ExVul

SMART CONTRACT AUDIT REPORT

TenetSwap

OCTOBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **TenetSwap** 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: 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
TenetSwap	04/10/2025 - 17/10/2025	Solidity

Repository
https://explorer.morphl2.io/address/0x0885F6E0824163f09459231FD9F2377a77A5346d?tab=contract_code https://explorer.morphl2.io/address/0x3ab6F687F8C2EcA42f0Eb6dE5a8BF8deE077A7C2?tab=contract_code

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	2
MEDIUM	3
LOW	3
INFO	3

2.3 Key Findings

Severity	Findings Title	Status
HIGH	Swap Fees Missing	Acknowledge
HIGH	ERC20 has a race condition vulnerability	Acknowledge
MEDIUM	Too much centralization of authority	Acknowledge
MEDIUM	Swap event data mismatch	Acknowledge
MEDIUM	Router Lacks Support for Fee-On-Transfer Tokens	Acknowledge
LOW	Repurchase configuration accumulation may overflow	Acknowledge
LOW	Missing balance check	Acknowledge
LOW	Incorrect Decimals Hardcoded in Swap Event	Acknowledge
INFO	Missing zero address check	Acknowledge
INFO	Failure to update Approval events in a timely manner	Acknowledge
INFO	Clarify the intent of the code through comments	Acknowledge

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Swap Fees Missing

SEVERITY: HIGH **STATUS:** Acknowledge

PATH:

core/UniswapV2Pair.sol

DESCRIPTION:

In the swap function, the calculation of balance0Adjusted will subtract the relevant transaction fees, but here sub(amount0In.mul(0)) will always be 0, and the caller does not actually pay the transaction fees.

```
// UniswapV2Pair.sol
function swap(uint256 amount0Out, uint256 amount1Out, address to, bytes
    calldata data)
     external
     override
     lock
     onlyRouter
{
     ...
     require(amount0In > 0 || amount1In > 0, ”TENETSWAPV2
    INSUFFICIENT_INPUT_AMOUNT”);
     {
         // scope for reserve{0,1}Adjusted, avoids stack too deep errors
         uint256 balance0Adjusted =
    balance0.mul(1000).sub(amount0In.mul(0));
         uint256 balance1Adjusted =
    balance1.mul(1000).sub(amount1In.mul(0));
         require(
             balance0Adjusted.mul(balance1Adjusted) >=
    uint256(_reserve0).mul(_reserve1).mul(1000 ** 2),
             ”TENETSWAPV2 K”
         );
     }
    _update(balance0, balance1, _reserve0, _reserve1);
    emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out,
   to);
}
```

IMPACT:

The caller does not deduct additional fees when performing swaps, which may cause an imbalance in the economic model.

RECOMMENDATIONS:

It is recommended to set the handling fee to 0.3% according to the UniswapV2 processing method, or modify it according to project requirements.

```
function swap(uint256 amount0Out, uint256 amount1Out, address to, bytes
   calldata data)
    external
    override
    lock
    onlyRouter
{
    ...
    require(amount0In > 0 || amount1In > 0, "TENETSWAPV2
   INSUFFICIENT_INPUT_AMOUNT");
    {
        // scope for reserve{0,1}Adjusted, avoids stack too deep errors
-       uint256 balance0Adjusted =
   balance0.mul(1000).sub(amount0In.mul(0));
-       uint256 balance1Adjusted =
   balance1.mul(1000).sub(amount1In.mul(0));
+       uint256 balance0Adjusted =
   balance0.mul(1000).sub(amount0In.mul(3));
+       uint256 balance1Adjusted =
   balance1.mul(1000).sub(amount1In.mul(3));
        require(
            balance0Adjusted.mul(balance1Adjusted) >=
   uint256(_reserve0).mul(_reserve1).mul(1000 ** 2),
            "TENETSWAPV2 K"
        );
    }
    _update(balance0, balance1, _reserve0, _reserve1);
    emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out,
   to);
}
```

3.2 ERC20 has a race condition vulnerability

SEVERITY: HIGH **STATUS:** Acknowledge

PATH:

`core/UniswapV2ERC20.sol`

DESCRIPTION:

In UniswapV2ERC20.sol, approve allows users to allocate authorized amounts. This will be directly overwritten here, which may lead to a special situation where the authorized amount is already waiting for execution on the chain. The user may modify the authorized amount later, which may cause unexpected results and cause the authorized amount to exceed the expected amount.

```
// UniswapV2ERC20.sol
function _approve(address owner, address spender, uint256 value) private {
    allowance[owner][spender] = value;
    emit Approval(owner, spender, value);
}
```

IMPACT:

Directly overwriting the allowance may result in the authorized amount exceeding the user's expectations, causing financial losses to the user.

RECOMMENDATIONS:

Add additional functions to increase and decrease the amount.

```
+function increaseAllowance(address spender, uint256 addedValue) external
    returns (bool) {
+    require(spender != address(0), "TENETSWAPV2: ZERO_ADDRESS");
+    allowance[msg.sender][spender] = allowance[msg.sender][spender] +
    addedValue;
+    emit Approval(msg.sender, spender, allowance[msg.sender][spender]);
+    return true;
+}
```

```
+function decreaseAllowance(address spender, uint256 subValue) external
    returns (bool) {
+     require(spender != address(0), ”TENETSWAPV2: ZERO_ADDRESS”);
+     uint256 current = allowance[msg.sender][spender];
+     require(current >= subValue, ”TENETSWAPV2: DECREASED_BELOW_ZERO”);
+     allowance[msg.sender][spender] = current - subValue;
+     emit Approval(msg.sender, spender, allowance[msg.sender][spender]);
+     return true;
+}
```

Add external restrictions to approve.

```
function approve(address spender, uint256 value) external virtual
    override returns (bool) {
+    require(value == 0 || allowance[msg.sender][spender] ==
    0,”TENETSWAPV2: approve non-zero to non-zero allowance”);
     _approve(msg.sender, spender, value);
     return true;
}
```

3.3 Too much centralization of authority

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`core/UniswapV2Pair.sol`

DESCRIPTION:

UniswapV2Pair.sol contains operations such as mint, burn, and swap. These operations set the permissions of onlyRouter. Other users cannot interact with the contract directly and must call the Router.

IMPACT:

Ordinary users cannot interact with Pair directly, but must call through Router, which leads to the centralization of Pair access rights.

RECOMMENDATIONS:

It is recommended to modify it to unauthorized call according to uniswapV2, or modify it according to the project's own needs.

3.4 Swap event data mismatch

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`periphery/UniswapV2Router02.sol`

DESCRIPTION:

In the UniswapV2Router02 contract, the pair received by _emitSwapEvent does not match the input_token and outputToken. The pair is path[0], path[1], and the outputToken is path[path.length - 1].

```
    address pair = UniswapV2Library.pairFor(factory, path[0], path[1]);
    _emitSwapEvent(
        msg.sender,
        path[0],
        path[path.length - 1],
        actualAmountIn,
        amounts[amounts.length - 1],
        fee,
        0, // no royalty in V3
        0, // no buyback in V3
        pair
    );
```

IMPACT:

When assigning swapData.new_pair_o_amount, IERC20(outputToken).balanceOf(pair) may not retrieve the expected value, affecting the normal operation of the project.

RECOMMENDATIONS:

Modify or delete the pair acquisition logic according to project requirements, and ensure that the pair actually exists.

3.5 Router Lacks Support for Fee-On-Transfer Tokens

SEVERITY: MEDIUM **STATUS:** Acknowledge

PATH:

`proxy/UniswapV2Router02UpgradeableV2.sol`

DESCRIPTION:

The router implementation is incompatible with fee-on-transfer tokens. Functions like swapExactTokensForTokens and removeLiquidityETH rely on pre-calculated theoretical amounts. This pattern fails when a token in the path deducts a fee during a transfer, as the actual amount received will be less than the theoretical amount, causing the transaction to revert.

The canonical Uniswap V2 Router 02 solves this by providing a separate suite of ...SupportingFeeOnTransferTokens functions that check actual balances post-transfer. This implementation lacks these necessary functions.

```
// proxy/UniswapV2Router02UpgradeableV2.sol
function swapExactTokensForTokens(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] calldata path,
    address to,
    uint256 deadline
) external virtual override ensure(deadline) returns (uint256[] memory
    amounts) {
    // ...
    // The fee logic is based on theoretical `amountIn`
    uint256 fee =
    amountIn.mul(FeeHandler.FEE_NUMERATOR).div(FeeHandler.FEE_DENOMINATOR);
    uint256 actualAmountIn = amountIn.sub(fee);

    // Theoretical amounts are calculated before the actual swap
    amounts = UniswapV2Library.getAmountsOut(factory, actualAmountIn,
    path);
    require(amounts[amounts.length - 1] >= amountOutMin, ”
    TENETSWAPV2Router: INSUFFICIENT_OUTPUT_AMOUNT”);
    // ... swap is executed ...
    _swap(amounts, path, to);
    // This can fail if any token in `path` has a transfer fee.
}
```

IMPACT:

Core functionalities like swaps and liquidity removal will fail for the growing class of fee-on-transfer tokens. This limits the protocol's utility, breaks composability, and can cause user liquidity to be effectively trapped.

RECOMMENDATIONS:

Instead of relying on pre-calculated amounts, adopt the standard Uniswap V2 pattern for supporting fee-on-transfer tokens. This involves checking the actual balance change at the end of the swap to verify the output amount.

3.6 Repurchase configuration accumulation may overflow

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

core/UniswapV2Factory.sol

DESCRIPTION:

In setBuyBackConfig, the externally passed _pcts will be looped through and added to totalPct. Here, uint16 type is used, which may cause totalPct to overflow. The range of uint16 is 65535.

```
// UniswapV2Factory.sol
function setBuyBackConfig(address[5] memory _tokens, address[5] memory
    _quotes, uint16[5] memory _pcts) external {
     require(msg.sender == feeToSetter, ”TENETSWAPV2 FORBIDDEN”);

     // Validate total percentage doesn't exceed 100%
     uint16 totalPct = 0;
     for (uint256 i = 0; i < 5; i++) {
         totalPct += _pcts[i];
     }
     require(totalPct <= 10000, ”TENETSWAPV2 INVALID_PERCENTAGE”); //
    10000 = 100%

     buyBackTokens = _tokens;
     buyBackQuotes = _quotes;
     buyBackPcts = _pcts;
}
```

IMPACT:

The sum of five uint16 values may exceed the maximum value of uint16, causing a calculation overflow and transaction interruption.

RECOMMENDATIONS:

Expand the numeric type of the totalPct parameter to avoid implicit errors.

```
function setBuyBackConfig(address[5] memory _tokens, address[5] memory
   _quotes, uint16[5] memory _pcts) external {
    require(msg.sender == feeToSetter, "TENETSWAPV2 FORBIDDEN");

    // Validate total percentage doesn't exceed 100%
-   uint16 totalPct = 0;
+   uint32 totalPct = 0;
    for (uint256 i = 0; i < 5; i++) {
        totalPct += _pcts[i];
    }
    require(totalPct <= 10000, "TENETSWAPV2 INVALID_PERCENTAGE"); //
   10000 = 100%

    buyBackTokens = _tokens;
    buyBackQuotes = _quotes;
    buyBackPcts = _pcts;
}
```

3.7 Missing balance check

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

`core/UniswapV2ERC20.sol`

DESCRIPTION:

There is a lack of balance checks when performing allowance and transfer. If the balance is insufficient, the transaction will be terminated. Explicit checks are required and errors are thrown.

IMPACT:

Missing explicit balance checks can lead to unclear error messages when transactions fail due to insufficient balance.

RECOMMENDATIONS:

Add explicit checks and throw reasonable errors. This problem also exists in _transfer and _burn.

```
function transferFrom(address from, address to, uint256 value) external
    virtual override returns (bool) {
     if (allowance[from][msg.sender] != type(uint256).max) {
+         require(allowance[from][msg.sender] >= value, "TENETSWAPV2:
    allowance is insufficient");
          allowance[from][msg.sender] =
    allowance[from][msg.sender].sub(value);
     }
     _transfer(from, to, value);
     return true;
}
```

3.8 Incorrect Decimals Hardcoded in Swap Event

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

`proxy/UniswapV2Router02Upgradeable.sol`

DESCRIPTION:

The helper function _emitSwapEvent2, called by swapExactTokensForTokens, incorrectly hardcodes token decimals to 18 in the emitted Swap event. This occurs even when tokens with different decimals (e.g., 6 for USDC) are traded.

```
// proxy/UniswapV2Router02Upgradeable.sol
    function _emitSwapEvent2(
        //...
    ) internal {
        IUniswapV2Router02.SwapData memory swapData;
        // ...
        swapData.input_decimal = 18;
        swapData.output_decimal = 18;
        // ...
    }
```

IMPACT:

Emitting incorrect decimal information will cause off-chain services like analytics platforms and portfolio trackers to miscalculate and display grossly inaccurate trade amounts and prices. This breaks data integrity and damages the protocol's observability.

RECOMMENDATIONS:

Dynamically query the token decimals() instead of using a hardcoded value.

3.9 Missing zero address check

SEVERITY: INFO **STATUS:** Acknowledge

PATH:

`core/UniswapV2Factory.sol, core/UniswapV2ERC20.sol`

DESCRIPTION:

In UniswapV2Factory.sol and UniswapV2ERC20.sol, the setter function will modify some new addresses or pass in some addresses for transfers, but no relevant zero address check is set, allowing the zero address operation to be passed in.

IMPACT:

If the address is changed to zero by mistake, the project may not run normally.

RECOMMENDATIONS:

Add zero address check when calling require(_feeTo != address(0), "INVALID_ADDRESS");

3.10 Failure to update Approval events in a timely manner

SEVERITY: INFO **STATUS:** Acknowledge

PATH:

core/UniswapV2ERC20.sol

DESCRIPTION:

The Approval event data is not updated after the transferFrom authorization transfer.

```
function transferFrom(address from, address to, uint256 value) external
    virtual override returns (bool) {
     if (allowance[from][msg.sender] != type(uint256).max) {
         allowance[from][msg.sender] =
    allowance[from][msg.sender].sub(value);
     }
     _transfer(from, to, value);
     return true;
}
```

IMPACT:

The missing Approval event emission may cause off-chain systems to have incomplete visibility of allowance changes.

RECOMMENDATIONS:

Add event update.

```
function transferFrom(address from, address to, uint256 value) external
    virtual override returns (bool) {
     if (allowance[from][msg.sender] != type(uint256).max) {
         allowance[from][msg.sender] =
    allowance[from][msg.sender].sub(value);
+        emit Approval(from, msg.sender, allowance[from][msg.sender]);
     }
     _transfer(from, to, value);
     return true;
}
```

3.11 Clarify the intent of the code through comments

SEVERITY: INFO **STATUS:** Acknowledge

PATH:

`proxy/UniswapV2Router02UpgradeableV2.sol`

DESCRIPTION:

feeBuyBack is calculated based on leftFee and deducted from it. Using leftFee.div(1) here will cause leftFee to be 0 in the end, which is inconsistent with _processBgbInputV2.

```
// UniswapV2Router02UpgradeableV2.sol
function _processMbInputV2(uint256 fee, address[] memory path)
    internal
    virtual
    returns (uint256 feeRoyalty, uint256 leftFee)
{
    ...

    leftFee = leftFee.sub(feeRoyalty);
    // feeBuyBack = leftFee.div(2);
    feeBuyBack = leftFee.div(1);
    leftFee = leftFee.sub(feeBuyBack);

    ...
}
```

IMPACT:

Unclear descriptions can lead to misunderstandings.

RECOMMENDATIONS:

If it is a project design requirement, please add corresponding comments.

4. CONCLUSION

In this audit, we thoroughly analyzed **TenetSwap** 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

8. About Exvul Security

Premier Security for the Web3 Ecosystem

ExVul is a premier Web3 security firm committed to forging a secure and trustworthy decentralized ecosystem. Our elite team consists of security veterans from world-leading technology and blockchain security firms, including Huawei, YBB Captical, Qihoo 360, Amber, ByteDance, MoveBit, and PeckShield. Team member Nolan is ranked as a top-40 whitehat on Immunefi and is the platform's sole All-Star in the APAC region.

Our expertise covers the full spectrum of Web3 security. We conduct **meticulous smart contract audits**, having fortified thousands of projects on chains like Evm, Solana, Aptos, Sui etc. Our **Blockchain Protocol Audits** secure the core infrastructure of L1/L2 by uncovering deep-seated vulnerabilities. We also offer **comprehensive wallet audits** to protect user assets and provide **proactive web3 pentest**, enabling partners to neutralize threats before they strike.

Trusted by industry leaders, ExVul is the security partner for **OKX, Bitget, Cobo, Infini, Stacks, Aptos, Sui, CoreDAO, Sei** etc.

Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul