05861564::exactInputSingle((0x82aF49447D8a07e3bd95BD0d56f35241523fBab1, 0x289ba1701C2F088cf0faf8B3705246331cB8A839,
dFa0139dFa1b3D433Cc23b72f, 1694600997 [1.694e9], 1000000000000000000 [1e18], 0, 0)) [staticcall]
    │   │   ├─ [3325] 0x270B27F18deaF0f851a9555d078e780f036a45Ce::swap(Swap: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], false, 1000000000000000000, 1461446703485210103287273052203
0x000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000000000000000000000004000000000000000000000000005615deb798bb3e4dfa0139dfa1b3
00000000000000000000000000000000000000000002b82af49447d8a07e3bd95bd0d56f35241523fbab1002710289ba1701c2f088cf0faf8b3705246331cb8a839000000000000000000000000000000000000000000)
    │   │   │   └─ ← "EvmError: StateChangeDuringStaticCall"
    │   │   └─ ← "EvmError: Revert"
    │   └─ ← ()
    ├─ [2261] Swap::currPrice() [staticcall]
    │   └─ ← 0
    ├─ [0] console::f5b1bba9(0000000000000000000000000000000000000000000000000000000000000000) [staticcall]
    │   └─ ← ()
    └─ ← ()
```

Figure 1: PoC result

Code Location:

Listing 4: src/adapters/LivepeerAdapter.sol (Lines 145-146,151)

```
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter.ExactInputSingleParams({
        tokenIn: address(WETH),
        tokenOut: address(address(LPT)),
        fee: UNISWAP_POOL_FEE,
        recipient: address(this),
        deadline: block.timestamp,
        amountIn: address(this).balance,
        amountOutMinimum: 0,
        sqrtPriceLimitX96: 0
    });

    (bool success, bytes memory returnData) =
        address(UNISWAP_ROUTER).staticcall(abi.encodeCall(
    ↳ UNISWAP_ROUTER.exactInputSingle, (params)));

    if (!success) return;

    // set return value of staticcall to minimum LPT value to
    ↳ receive from swap
    params.amountOutMinimum = abi.decode(returnData, (uint256));
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:H/R:N/S:U (8.1)

Recommendation:

Since this methodology to fetch prices does not work, it is recommended to remove it and implement oracles.

Remediation Plan:

SOLVED: The Tenderize team has solved this issue by removing the aforementioned part of code in the following commit:

- 4073c1f5b387ef7b4ce16c8e979ace68183e9226

4.4 (HAL-04) APPROVAL NOT REVOKED IF STATICCALL FAILS - LOW (3.1)

Description:

Before a swap is executed in Uniswap, it's required to approve an amount of tokens that will be used as amointIn parameter in the swap. However, if the staticcall fails, the transaction is not reverted; therefore the approval remains assigned to the router which would increase the impact of an attack, in case the approved contract gets hacked.

Code Location:

Listing 5: src/adapters/LivepeerAdapter.sol (Lines 129,145)

```
    ERC20(address(WETH)).safeApprove(address(UNISWAP_ROUTER),
↳ address(this).balance);
    // Create initial params for swap
    ISwapRouter.ExactInputSingleParams memory params = ISwapRouter
↳ .ExactInputSingleParams({
        tokenIn: address(WETH),
        tokenOut: address(address(LPT)),
        fee: UNISWAP_POOL_FEE,
        recipient: address(this),
        deadline: block.timestamp,
        amountIn: address(this).balance,
        amountOutMinimum: 0,
        sqrtPriceLimitX96: 0
    });

    (bool success, bytes memory returnData) =
        address(UNISWAP_ROUTER).staticcall(abi.encodeCall(
↳ UNISWAP_ROUTER.exactInputSingle, (params)));

    if (!success) return;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:L/R:N/S:U (3.1)

Recommendation:

It is recommended to revoke the given approval during function execution in case staticcall fails.

Remediation Plan:

SOLVED: The Tenderize team has solved this issue by removing the aforementioned part of code in the following commit:

- 4073c1f5b387ef7b4ce16c8e979ace68183e9226

4.5 (HAL-05) CONTRACT PAUSE FEATURE MISSING - LOW (2.5)

Description:

It was identified that no high-privileged user can pause any of the scoped contracts. In the event of a security incident, the owner would not be able to stop any plausible malicious actions. Pausing the contract can also lead to more considered decisions.

Recommendation:

It is recommended including the pause feature in the contracts.

BVSS:

AO:A/AC:L/AX:M/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (2.5)

Remediation Plan:

RISK ACCEPTED: The Tenderize team accepted the risk of this issue.

4.6 (HAL-06) INCOMPLETE NATSPEC DOCUMENTATION - INFORMATIONAL (0.0)

Description:

Natspec documentation is useful for internal developers that need to work on the project, external developers that need to integrate with the project, security researchers that have to review it but also for end users given that many chain explorers have officially integrated the support for it directly on their site.

It was detected that the scoped contracts have an incomplete **natspec** documentation.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider adding the missing **natspec** documentation, adhering to the format guideline included in Solidity documentation.

Remediation Plan:

ACKNOWLEDGED: The Tenderize team acknowledged this issue.

4.7 (HAL-07) OPEN TO-DO - INFORMATIONAL (0.0)

Description:

Open TO-DOs can point to architecture or programming issues that still need to be resolved. Often these kinds of comments indicate areas of complexity or confusion for developers. This provides value and insight to an attacker who aims to cause damage to the protocol.

Code Location:

Listing 6: LivepeerAdapter.sol (Line 119)

```
    function isValidator(address validator) public view override
 ↳ returns (bool) {
        // TODO: Change to ACTIVE Orchestrator
        return LIVEPEER.isRegisteredTranscoder(validator);
    }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider resolving the TO-DOs before deploying code to production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

SOLVED: The Tenderize team has solved this issue by removing the aforementioned **TO-DO** comment in the following commit:

- e311dff2ee5d427840fa1b900365a678b63f08d4

AUTOMATED TESTING

5.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

Slither results for livepeer.sol	
Finding	**Impact**
LivepeerAdapter._livepeerClaimFees().pendingFees (src/adapters/LivepeerAdapter.sol#126) is a local variable never initialized	Medium
LivepeerAdapter._livepeerClaimFees() (src/adapters/LivepeerAdapter.sol#124-156) ignores return value by UNISWAP_ROUTER.exactInputSingle(params) (src/adapters/LivepeerAdapter.sol#154)	Medium
LivepeerAdapter.stake(address,uint256) (src/adapters/LivepeerAdapter.sol#79-82) ignores return value by LPT.approve(address(LIVEPEER),amount) (src/adapters/LivepeerAdapter.sol#80)	Medium
End of table for livepeer.sol	

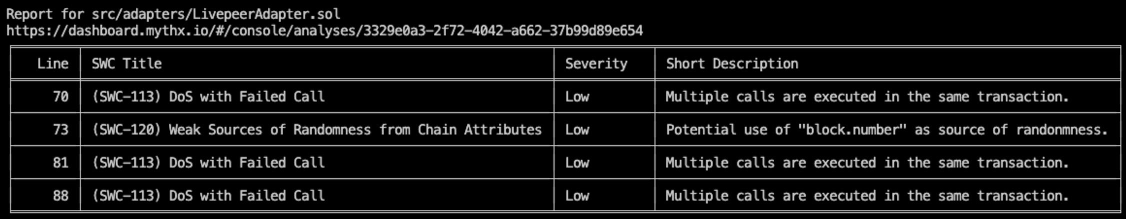
All the issues flagged by Slither were manually reviewed by Halborn. Reported issues were either considered as false positives or are already included in the report findings.

5.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:



```
Report for src/adapters/LivepeerAdapter.sol
https://dashboard.mythx.io/#/console/analyses/3329e0a3-2f72-4042-a662-37b99d89e654
```

Line	SWC Title	Severity	Short Description
70	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
73	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
81	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.
88	(SWC-113) DoS with Failed Call	Low	Multiple calls are executed in the same transaction.

Figure 2: LivepeerAdapter.sol

All the issues flagged by MythX were manually reviewed by Halborn. Reported issues were either considered as false positives or are already included in the report findings.

THANK YOU FOR CHOOSING

HALBORN