ExVul

SMART CONTRACT AUDIT REPORT

BWSwapV2 Smart Contract

JUNE 2025

Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **BWSwapV2** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- 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: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood	*Informational*	*Low*	*Medium*	*High*
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

IMPACT

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on

our private test environment 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 1.2.

- 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.
- Code and business security testing: 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.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
	Abnormal Resource Consumption

Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

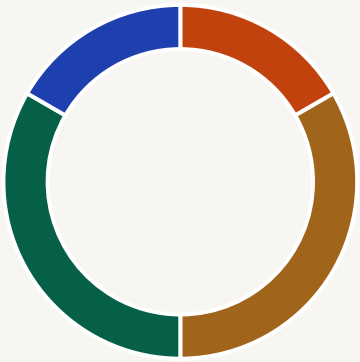
2.1 Project Info And Contract Address

Project Name	Audit Time	Language
BWSwapV2	June 18 2025 – June 23 2025	solidity

Soure code	Link
BWSwapV2	https://github.com/bitgetwallet/bgwswapv2
Commit Hash	b2e5f1abc43b6c400736ed39b1971c601c21fbbb

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	1
MEDIUM	2
LOW	2
INFO	1



2.3 Key Findings

Severity	Findings Title	Status
HIGH	Bypass DeductAmount Fees Through Parameter Manipulation	Acknowledge
MEDIUM	Partial Fill Token Trapped	Fixed
MEDIUM	Unprotected Feature Overwrite in Registry	Fixed
LOW	Missing validation for identical input and output tokens	Acknowledge
LOW	Missing Redundancy Check in manageBWSwapV2	Acknowledge

INFO	Missing Access Control	Acknowledge

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Bypass DeductAmount Fees Through Parameter Manipulation

Location	Severity	Category
BWSwapFee.sol	HIGH	Business Logic Issue

Description:

The protocol's fee collection mechanism allows users to completely circumvent deductAmount fees through proxy contract parameter manipulation. The vulnerability exists in the fee validation logic where the protocol blindly trusts user-provided deductAmount values without any verification.

Root Cause Analysis:

In BWSwapFee._checkDeduct() where fee collection depends entirely on the user-provided deductAmount parameter:

```
// contracts/bwCommon/bwSwapFee/BWSwapFee.sol
function _checkDeduct(
    FeeConfig memory tempConfig,
    uint256 _deductAmount,  // User-controlled input
    uint256 _deductToIndex
) internal pure returns (bool isNeedDeduct, address deductTo) {
    if (_deductAmount > 0) {  // If user sets this to 0, fee collection is skipped
        _checkArrayIndex(_deductToIndex, tempConfig.deductToArray.length);
        deductTo = tempConfig.deductToArray[_deductToIndex];
        isNeedDeduct = true;
    }
    // When _deductAmount = 0, isNeedDeduct remains false (default value)
}
```

This leads to the fee collection logic being completely bypassed:

```
// contracts/bwSwapV2/features/SwapAggregationFeature.sol:422-435
function _collectDeduct(...) internal returns (uint256 deductAmount) {
    deductAmount = 0;

    if (handleFeeCallback.isNeedDeduct) {  // False when deductAmount = 0
        deductAmount = basicParams.feeParams.deductAmount;
        // Fee collection logic executed only when isNeedDeduct = true
    }
    // Returns 0 when isNeedDeduct = false, effectively bypassing all deduct fees
}
```

Attack Vector:

Users can deploy a simple proxy contract that:

1. Accepts original swap parameters from users

2. Modifies swapDetail.basicParams.feeParams.deductAmount = 0

3. Calls BWSwapRouterV2.swap() with modified parameters

4. Passes all protocol validation checks since BWSwapRouterV2 is whitelisted

Parameter Validation Gap:

The protocol's _checkBasicParams() validates various parameters but completely ignores fee-related parameters:

```
// contracts/bwSwapV2/features/SwapAggregationFeature.sol
function _checkBasicParams(SwapDetail calldata swapDetail) internal view {
    // Validates callTarget, approveTarget, deadline, receiver, amounts
    // Gap: No validation of feeParams.deductAmount
    // Missing: Minimum fee enforcement or fee parameter integrity checks
}
```

Bypass Mechanism:

```
User → ProxyContract.swapWithoutDeduct() → BWSwapRouterV2.swap() → BWSwapV2.fallback() → SwapAggregationFeature.swap()
                ↓ (sets deductAmount = 0)
BWSwapFee.handleFee() → _checkDeduct() returns (isNeedDeduct: false) → _collectDeduct() returns 0
```

Recommendations:

Implement mandatory fee enforcement at the protocol level to prevent user manipulation of fee parameters:

```
// contracts/bwSwapV2/features/SwapAggregationFeature.sol
function _checkBasicParams(SwapDetail calldata swapDetail) internal view {
    // ... existing validations ...

+   // Enforce minimum deductAmount based on swap value or fixed minimum
+   uint256 minimumDeduct = _calculateMinimumDeduct(swapDetail.basicParams.amountIn);
+   if (swapDetail.basicParams.feeParams.deductAmount < minimumDeduct) {
+       revert InsufficientDeductAmount();
+   }
}

+function _calculateMinimumDeduct(uint256 amountIn) internal pure returns (uint256) {
+    // Option 1: Percentage-based minimum (e.g., 0.1% of swap amount)
+    return (amountIn * 10) / 10000;
+
+    // Option 2: Fixed minimum amount
+    // return MINIMUM_DEDUCT_AMOUNT;
+}
```

Alternative approach - Remove user control over deductAmount:

```
// contracts/bwCommon/bwSwapFee/BWSwapFee.sol
function handleFee(
-   uint256 _deductAmount,
    uint256 _deductToIndex,
    uint256 _feeRate,
    uint256 _feeToIndex
) external view returns (HandleFeeCallback memory callback) {
    FeeConfig memory tempConfig = config;

+   // Calculate deductAmount based on protocol-defined rules instead of user input
+   uint256 calculatedDeductAmount = _calculateProtocolDeductAmount();

-   (callback.isNeedDeduct, callback.deductTo) = _checkDeduct(tempConfig, _deductAmount, _deductToIndex);
+   (callback.isNeedDeduct, callback.deductTo) = _checkDeduct(tempConfig, calculatedDeductAmount, _deductToIndex);
}
```

Result	FixResult
Confirmed	Acknowledge

3.2 Partial Fill Token Trapped

Location	Severity	Category
SwapAggregationFeature.sol	MEDIUM	Business Logic Issue

Description:

Modern DEX aggregators support partial fills when market conditions prevent full execution. Taking 1inch as an example, their documentation states:

"If the rate for your swap changes beyond your preferred slippage tolerance, there is now the option of enabling 1inch Network's "Partial Fill" setting. With this setting, the 1inch pathfinder algorithm can now cancel part of the route, preventing failed transactions from rate changes, and sending the user's remaining unswapped tokens back to their wallet."

The problem is that in BWSwapV2's architecture, the receiving "wallet" is the protocol contract, not the end user.

The current implementation only tracks tokenOut balances:

```
// contracts/bwSwapV2/features/SwapAggregationFeature.sol
function _swapToken2Token(...) internal returns (uint256 feeAmount) {
    address tokenIn = swapDetail.basicParams.tokenIn;
    address tokenOut = swapDetail.basicParams.tokenOut;
    uint256 balanceBefore = IERC20(tokenOut).balanceOf(address(this));

    (bool success, ) = swapDetail.aggregationParams.callTarget.call{value: 0}(swapDetail.aggregationParams.data);
    _checkCallResult(success);

    uint256 balanceAfter = IERC20(tokenOut).balanceOf(address(this));
    uint256 toReceiverAmount = balanceAfter - balanceBefore;

    TokenHelper.safeTransfer(tokenOut, swapDetail.basicParams.receiver, toReceiverAmount);
}
```

When partial fills occur, unconsumed tokenIn gets returned to the BWSwapV2 contract.

Impact

- User fund loss: Unconsumed tokens remain trapped in the protocol contract

- Accounting mismatch: Protocol assumes full consumption but reality differs

Recommendations:

Track input token balances to detect and return partial fill remainders:

```
function _swapToken2Token(...) internal returns (uint256 feeAmount) {
    address tokenIn = swapDetail.basicParams.tokenIn;
    address tokenOut = swapDetail.basicParams.tokenOut;
+   uint256 tokenInBefore = IERC20(tokenIn).balanceOf(address(this));
    uint256 tokenOutBefore = IERC20(tokenOut).balanceOf(address(this));

    uint256 deductAmount = _collectDeduct(tokenIn, false, swapDetail.basicParams, handleFeeCallback);
    uint256 amountInForSwap = swapDetail.basicParams.amountIn - deductAmount;
    feeAmount = _collectFee(amountInForSwap, tokenIn, false, swapDetail.basicParams, handleFeeCallback);

    (bool success, ) = swapDetail.aggregationParams.callTarget.call{value: 0}(swapDetail.aggregationParams.data);
    _checkCallResult(success);

+   uint256 tokenInAfter = IERC20(tokenIn).balanceOf(address(this));
    uint256 tokenOutAfter = IERC20(tokenOut).balanceOf(address(this));

+   // Return any unconsumed tokens from partial fills
+   uint256 expectedConsumed = amountInForSwap - feeAmount;
+   uint256 actualConsumed = tokenInBefore - tokenInAfter;
+   if (actualConsumed < expectedConsumed) {
+       uint256 remainder = expectedConsumed - actualConsumed;
+       TokenHelper.safeTransfer(tokenIn, swapDetail.basicParams.receiver, remainder);
+   }

    uint256 toReceiverAmount = tokenOutAfter - tokenOutBefore;
    TokenHelper.safeTransfer(tokenOut, swapDetail.basicParams.receiver, toReceiverAmount);
}
```

Result	FixResult
Confirmed	Fixed

3.3 Unprotected Feature Overwrite in Registry

Location	Severity	Category
BWRegistry.sol	MEDIUM	Business Logic Issue

Description:

The manageFeature function in the BWRegistry contract allows complete replacement of existing feature implementations without any verification or confirmation. This creates a significant security risk where existing, critical features can be overwritten accidentally or maliciously.

```
function manageFeature(
    bytes4 _methodId,
    address _proxy,
    bool _isLib,
    bool _isActive
) external onlyOwner whenNotPaused {
    // No check if the feature already exists
    Feature memory feat = Feature({
        proxy: _proxy,
        isLib: _isLib,
        isActive: _isActive
    });
    features[_methodId] = feat; // Overwrites any existing feature

    emit ManageFeature(msg.sender, _methodId, _proxy, _isLib, _isActive);
}
```

Impact

1. Accidental Overwrite: An administrator could accidentally overwrite an existing feature with incorrect parameters, leading to service disruption or security vulnerabilities

2. Malicious Replacement: If the owner account is compromised, an attacker could replace legitimate implementations with malicious code

Recommendations:

Separate Feature Addition and Replacement.

```
// ...existing code...

/**
 * @dev Adds a new feature. Reverts if the feature already exists.
 */
function addFeature(
    bytes4 _methodId,
    address _proxy,
    bool _isLib,
    bool _isActive
) external onlyOwner whenNotPaused {
    // Check that feature does not already exist
    if (features[_methodId].proxy != address(0)) {
        revert FeatureAlreadyExists();
    }

    // Prevent zero address proxies
    if (_proxy == address(0)) {
        revert InvalidProxyAddress();
    }

    Feature memory feat = Feature({
        proxy: _proxy,
        isLib: _isLib,
        isActive: _isActive
    });
    features[_methodId] = feat;

    emit FeatureAdded(msg.sender, _methodId, _proxy, _isLib, _isActive);
}

/**
 * @dev Replaces an existing feature. Reverts if the feature doesn't exist.
 */
function replaceFeature(
    bytes4 _methodId,
    address _proxy,
    bool _isLib,
    bool _isActive
) external onlyOwner whenNotPaused {
    // Check that feature exists
    if (features[_methodId].proxy == address(0)) {
        revert FeatureNotExist();
    }

    // Prevent zero address proxies
    if (_proxy == address(0)) {
        revert InvalidProxyAddress();
    }

    // Store old values for event
    address oldProxy = features[_methodId].proxy;
    bool oldIsLib = features[_methodId].isLib;

    Feature memory feat = Feature({
        proxy: _proxy,
        isLib: _isLib,
        isActive: _isActive
    });
    features[_methodId] = feat;

    emit FeatureReplaced(msg.sender, _methodId, oldProxy, oldIsLib, _proxy, _isLib, _isActive);
}

// The original function can be deprecated or made to call the appropriate new function
// ...existing code...
```

Result	FixResult
Confirmed	Fixed

3.4 Missing validation for identical input and output tokens

Location	Severity	Category
SwapAggregationFeature.sol	LOW	Input Validation

Description:

The swap lacks validation to prevent swaps where tokenIn equals tokenOut, leading to accounting discrepancies in the _swapToken2Token function. The actual implementation fails to correctly account for token balance changes when both tokens are identical.

```
// contracts/bwSwapV2/features/SwapAggregationFeature.sol
function _swapToken2Token(
    IBWSwapFee.HandleFeeCallback memory handleFeeCallback,
    SwapDetail calldata swapDetail
) internal returns (uint256 feeAmount) {
    address tokenIn = swapDetail.basicParams.tokenIn;
    address tokenOut = swapDetail.basicParams.tokenOut;
    uint256 balanceBefore = IERC20(tokenOut).balanceOf(address(this)); // Records initial balance

    uint256 deductAmount = _collectDeduct(
        tokenIn,     // Transfers tokenIn (same as tokenOut)
        false,
        swapDetail.basicParams,
        handleFeeCallback
    );

    feeAmount = _collectFee(
        amountInForSwap,
        tokenIn,     // Transfers more tokenIn (same as tokenOut)
        false,
        swapDetail.basicParams,
        handleFeeCallback
    );

    // External aggregator call
    (bool success, ) = swapDetail.aggregationParams.callTarget.call{value: 0}(swapDetail.aggregationParams.data);
    _checkCallResult(success);

    uint256 balanceAfter = IERC20(tokenOut).balanceOf(address(this));
    uint256 toReceiverAmount = balanceAfter - balanceBefore; // Incorrect calculation

    TokenHelper.safeTransfer(tokenOut, swapDetail.basicParams.receiver, toReceiverAmount);
}
```

Recommendations:

Add explicit validation in _checkBasicParams to reject identical input and output tokens.

Result	FixResult
Confirmed	Acknowledge

3.5 Missing Redundancy Check in manageBWSwapV2

Location	Severity	Category
BWSwapRouterV2.sol	LOW	Code Quality

Description:

The manageBWSwapV2 function doesn't check if the new address is identical to the current one. This has several negative implications:

```
1
2  function manageBWSwapV2(address _bwSwapV2) external onlyOwner {
3      bwSwapV2 = _bwSwapV2;
4      emit ManageBWSwapV2(msg.sender, _bwSwapV2);
5  }
6
```

Impact:

1. Wasted Gas: Unnecessary state updates and event emissions when the address isn't actually changing.

2. Misleading Events: Events are emitted suggesting a change occurred when nothing actually changed.

3. Potential Security Risk: Allows accidentally confirming the wrong address is in use without realizing it.

Recommendations:

Add a check to ensure the new address is different from the current one:

```solidity
function manageBWSwapV2(address _bwSwapV2) external onlyOwner {
    // Check if the new address is different from the current one
    if (_bwSwapV2 == bwSwapV2) {
        revert AddressUnchanged(); // Custom error for clarity
    }

    bwSwapV2 = _bwSwapV2;
    emit ManageBWSwapV2(msg.sender, _bwSwapV2);
}
```

Result	FixResult
Confirmed	Acknowledge

3.6 Missing Access Control

Location	Severity	Category
	INFO	Access Control

Description:

- Only onlyOwner protection for modifying the critical bwSwapV2 address。

- No timelock or multi-signature requirements for changing this critical parameter.

Recommendations:

Implement a Timelock and Integrate a Multi-signature (Multisig) Wallet.

Result	FixResult
Confirmed	Acknowledge

4. CONCLUSION

In this audit, we thoroughly analyzed **BWSwapV2** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

Description	The security of apply verification
Result	Not found
Severity	CRITICAL

5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	CRITICAL

5.1.3 Forged Transfer Vulnerability

Description	Assess whether there is a forged transfer notification vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.4 Transaction Rollback Attack

Description	Assess whether there is transaction rollback attack vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.5 Transaction Block Stuffing Attack

Description	Assess whether there is transaction blocking attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.6 Soft Fail Attack Assessment

Description	Assess whether there is soft fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.7 Hard Fail Attack Assessment

Description	Examine for hard fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.8 Abnormal Memo Assessment

Description	Assess whether there is abnormal memo vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.9 Abnormal Resource Consumption

Description	Examine whether abnormal resource consumption in contract processing
Result	Not found
Severity	CRITICAL

5.1.10 Random Number Security

Description	Examine whether the code uses insecure random number
Result	Not found
Severity	CRITICAL

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

Description	Examine for weakness in cryptograph implementation
Result	Not found
Severity	HIGH

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	MEDIUM

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	Not found
Severity	MEDIUM

5.2.4 Sensitive Information Disclosure

Description	Examine whether sensitive information disclosure issue present in the code
Result	Not found
Severity	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	Not found
Severity	LOW

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

Contact

Website

www.exvul.com

Email

contact@exvul.com

Twitter

@EXVULSEC

Github

github.com/EXVUL-Sec

ExVul