ExVul

SMART CONTRACT AUDIT REPORT

Hyperionxyz Vaults

Smart Contract

April 2025

Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **Hyperionxyz Vaults** 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 \ 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

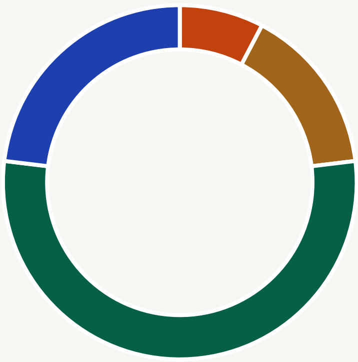
2.1 Project Info And Contract Address

ProjectName	AuditTime	Language
Vaults	April 16 2025–April 24 2025	move

Soure code	Link
Vaults	https://github.com/ Hyperionxyz/Vaults
Commit Hash	cf7cddeeca2145dd577be2a4d25b2bfc50d4bbec

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	1
MEDIUM	2
LOW	7
INFO	3



2.3 Key Findings

Severity	Findings Title	Status
HIGH	Incorrect Slippage Check	Fixed
MEDIUM	Missing Token Order and Identity Validation in LP Token	Fixed
MEDIUM	Missing Tick Range Validation in Vault Creation	Fixed
LOW	Missing Pause Functionality in Vault Contract	Fixed

LOW	Potential Overflow in u128 to u64 Conversion	Fixed
LOW	Integer Truncation Risk in LP Minting	Fixed
LOW	Hardcoded Dust Threshold in Swap Logic	Acknowledge
LOW	Missing Zero Liquidity Check	Fixed
LOW	should check if liquidity_amount is zero or not	Fixed
LOW	should check if the amount_in is bigger than 0	Fixed
INFO	Redundant Admin Check	Acknowledge
INFO	Unused LPObjectRef Structure	Acknowledge
INFO	Single Slippage Check at the End Only	Acknowledge

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Incorrect Slippage Check

Location	Severity	Category
vaults.move	HIGH	Business Logic

Description:

In the Vaults.move module, the slippage protection logic contains a backward condition. Specifically, the following assertion is used to ensure that enough tokens were spent:

```
assert!(amount_in_min < (amount_in - remain_amount), EAMOUNT_IN_TOO_LESS);
```

However, this strict inequality (<) will cause the transaction to revert even if the input exactly matches amount_in_min, which contradicts typical slippage expectations. This can lead to false negatives and failed user transactions under normal conditions.

Recommendations:

Change the slippage check to use a non-strict inequality (<=) to allow exact matches with the minimum input threshold. This ensures the logic aligns with standard slippage protections and avoids unnecessary reverts.

Result	FixResult
Confirmed	Fixed

3.2 Missing Token Order and Identity Validation in LP Token

Location	Severity	Category
vaults.move	MEDIUM	Business Logic

Description:

Two validation issues exist in the lp.move contract's LP token creation functions:

1. Token Pair Order Issue in get_pool_seeds Function

The get_pool_seeds function generates seeds directly from token_a and token_b without sorting. This can create different LP tokens for the same pair in different orders, potentially splitting liquidity pools.

2. Lack of Token Identity Check in LP Creation

The create_share_token function doesn't verify if token_a and token_b are the same, allowing creation of invalid single-token LP tokens.

```
public entry fun create_vault(
        admin: &signer,
        token_a: Object<Metadata>,
        token_b: Object<Metadata>,
        fee_tier: u8,
        tick_lower: u32,
        tick_upper: u32,
        share_cap: u64,
        performance_fee_rate: u64,
        deposit_fee_rate: u64,
        withdraw_fee_rate: u64,
        fee_receiver: address
    ) {

        assert!(is_admin(signer::address_of(admin)), ENOT_ADMIN);
        assert!(performance_fee_rate <= FEE_RATE_DENOMINATOR, EINVALID_FEE_RATE);
        assert!(deposit_fee_rate <= FEE_RATE_DENOMINATOR, EINVALID_FEE_RATE);
        assert!(withdraw_fee_rate <= FEE_RATE_DENOMINATOR, EINVALID_FEE_RATE);

        let (lp_token_refs, lp_signer, _lp_meta) = lp::create_share_token(token_a, token_b, fee_tier);
```

Recommendations:

In the create_vault function, add an is_sorted check before calling create_share_token.

Result	FixResult
Confirmed	Fixed

3.3 Missing Tick Range Validation in Vault Creation

Location	Severity	Category
vaults.move	**MEDIUM**	Business Logic

Description:

The create_vault function allows users to specify tick_lower and tick_upper without any validation. This leads to two critical problems:

Invalid Tick Order:

There is no check ensuring that tick_lower < tick_upper. This violates the core design of Uniswap V3-style tick ranges, potentially resulting in vaults that cannot function properly due to misconfigured tick boundaries.

Lack of Tick Bound Checks:

Neither tick_lower nor tick_upper are validated against the protocol's global minimum/maximum tick bounds. This may allow the creation of positions outside the valid price range supported by the underlying pool, which could cause failures in liquidity provisioning or swaps.

```
public entry fun create_vault(
        admin: &signer,
        token_a: Object<Metadata>,
        token_b: Object<Metadata>,
        fee_tier: u8,
        tick_lower: u32,
        tick_upper: u32,
        share_cap: u64,
        performance_fee_rate: u64,
        deposit_fee_rate: u64,
        withdraw_fee_rate: u64,
        fee_receiver: address
    ) {

        assert!(is_admin(signer::address_of(admin)), ENOT_ADMIN);
        assert!(performance_fee_rate <= FEE_RATE_DENOMINATOR, EINVALID_FEE_RATE);
        assert!(deposit_fee_rate <= FEE_RATE_DENOMINATOR, EINVALID_FEE_RATE);
        assert!(withdraw_fee_rate <= FEE_RATE_DENOMINATOR, EINVALID_FEE_RATE);

        let (lp_token_refs, lp_signer, _lp_meta) = lp::create_share_token(token_a, token_b, fee_tier);
```

Recommendations:

Add validation to ensure tick_lower is less than tick_upper and that both ticks are within the valid range.

Result	FixResult
Confirmed	Fixed

3.4 Missing Pause Functionality in Vault Contract

Location	Severity	Category
vaults.move	**LOW**	Operational Controls

Description:

The Vault contract previously lacked a pause mechanism, which is a fundamental operational control in DeFi contracts. Without a paused flag and corresponding validation logic, administrators are unable to temporarily disable critical functions (e.g., deposit, withdraw) during emergencies, upgrades, or abnormal conditions.

```
struct Vault has key, store {
        lp_token_refs: LPTokenRefs,
        share_cap: u64,
        pool: Object<LiquidityPoolV3>,
        position_id: address,
        total_shares: u64,
        performance_fee_rate: u64,
        withdraw_fee_rate: u64,
        deposit_fee_rate: u64,
        fee_receiver: address,
        token_a: Object<Metadata>,
        token_b: Object<Metadata>,
        fee_tier: u8,
        rewards_meta_list: vector<Object<Metadata>>,
        token_a_store: Object<FungibleStore>,
        token_b_store: Object<FungibleStore>,
        rewards_list: vector<Object<FungibleStore>>,
        pause: bool
    }
```

Recommendations:

Implement a pause mechanism by adding a 'paused' boolean field to the Vault struct. Include access-controlled functions to allow authorized addresses to pause/unpause operations.

Result	FixResult
Confirmed	Fixed

3.5 Potential Overflow in u128 to u64 Conversion

Location	Severity	Category
lp.move	**LOW**	Arithmetic Errors

Description:

In the mint_to function within the lp.move contract, a u128 value is directly cast to u64 without any bounds checking. This may result in silent truncation or transaction aborts if amount > u64::MAX, which can occur in high-liquidity scenarios or incorrect mint calculations. Such overflows break value consistency and may cause minting fewer LP tokens than intended or runtime panics.

```
public(package) fun mint_to(
        lp_token_refs: &LPTokenRefs,
        pool: Object<Metadata>,
        amount: u128,
        lp: address
    ) {
        let token_amount = (amount as u64);
        let store = ensure_lp_token_store(lp_token_refs, lp, pool);
        let token = fungible_asset::mint(&lp_token_refs.mint_ref, token_amount);
        fungible_asset::deposit_with_ref(&lp_token_refs.transfer_ref, store, token);
    }

```

Recommendations:

Add an assertion to ensure the amount is within the valid range for u64 before conversion.

Result	FixResult
Confirmed	Fixed

3.6 Integer Truncation Risk in LP Minting

Location	Severity	Category
vaults.move	LOW	Arithmetic Errors

Description:

In the deposit_with_pair function of the vaults.move contract, the mint_amount is calculated as a u128 using the lp_mint_amount function:

```
let mint_amount = lp_mint_amount(liquidity_total_before_add_liquidity, liquidity_delta, supply);
```

However, it is later cast directly to u64 and added to vault.total_shares:

```
assert!((vault.total_shares + (mint_amount as u64)) <= vault.share_cap, ECAP_REACHED);
```

This direct cast risks integer truncation if mint_amount > u64::MAX, which can silently corrupt share accounting or cause overflows during subsequent additions.

Recommendations:

Add an assertion to verify that mint_amount does not exceed u64::MAX before performing the cast and addition.

Result	FixResult
Confirmed	Fixed

3.7 Hardcoded Dust Threshold in Swap Logic

Location	Severity	Category
vaults.move	**LOW**	Logic Issues

Description:

the threshold 10 used to decide whether to perform another swap (swap_deposit) is hardcoded. This doesn't account for the varying decimal places of different tokens, potentially leaving significant value as dust for some tokens or unnecessarily swapping for others.

```
inline fun deposit_swap_deposit(lp_signer: signer, vault: &mut Vault) {
        let token_a_amount = fungible_asset::balance(vault.token_a_store);
        let token_b_amount = fungible_asset::balance(vault.token_b_store);
        let position = object::address_to_object<Info>(vault.position_id);
        let (tick_lower_i, tick_upper_i) =
            position_v3::get_tick(object::address_to_object<Info>(vault.position_id));
        let tick_lower = i32::as_u32(tick_lower_i);
        let tick_upper = i32::as_u32(tick_upper_i);
        let(liquidity_delta, _amount_a_calc, _amount_b_calc) = router_v3::optimal_liquidity_amounts(
            tick_lower,
            tick_upper,
            vault.token_a,
            vault.token_b,
            vault.fee_tier,
            token_a_amount,
            token_b_amount,
            0,
            0
        );

        let fa_a = dispatchable_fungible_asset::withdraw(&lp_signer, vault.token_a_store, token_a_amount);
        let fa_b = dispatchable_fungible_asset::withdraw(&lp_signer, vault.token_b_store, token_b_amount);

        let(a_in, b_in, a_remain, b_remain) = if(liquidity_delta == 0) {
            (token_a_amount, token_b_amount, fa_a, fa_b)
        } else {
            pool_v3::add_liquidity(&lp_signer, position, liquidity_delta, fa_a, fa_b)
        };

        dispatchable_fungible_asset::deposit(vault.token_a_store, a_remain);
        dispatchable_fungible_asset::deposit(vault.token_b_store, b_remain);

        let amount_a_delta = token_a_amount - a_in;
        let amount_b_delta = token_b_amount - b_in;
        // dust....let it go
        if(amount_a_delta > 10 || amount_b_delta > 10) {
            swap_deposit(lp_signer, vault, amount_a_delta, amount_b_delta);
        };
    }
```

Recommendations:

Replace the hardcoded constant with a configurable or token-aware threshold.

Result	FixResult
Confirmed	Acknowledge

3.8 Missing Zero Amount Checks

Location	Severity	Category
vaults.move	LOW	Input Validation

Description:

Functions like remove_as_pair and remove_as_single assert token_amount != 0, but deposits (deposit_with_pair, deposit_with_single) don't explicitly check for zero amount_a_desired/amount_b_desired or amount_in. While downstream calls might handle this, explicit checks can prevent wasted gas or unexpected behavior.

Recommendations:

Add explicit assertions at the beginning of each deposit function to check that user-supplied amounts are non-zero.

Result	FixResult
Confirmed	Fixed

3.9 Missing Zero Liquidity Check

Location	Severity	Category
vaults.move	LOW	Input Validation

Description:

In the close_position function, the logic directly calls pool_v3::remove_liquidity using the liquidity_amount retrieved from position_v3::get_liquidity. However, there is no check to ensure that liquidity_amount is non-zero. If the value is 0, calling remove_liquidity with zero liquidity may lead to unexpected behavior, wasted gas, or even reverts inside the remove_liquidity logic, depending on the pool implementation.

```
public entry fun close_position(
        admin: &signer,
        vault_obj: Object<Vault>,
    ) acquires Vault {

        assert!(package_manager::is_admin(signer::address_of(admin)), ENOT_ADMIN);

        let vault = get_vault_mut(vault_obj);

        // remove_liquidity
        let lp_signer = lp::get_signer(&vault.lp_token_refs);
        let position = object::address_to_object<Info>(vault.position_id);
        let liquidity_amount = position_v3::get_liquidity(position);

        claim_fees_and_rewards(&lp_signer, position, vault);

        let (
            token_a_opt,
            token_b_opt
        ) = pool_v3::remove_liquidity(&lp_signer, position, liquidity_amount);
        deposit_liquidity(vault, token_a_opt, token_b_opt);
        let(pool_current_tick, pool_current_sqrt_price) =
            pool_v3::current_tick_and_price(object::object_address(&vault.pool));
```

Recommendations:

Add an assertion before removing liquidity to validate that liquidity_amount > 0.

Result	FixResult
Confirmed	Fixed

3.10 Should check if the amount_in is biggger than 0

Location	Severity	Category
vaults.move	LOW	Input Validation

Description:

In the swap_liquidity_token_to_another function, there is no explicit check to ensure that amount_in is greater than zero before performing a withdrawal and initiating a swap.

```
public entry fun swap_liquidity_token_to_another(
        admin: &signer,
        vault: Object<Vault>,
        thala_pool: Object<Pool>,
        token_from: Object<Metadata>,
        token_to: Object<Metadata>,
        amount_in: u64,
        amount_out_min: u64
    ) acquires Vault {

        assert!(package_manager::is_admin(signer::address_of(admin)), ENOT_ADMIN);

        let vault = get_vault_mut(vault);

        let a2b = if(token_from == vault.token_a) {true} else {false};

        let lp_signer = lp::get_signer(&vault.lp_token_refs);

        if(object::object_exists<Info>(vault.position_id)) {
            let position = object::address_to_object<Info>(vault.position_id);
            claim_fees_and_rewards(&lp_signer, position, vault);
        };

        let fa_in = if(a2b) {
            dispatchable_fungible_asset::withdraw(&lp_signer, vault.token_a_store, amount_in)
        } else {
            dispatchable_fungible_asset::withdraw(&lp_signer, vault.token_b_store, amount_in)
        };
```

Recommendations:

Add a check to validate that amount_in > 0 before continuing with the function logic.

Result	FixResult
Confirmed	Fixed

3.11 Redundant Admin Check

Location	Severity	Category
vaults.move	**INFO**	Logic Optimization

Description:

The close_swap_rebalance_auto function performs an unnecessary admin check after calling close_position, which already includes an admin check.

```
public entry fun close_swap_rebalance_auto(
        admin: &signer,
        vault_obj: Object<Vault>,
        tick_lower: u32,
        tick_upper: u32,
    ) acquires Vault {

        close_position(admin, vault_obj);

        assert!(package_manager::is_admin(signer::address_of(admin)), ENOT_ADMIN);
```

```
public entry fun close_position(
        admin: &signer,
        vault_obj: Object<Vault>,
    ) acquires Vault {

        assert!(package_manager::is_admin(signer::address_of(admin)), ENOT_ADMIN);

```

Recommendations:

Remove the unnecessary second admin verification in close_swap_rebalance_auto to streamline execution.

Result	FixResult
Confirmed	Acknowledge

3.12 Unused LPObjectRef Structure

Location	Severity	Category
lp.move	**INFO**	Code Maintainability

Description:

The LPObjectRef struct is defined as a resource group member but is never created or utilized in the code.

```
#[resource_group_member(group = aptos_framework::object::ObjectGroup)]
    struct LPObjectRef has key,drop {
        token_a: Object<Metadata>,
        token_b: Object<Metadata>,
        fee_tier: u8,
        lp_amount: u64,
        transfer_ref: object::TransferRef,
        delete_ref: object::DeleteRef,
        extend_ref: object::ExtendRef
    }
```

Recommendations:

Remove the LPObjectRef struct if it's unnecessary, or implement its functionality if it's required.

Result	FixResult
Confirmed	Acknowledge

3.13 Single Slippage Check at the End Only

Location	Severity	Category
vaults.move	**INFO**	Protocol Logic

Description:

The function only validates the final output amount meets the minimum threshold (amount_out_min), with no checks on intermediate swaps. This means severe losses could occur during any intermediate step before being detected.

```
assert!(comparator::compare(&temp_metadata, &to_token).is_equal(), EOUT_TOKEN_NOT_MATCHED);

assert!(fungible_asset::amount(&out) >= amount_out_min, EOUT_AMOUNT_TOO_LESS);
```

Recommendations:

A malicious actor could observe a pending multi-hop swap transaction and execute a sandwich attack:

1. Front-run by buying tokens in one of the intermediate pools to drive up prices.

2. Let the victim's transaction execute at unfavorable rates in the manipulated pool.

3. Back-run by selling the tokens, profiting from the victim's forced execution.

Result	FixResult
Confirmed	Acknowledge

4. CONCLUSION

In this audit, we thoroughly analyzed **Hyperionxyz Vaults** 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