SPYWOLF.CO @SPYWOLFNETWORK @SPYWOLFNETWORK

SPYWOLF

Security Audit Report

Audit prepared for

Utherverse: Token Claim

Completed on

November 24, 2024

OVERVIEW

This goal of this report is to review the main aspects of the project to help investors make an informative decision during their research process.

You will find a a summarized review of the following key points:

- ✔ Program's source code
- ✔ Team transparency and goals
- ✔ Website's age, code, security and UX
- ✔ Whitepaper and roadmap
- ✔ Social media & online presence

" *The results of this audit are purely based on the team's evaluation and does not guarantee nor reflect the projects outcome and goal* "

- SPYWOLF Team -

TABLE OF CONTENTS

UTHERVERSE (Token/Staking)

PROJECT DESCRIPTION

Utherverse is not just another player in the metaverse space – we are the pioneers, with over 15 years of experience in building successful virtual economies and communities. Our track record speaks for itself, but our vision for the future is what truly sets us apart. By harnessing the immense potential of web3, blockchain, and AI, we are creating a metaverse that is unmatched in its immersion, adaptability, and profitability.

Release Date: Launches in November, 2024

Category: Token Claim

CLAIM PROGRAM

Token Name
Claim Program

Symbol

Program Address
GJdNPUnhyjz4x47fibedUxmLY7Xx42E6SrQGfEJUzj9S

Network
Solana

Language
Rust

Deployment Date
Nov 18, 2024

Program Type
Claim

Total Supply

Status
mainnet

TAXES

Buy Tax
0

Sell Tax
0

*This type of program does not have taxes

Our Program Review Process

The contract review process pays special attention to the following:

- Testing the programs against both common and uncommon vulnerabilities
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring program logic meets the specifications and intentions of the client.
- Cross referencing program structure and implementation against similar programs produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Blockchain security tools used:

- Solana Program Library (SPL)
- Manual Auditing / Sec3 / Neodyme
- Rust Compiler
- Anchor Framework

CODE REVIEW

Vulnerability	
Insecure Access Control for Beneficiary Updates	High Risk
State Update vs. Transfer Atomicity	Medium Risk
Escrow Wallet Authority Hijack	High Risk
Infinite Token Minting via Malicious Initialization	Medium Risk
No Recovery Mechanism for Unclaimed Tokens	Low Risk
Program Initialization and Transaction Validation	Passed

VULNERABILITY ANALYSIS

ERRORS FOUND

1. Insecure Access Control for Beneficiary Updates

High Risk

The update_bulk_user_status function allows bulk updates to beneficiary statuses without verifying the caller's identity. This allows unauthorized actors to maliciously update all beneficiaries to is_claimed = true, effectively locking funds.

Real Impact

- Malicious actors can disrupt the claim process, locking all escrowed tokens permanently.
- Potential loss of user trust and system functionality.

Mitigation

- Implement strict access control to validate the identity of the caller. Ensure that only the program's initializer or a designated admin can perform such updates. Use Anchor constraints for access control:

```
#[derive(Accounts)]
pub struct UpdateBeneficiaryStatus<'info> {
    #[account(mut, has_one = initializer @ ErrorCode::UnauthorizedAccess)]
    pub data_account: Account<'info, DataAccount>,
    // Additional accounts...
}
```

This ensures the function cannot be called by unauthorized accounts.

ERRORS FOUND

2. State Update vs. Transfer Atomicity

Medium Risk

The claim function updates the beneficiary's state (is_claimed and in_process) before completing the token transfer. If the transfer fails (e.g., due to insufficient escrow balance or network issues), the state becomes inconsistent.

Real Impact

- Claims may be marked as completed even though tokens are not transferred.
- Loss of claim rights for the beneficiary due to inconsistent state.

Mitigation

- Update the state atomically with the token transfer. Implement a "commit-then-execute" pattern to ensure consistency:

```
data_account.beneficiaries[index].in_process = true;

// Attempt the transfer
let transfer_result = token_transfer(...);

// Revert state if the transfer fails
require!(transfer_result.is_ok(), ErrorCode::TransferFailed);
data_account.beneficiaries[index].is_claimed = true;
data_account.beneficiaries[index].in_process = false;
```

Using Anchor's error-handling mechanism ensures that state updates occur only if the transfer is successful.

ERRORS FOUND

3. Escrow Wallet Authority Hijack

High Risk

The escrow_wallet authority is tied to a mutable account rather than a Program Derived Address (PDA). If the private key of the escrow authority is compromised, an attacker can drain all escrowed funds.

Real Impact

- Full loss of funds held in escrow.
- Severe damage to the program's integrity.

Mitigation

- Use a PDA as the escrow wallet authority. A PDA ensures that only the program can control the escrow wallet, eliminating the risk of key compromise.

```
#[account(
    mut,
    seeds = [b"escrow-wallet".as_ref(), data_account.key().as_ref()],
    bump,
    token::authority = pda_authority,
    token::mint = mint_account
)]
pub struct EscrowWallet<'info> {
    // Accounts...
}
```

This approach uses the program's logic to secure the authority instead of relying on external keys.

VULNERABILITY ANALYSIS

ERRORS FOUND

4. Infinite Token Minting via Malicious Initialization

Medium Risk

The initialize function does not enforce limits on the allocation amount or validate the beneficiaries. This allows malicious actors to:

1. Over-allocate tokens, devaluing the token supply.
2. Create fake beneficiaries, disrupting legitimate use.

Real Impact

- Economic devaluation due to excessive token allocation.
- Spam attacks, making the program unusable for genuine participants.

Mitigation

- Introduce constraints during initialization to validate inputs and enforce limits:

```
require!(amount <= MAX_TOTAL_ALLOCATION, ErrorCode::ExcessiveAllocation);
require!(beneficiaries.len() > 0, ErrorCode::NoBeneficiaries);
```

Additionally:

- Implement beneficiary validation to ensure they are legitimate accounts.
- Define a hard-coded maximum allocation per round.
- This safeguards the token supply and ensures program integrity.

ERRORS FOUND

5. No Recovery Mechanism for Unclaimed Tokens

Low Risk

Tokens unclaimed by beneficiaries remain locked indefinitely, reducing the total token supply. This can occur due to:

- Misconfigured accounts.
- Forgotten or abandoned claims.

Real Impact

- Permanent loss of unclaimed tokens.
- Reduction in available supply for legitimate use.

Mitigation

- Introduce an "admin reclaim" function to recover unclaimed tokens after a deadline:

```
pub fn reclaim_unclaimed_tokens(ctx: Context<Reclaim>, round: u64) -> Result<()> {
    let current_time = Clock::get()?.unix_timestamp;
    require!(
        current_time > data_account.deadline,
        ErrorCode::ClaimPeriodNotOver
    );

    // Transfer unclaimed tokens back to the admin
    let reclaim_result = token_transfer(ctx.accounts.unclaimed_tokens_transfer)?;
    require!(reclaim_result.is_ok(), ErrorCode::ReclaimFailed);

    Ok(())
}
```

This function allows recovery of unused tokens, ensuring efficient use of resources.

VULNERABILITY ANALYSIS

ERRORS FOUND

Program Initialization: PASSED

Passed

- Initialization parameters are correctly enforced in terms of setting deadlines and token allocation amounts.
- Account derivations are secure, with proper seeds and bump validations.

Transaction Validation: PASSED

Passed

- Critical functions validate inputs, avoiding overflows or underflows.
- Function reentrancy is effectively prevented using temporary state markers (e.g., in_process).

STAKE PROGRAM

Token Name
Stake Program

Symbol

Program Address

Network
Solana

Language
Rust

Deployment Date
Nov 18, 2024

Program Type
Stake

Total Supply

Status
mainnet

TAXES

Buy Tax
0

Sell Tax
0

*This type of program does not have taxes

Our Program Review Process

The contract review process pays special attention to the following:

- Testing the programs against both common and uncommon vulnerabilities
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring program logic meets the specifications and intentions of the client.
- Cross referencing program structure and implementation against similar programs produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Blockchain security tools used:

- Solana Program Library (SPL)
- Manual Auditing / Sec3 / Neodyme
- Rust Compiler
- Anchor Framework

CODE REVIEW

Vulnerability	
Integer Overflow/Underflow	High Risk
Lack of Input Validation for admin_withdraw	Medium Risk
Unauthorized Access in update_pool_info	High Risk
No Handling for Division by Zero	Medium Risk
Stake Seed Conflict Check	Medium Risk
Token Transfer Validation and Stake Initialization	Passed

VULNERABILITY ANALYSIS

ERRORS FOUND

1. Integer Overflow/Underflow

High Risk

Unchecked arithmetic operations in calculations, such as rewards or staking amounts, may lead to overflow or underflow errors. While checked_mul is used in some instances, other areas, such as reward rate calculations, lack similar safeguards.

Attack Scenario

A malicious user could manipulate apy, apy_denominator, or other parameters to trigger overflow or underflow. This could result in:

- Excessive rewards credited to themselves.
- Locking up all tokens due to miscalculated reward rates.

Mitigation

- Use safe arithmetic methods, such as checked_*, for all calculations involving user input or contract state.

Code Adjustment Example

```
let reward_rate = stake_info
    .staked_amount
    .checked_mul(pool_info.apy)
    .and_then(|x| x.checked_div(pool_info.apy_denominator))
    .and_then(|x| x.checked_div(constants::SLOTS_PER_YEAR))
    .ok_or(ErrorCode::ArithmeticOverflow)?;
```

This ensures calculations are safe and prevents unexpected behavior from invalid inputs.

ERRORS FOUND

2. Lack of Input Validation for admin_withdraw Function

Medium Risk

The admin_withdraw function does not validate the withdrawal amount, allowing requests for amounts exceeding the vault's balance.

Real Impact

- A failed transaction due to insufficient funds may freeze the contract.
- Malicious or careless admins could destabilize the program.

Mitigation

- Add validation to ensure the vault has sufficient funds before executing withdrawals.

Code Adjustment Example:

```
let vault_balance = ctx.accounts.token_vault_account.amount;
if vault_balance < value {
    return Err(ErrorCode::InsufficientVaultBalance.into());
}
```

This prevents over-withdrawal and ensures the program operates reliably.

VULNERABILITY ANALYSIS

ERRORS FOUND

3. Unauthorized Access in update_pool_info

High Risk

The update_pool_info function relies on the admin's public key passed as a parameter. This lacks robust verification and could allow an attacker to spoof the admin address.

Explanation:

- An attacker could hijack pool control by manipulating the admin key.
- Lack of strict verification increases the risk of unauthorized access.

Mitigation

- Verify the caller's identity using trusted sources, such as comparing the admin's public key against stored data in the program.

Code Adjustment Example

```
if ctx.accounts.admin.key() != ctx.accounts.pool_info.admin {
    return Err(ErrorCode::Unauthorized.into());
}
```

To improve security further, avoid allowing arbitrary admin key changes, or implement strict change procedures (e.g., multi-signature or timelock).

VULNERABILITY ANALYSIS

ERRORS FOUND

4. No Handling for Division by Zero (APY and Denominator)

Medium Risk

The program performs division operations without validating divisors. If apy_denominator is zero, it may cause a panic or undefined behavior.

Explanation

- Malicious actors setting apy_denominator to zero.
- Bugs or improper state updates.

Mitigation

- Add validation to ensure divisors, such as apy_denominator, are non-zero before performing calculations.

Code Adjustment Example:

```
if pool_info.apy_denominator == 0 {
    return Err(ErrorCode::InvalidApyDenominator.into());
}
```

This ensures safe and predictable calculations, preventing runtime errors.

ERRORS FOUND

5. Stake Seed Conflict Check

Medium Risk

The DeStake function does not ensure that stake_seed is unique. Conflicting seeds could lead to one user's stake being overwritten by another's.

Explanation

- Accidental or malicious reuse of stake_seed could disrupt staking data.
- Users may lose their stake due to overwritten accounts.

Mitigation

- Verify the uniqueness of stake_seed before creating new stakes.

Code Adjustment Example:

```
let existing_stake = StakeInfo::fetch_by_seed(stake_seed);
if existing_stake.is_some() {
    return Err(ErrorCode::StakeSeedConflict.into());
}
```

This guarantees that each stake_seed is unique, preventing conflicts.

VULNERABILITY ANALYSIS

ERRORS FOUND

Token Transfer Validation: PASSED

Passed

- The program correctly validates token transfers and ensures that accounts are correctly authorized.
- Checks for ownership and sufficient token balances are implemented.

Stake Initialization and Withdrawal: PASSED

Passed

- The stake initialization process enforces deadlines and minimum staking amounts.
- Token withdrawals are checked for eligibility, preventing unauthorized access.

SPYWOLF
CRYPTO SECURITY

Audits | KYCs | dApps
Contract Development

ABOUT US

We are a growing crypto security agency offering audits, KYCs and consulting services for some of the top names in the crypto industry.

- OVER 700 SUCCESSFUL CLIENTS
- MORE THAN 1000 SCAMS EXPOSED
- MILLIONS SAVED IN POTENTIAL FRAUD
- PARTNERSHIPS WITH TOP LAUNCHPADS, INFLUENCERS AND CRYPTO PROJECTS
- CONSTANTLY BUILDING TOOLS TO HELP INVESTORS DO BETTER RESEARCH

To hire us, reach out to
contact@spywolf.co or
t.me/joe_SpyWolf

FIND US ONLINE

SPYWOLF.CO

@SPYWOLFNETWORK

@SPYWOLFNETWORK

Disclaimer

This report shows findings based on our limited project analysis, following good industry practice from the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, overall social media and website presence and team transparency details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report.
While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER:
By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice.
No one shall have any right to rely on the report or its contents, and SpyWolf and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SpyWolf) owe no duty of care towards you or any other person, nor does SpyWolf make any warranty or representation to any person on the accuracy or completeness of the report.
The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SpyWolf hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SpyWolf hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SpyWolf, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts, website, social media and team.
No applications were reviewed for security. No product code has been reviewed.