EXVUL

SMART CONTRACT AUDIT REPORT

ipflow.fun Smart Contract

NOVEMBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **ipflow.fun** 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 \ IMPACT	Informational	Low	Medium	High
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

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
ipflow.fun	17/11/2025 - 21/11/2025	Rust

Repository
https://github.com/IPFLOW-FUN/duanju-program

Commit Hash
ddcfbebce59050208953c3a17bf61ed36b2cc0f9

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	0
MEDIUM	4
LOW	7
INFO	0

2.3 Key Findings

Severity	Findings Title	Status
MEDIUM	Missing invariant allows withdraw underflow DoS	Fixed
MEDIUM	CreateToken allowed before params configured	Fixed
MEDIUM	migrate_liquidity did not update reserves in a timely manner	Fixed
MEDIUM	Preemptive pool creation DoS official migration	Fixed
LOW	Withdraw fails if recipient ATA missing	Acknowledge
LOW	The create_token parameter input lacks validation	Fixed
LOW	The investment time verification is not rigorous	Fixed
LOW	Premature migration flag allows claim before withdraw	Fixed
LOW	Deadline exact-second blackout	Fixed
LOW	Incorrect error used for already withdrawed	Fixed
LOW	The pre-validation of migrate_liquidity is incomplete	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Missing invariant allows withdraw underflow DoS

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`instructions/withdraw.rs, instructions/set_params.rs, instructions/create_token.rs`

DESCRIPTION:

Globals miss pool SOL requirement invariant, so unsafe params get snapshotted into new BondingCurves. If pool SOL needed exceeds raised SOL, withdraw underflows. Invariant: `token_pool_reserve * token_price_up_bps <= token_investing_supply * BASE_POINTS`

```
// instructions/withdraw.rs
pub fn withdraw(ctx: Context<Withdraw>) -> Result<()> {
    // [...]
    let token_burn = bonding_curve.token_reserves - final_token_reserves
        - bonding_curve.token_creator_reserve -
    bonding_curve.token_platform_reserve;

    let final_sol_reserves = (final_token_reserves as u128
        * bonding_curve.token_launching_price as u128
        / 10u128.pow(token_decimals.into())) as u64;
    let sol_withdraw = bonding_curve.sol_reserves - final_sol_reserves;
    let sol_fee = sol_withdraw * bonding_curve.withdraw_fee_bps as u64 /
    BASE_POINTS;
    let sol_creator = sol_withdraw - sol_fee;
    // [...]
}
```

IMPACT:

withdraw reverts, causing migrate_*/claim to be blocked.

RECOMMENDATIONS:

Add invariant check in set_params with u128; compute token_launching_price in create_token via u128 with overflow checks.

```
// instructions/set_params.rs
pub fn set_params(
    // [...]
) -> Result<()> {
    // [...]
    let total_allocation = token_investing_supply
        .checked_add(token_creator_reserve)
        .and_then(|sum| sum.checked_add(token_platform_reserve))
        .and_then(|sum| sum.checked_add(token_pool_reserve))
        .ok_or(Errors::MathOverflow)?;
    require!(total_allocation <= token_total_supply,
    Errors::InvalidValue);

+   // Invariant I2: pool SOL requirement must not exceed raised SOL
+   // token_pool_reserve * token_price_up_bps <= token_investing_supply
    * BASE_POINTS
+   let lhs = (token_pool_reserve as u128)
+       .checked_mul(token_price_up_bps as u128)
+       .ok_or(Errors::MathOverflow)?;
+   let rhs = (token_investing_supply as u128)
+       .checked_mul(BASE_POINTS as u128)
+       .ok_or(Errors::MathOverflow)?;
+   require!(lhs <= rhs, Errors::InvalidValue);

    let global = &mut ctx.accounts.global;
    // [...]
    Ok(())
}
```

3.2 CreateToken allowed before params configured

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`instructions/create_token.rs`

DESCRIPTION:

create_token only checks global.initialized == true and happily reads global.token_total_supply, token_investing_supply, fee_recipient, migration_caller etc. If set_params was never called, these remain zero/default and a new bonding curve is created with zero supply and unusable global addresses.

```
// instructions/create_token.rs
#[account(
    seeds = [GLOBAL_SEED.as_ref()],
    bump,
    constraint = global.initialized == true @ Errors::NotInitialized,
)]
pub global: Box<Account<'info, Global>>;

pub fn create_token(...) -> Result<()> {
    // [...]
    let global = &ctx.accounts.global;
    // uses global.token_total_supply, token_investing_supply,
    // fee_recipient, migration_caller, ...
}
```

IMPACT:

Tokens can be created against an unconfigured global, leading to zero-supply curves and broken withdraw/migrate flows that cannot be called due to default recipients.

RECOMMENDATIONS:

When calling create_token, enforce that global params have been set (for example, require global.token_total_supply > 0 or a dedicated params_initialized == true flag).

3.3 migrate_liquidity did not update reserves in a timely manner

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`instructions/migrate_liquidity.rs`

DESCRIPTION:

The migrate_liquidity and migrate_liquidity_fallback function allows the migration_caller to transfer tokens from the bonding_curve_vault to the token_account account, but it does not update the reserves in bonding_curve.

```
pub fn migrate_liquidity(
    ctx: Context<MigrateLiquidity>,
) -> Result<()> {
    // ...

    let bonding_curve = &mut ctx.accounts.bonding_curve;
    let token_amount = bonding_curve.token_reserves;
    let sol_amount = bonding_curve.sol_reserves;

    require!(ctx.accounts.bonding_curve_vault.lamports()
        >= bonding_curve.sol_reserves,
   Errors::BondingCurveAlreadyMigrated);

    // ...

    system_program::transfer(
        CpiContext::new_with_signer(
            ctx.accounts.system_program.to_account_info(),
            system_program::Transfer {
                from:
   ctx.accounts.bonding_curve_vault.to_account_info().clone(),
                to: creator_native_account.to_account_info().clone(),
            },
            vault_signer_seeds,
        ),
        sol_amount,
```

```
    )?;

    // ...

    token::transfer(
        CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            token::Transfer {
                from:
   ctx.accounts.associated_bonding_curve.to_account_info().clone(),
                to: creator_token_account.to_account_info().clone(),
                authority: bonding_curve.to_account_info(),
            },
            signer_seeds,
        ),
        token_amount,
    )?;

    // ...

    Ok(())
}
```

IMPACT:

This could lead to discrepancies between the recorded contract status and the actual situation, and subsequent business operations might mistakenly manipulate old data.

RECOMMENDATIONS:

Update bonding_curve.sol_reserves and bonding_curve.token_reserves promptly. migration_liquidity and migration_liquidity_fallback are fixed in the same way.

3.4 Preemptive pool creation DoS official migration

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`instructions/withdraw.rs, instructions/migrate_liquidity.rs`

DESCRIPTION:

After withdraw, the withdraw_recipient (meme creator address configured at create_token) receives a concentrated chunk of MEME and SOL and can preemptively create a pool for the same pair at an arbitrary initial price. The official migration then calls Raydium initialize to create a new pool; if a pool for the same pair already exists, this CPI fails and migrated remains false, blocking claim.

```
// instructions/withdraw.rs
pub fn withdraw(ctx: Context<Withdraw>) -> Result<()> {
    // [...]
    // creator SOL share
    system_program::transfer(
        /* from: bonding_curve_vault, to: withdraw_recipient */,
         sol_creator)?;
    // [...]
    // creator reserved tokens
    token::transfer(
        /* from: associated_bonding_curve, to:
   associated_withdraw_recipient */,
         bonding_curve.token_creator_reserve)?;
    // [...]
    Ok(())
}

// instructions/migrate_liquidity.rs
pub fn migrate_liquidity(ctx: Context<MigrateLiquidity>) -> Result<()> {
    require!(ctx.accounts.bonding_curve.completed == true,
        Errors::BondingCurveNotComplete);
    require!(ctx.accounts.bonding_curve.withdrawed == true,
        Errors::BondingCurveNotWithdrawed);
    // create Raydium pool (fails if already exists for the pair)
```

```
    cpi::initialize(cpi_context, init_amount_0, init_amount_1,
    open_time)?;
    // [...]
    Ok(())
}
```

IMPACT:

DoS of the official migration path, migrated stays false.

RECOMMENDATIONS:

- Introduce an admin-only withdraw_and_migrate entrypoint that performs withdraw and Raydium pool initialization atomically, so the withdraw_recipient (meme creator) only receives MEME and SOL if pool creation succeeds.
- Restrict withdraw and migration entrypoints (migrate_liquidity / fallback) to a trusted project authority (e.g., global.migration_caller or global.authority), keeping withdraw_recipient purely as a payout address rather than a privileged caller.

3.5 Withdraw fails if recipient ATA missing

SEVERITY: LOW **STATUS:** Acknowledge

PATH:

`instructions/withdraw.rs`

DESCRIPTION:

Recipient ATAs (fee_recipient / withdraw_recipient / blackhole) must already exist, accounts are declared without init_if_needed, so missing ATAs revert. Also, set_params can update fee_recipient, the new address may not have its ATA.

```
// instructions/withdraw.rs
#[derive(Accounts)]
pub struct Withdraw<'info> {
    // [...]
    #[account(
        mut,
        associated_token::mint = mint,
        associated_token::authority = fee_recipient,
    )]
    pub associated_fee_recipient: Box<Account<'info, TokenAccount>>,

    #[account(
        mut,
        associated_token::mint = mint,
        associated_token::authority = withdraw_recipient,
    )]
    pub associated_withdraw_recipient: Box<Account<'info, TokenAccount>>,

    // [...]
    #[account(
        mut,
        associated_token::mint = mint,
        associated_token::authority = blackhole,
    )]
    pub associated_blackhole: Box<Account<'info, TokenAccount>>,
    // [...]
```

```
}
```

IMPACT:

Withdraw reverts if ATAs are absent; settlement blocked.

RECOMMENDATIONS:

Auto-create missing ATAs using init_if_needed with a payer signer.

3.6 The create_token parameter input lacks validation

SEVERITY: LOW **STATUS:** Fixed

PATH:

`instructions/create_token.rs`

DESCRIPTION:

In the create_token function, users can pass in basic information about the token to be created, but there are explicit length limits on the name, symbol, and uri at the underlying level. Exceeding these limits will cause errors when initializing or updating metadata.

```
pub fn create_token(
    ctx: Context<CreateToken>,
    token_name: String,
    token_symbol: String,
    token_uri: String,
    // ...
) -> Result<()> {
    require!(token_investing_price > 0 && token_investing_deadline > 0,
        Errors::InvalidValue);
    require!(whitelist_start_at <= investing_start_at,
    Errors::InvalidValue);

    // ...
}
```

IMPACT:

Users may enter parameters that exceed the limits, causing transaction execution to fail. Furthermore, due to the lack of error information, it is impossible to pinpoint the root cause of the error, which affects the user experience.

RECOMMENDATIONS:

Add length limits for name, symbol, and uri parameters.

```
pub fn create_token(
    ctx: Context<CreateToken>,
    token_name: String,
    token_symbol: String,
    token_uri: String,
    // ...
) -> Result<()> {
    require!(token_investing_price > 0 && token_investing_deadline > 0,
        Errors::InvalidValue);
    require!(whitelist_start_at <= investing_start_at,
   Errors::InvalidValue);
+   require!(token_name.len() <= 32, Errors::NameTooLong);
+   require!(token_symbol.len() <= 10, Errors::SymbolTooLong);
+   require!(token_uri.len() <= 200, Errors::UriTooLong);
    // ...
}
```

3.7 The investment time verification is not rigorous

SEVERITY: LOW **STATUS:** Fixed

PATH:

`instructions/create_token.rs`

DESCRIPTION:

The create_token function allows users to set investment time and other related settings. It validates investing_start_at and whitelist_start_at, but it lacks checks for the end time and the current time.

```
// instructions/create_token.rs
pub fn create_token(
    ctx: Context<CreateToken>,
    token_name: String,
    token_symbol: String,
    token_uri: String,
    token_investing_price: u64,
    token_investing_deadline: u64,
    investing_start_at: u64,
    whitelisted: bool,
    merkle_root: [u8; 32],
    whitelist_start_at: u64,
) -> Result<()> {
    require!(token_investing_price > 0 && token_investing_deadline > 0,
        Errors::InvalidValue);
    require!(whitelist_start_at <= investing_start_at,
   Errors::InvalidValue);
    // ...
}
```

IMPACT:

Setting token_investing_deadline to investing_start_at will cause the relevant logic to never start.

RECOMMENDATIONS:

Improve relevant checks to ensure that now <= whitelist_start_at <= investing_start_at < token_investing_deadline.

```
pub fn create_token(
    // ...
    token_investing_deadline: u64,
    investing_start_at: u64,
    // ...
    whitelist_start_at: u64,
) -> Result<()> {
    require!(token_investing_price > 0 && token_investing_deadline > 0,
        Errors::InvalidValue);
    require!(whitelist_start_at <= investing_start_at,
    Errors::InvalidValue);
+   require!(investing_start_at < token_investing_deadline,
    Errors::InvalidValue);
    // Getting clock
    let clock: Clock = Clock::get()?;
+   let now = clock.unix_timestamp.try_into().unwrap();
+   require!(whitelist_start_at >= now, Errors::InvalidValue);

    // ...
}
```

3.8 Premature migration flag allows claim before withdraw

SEVERITY: LOW **STATUS:** Fixed

PATH:

`instructions/set_migrated.rs`

DESCRIPTION:

set_migrated only checks completed, so an admin miscall can mark migrated=true before withdraw/migration.

IMPACT:

Lifecycle ordering is weakened: admin can open claim before withdraw/migration, so creator/platform settlement and LP migration rely on correct off-chain operation.

RECOMMENDATIONS:

Require withdrawed == true (and optionally that migration ran) before setting migrated=true in set_migrated.

3.9 Deadline exact-second blackout

SEVERITY: LOW **STATUS:** Fixed

PATH:

`instructions/sell.rs`

DESCRIPTION:

Sell gate uses strict < while buy uses >; when now == token_investing_deadline, both buy and sell reject, creating a 1-second blackout.

```
// instructions/sell.rs
pub fn sell(...) -> Result<()> {
    // [...]
    // Selling is only allowed after the investing deadline has passed
    require!(bonding_curve.token_investing_deadline < now,
        Errors::BondingCurveNotEnded);
    // [...]
}
```

IMPACT:

Boundary transactions fail unpredictably; short no-trade window at exact deadline.

RECOMMENDATIONS:

Unify boundary semantics: allow sell when now >= deadline.

```
// instructions/sell.rs
- require!(bonding_curve.token_investing_deadline < now,
    Errors::BondingCurveNotEnded);
+ require!(now >= bonding_curve.token_investing_deadline,
    Errors::BondingCurveNotEnded);
```

3.10 Incorrect error used for already withdrawed

SEVERITY: LOW **STATUS:** Fixed

PATH:

`instructions/withdraw.rs, errors.rs`

DESCRIPTION:

Withdraw double-call guard returns a misleading error. It checks withdrawed == false but uses Errors::BondingCurveAlreadyMigrated instead of an “already withdrawed” error.

```
// instructions/withdraw.rs
pub fn withdraw(ctx: Context<Withdraw>) -> Result<()> {
    let bonding_curve = &mut ctx.accounts.bonding_curve;
    require!(bonding_curve.completed == true,
   Errors::BondingCurveNotComplete);
    require!(bonding_curve.withdrawed == false,
        Errors::BondingCurveAlreadyMigrated);
    // [...]
}
```

IMPACT:

Misleads monitoring and debugging; obscures the real state and root cause.

RECOMMENDATIONS:

Use a dedicated error for the already-withdrawn state and reference it in withdraw.

```
// instructions/withdraw.rs
pub fn withdraw(ctx: Context<Withdraw>) -> Result<()> {
    // [...]
-   require!(bonding_curve.withdrawed == false,
        Errors::BondingCurveAlreadyMigrated);
+   require!(bonding_curve.withdrawed == false,
   Errors::AlreadyWithdrawed);
```

```
    // [...]
}
// errors.rs
#[error_code]
pub enum Errors {
   // [...]
   #[msg("The bonding curve has not been withdrawed.")]
   BondingCurveNotWithdrawed,
+  #[msg("The bonding curve has been withdrawed.")]
+  AlreadyWithdrawed,
}
```

3.11 The pre-validation of migrate_liquidity is incomplete

SEVERITY: LOW **STATUS:** Fixed

PATH:

`instructions/migrate_liquidity.rs`

DESCRIPTION:

The set_migrated function directly sets bonding_curve.migrated = true, but migrate_liquidity does not check if bonding_curve.migrated is false during execution, so it may still be called.

IMPACT:

Even when bonding_curve.migrated is set to true, migration_liquidity may still be called, which is not what the project expected.

RECOMMENDATIONS:

When executing migrate_liquidity, check if bonding_curve.migrated is false and set the error code accordingly.

4. CONCLUSION

In this audit, we thoroughly analyzed **ipflow.fun** 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