SALUS SECURITY

OCT 2025

CODE SECURITY ASSESSMENT

PVPFUN

Overview

Project Summary

- Name: Pvpfun token
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Gametaverse/pvpfun-token
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Pvpfun token
Version	v2
Type	Solidity
Dates	Oct 24 2025
Logs	Oct 23 2025; Oct 24 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	0
Total informational issues	2
Total	3

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Blacklist can be bypassed via transferFrom and unrestricted approvals	Medium	Business Logic	Resolved
2	Custom errors in require statements	Informational	Gas Optimization	Resolved
3	Function visibility overly permissive	Informational	Code Quality	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Blacklist can be bypassed via transferFrom and unrestricted approvals

Severity: Medium	Category: Business Logic
Target: - contracts/PVPToken.sol	

Description

The blacklist mechanism only checks `from` and `to` within `_update()`, but does not restrict the actual caller (`spender = msg.sender`) in `transferFrom()`. As a result, blacklisted addresses can still spend previously granted allowances to move funds from non-blacklisted users to other non-blacklisted accounts. Additionally, the contract does not prevent users from approving or maintaining allowances for blacklisted addresses, enabling them to retain or even gain new spending rights despite being blacklisted.

Blacklisted actors can continue interacting with the token by abusing existing or newly obtained approvals, potentially draining user funds.

contracts/PVPToken.sol:L65-L73

```
function _update(
    address from,
    address to,
    uint256 value
) internal override {
    require(!_isBlacklisted[from], "ERC20: sender is blacklisted");
    require(!_isBlacklisted[to], "ERC20: receiver is blacklisted");
    super._update(from, to, value);
}
```

Recommendation

Block blacklisted `msg.sender` in the `_update()` function, and restrict approvals to/from blacklisted addresses.

Status

The team has resolved this issue in commit 671a2d3.

2.3 Informational Findings

2. Custom errors in require statements

Severity: Informational	Category: Gas Optimization
Target: - contracts/PVPToken.sol	

Description

contracts/PVPToken.sol:L65-L73

```
function _update(
    address from,
    address to,
    uint256 value
) internal override {
    require(!_isBlacklisted[from], "ERC20: sender is blacklisted");
    require(!_isBlacklisted[to], "ERC20: receiver is blacklisted");
    super._update(from, to, value);
}
```

In `PVPToken`, the contract uses `require(condition, "error message")` statements. Since Solidity version 0.8.26, require statements support custom errors, which are more gas-efficient and improve code clarity. Initially, this feature was only available through the IR pipeline, but starting from Solidity 0.8.27, it is also supported in the legacy pipeline.

Recommendation

Consider replacing all `require(condition, "error message")` statements with `require(condition, CustomError())` to improve readability and reduce gas consumption.

Status

The team has resolved this issue in commit 671a2d3.

3. Function visibility overly permissive

Severity: Informational	Category: Code Quality
Target: - contracts/PVPToken.sol	

Description

`addToBlacklist()` and `removeFromBlacklist()` functions in the contract are declared with broader visibility than necessary (public instead of external). Overly permissive visibility can expose internal logic to unintended callers, increase the attack surface, and slightly increase gas costs. Functions intended to be called externally only do not need public visibility and could be declared external to better communicate intended usage and optimize gas.

contracts/PVPToken.sol:L33-L45

```
function addToBlacklist(address account) public onlyOwner {
   _isBlacklisted[account] = true;
   emit AddedToBlacklist(account);
}

function removeFromBlacklist(address account) public onlyOwner {
   _isBlacklisted[account] = false;
   emit RemovedFromBlacklist(account);
}
```

Recommendation

Consider changing the visibility of the above function.

Status

The team has resolved this issue in commit 671a2d3.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 8ed6876:

File	SHA-1 hash
PVPToken.sol	22b0f0c45e412e72d8faff52c5d9d81ae1f9c9a4