vEMPIRE

SMART CONTRACT AUDIT
FINAL REPORT

June 08, 2022

IMMUNEBYTES

TOC

TABLE OF CONTENTS

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Introduction

1. About vEMPIRE

vEmpire Metaverse Token Staking is the world's first protocol to enable Metaverse token staking. The Metaverse DeFi aggregator enables the community to earn lucrative interest on their Metaverse holdings passively. This means that as well as being exposed to assets with enormous potential for capital appreciation, investors can now earn interest in the hundreds of percent on their holdings while helping contribute to decentralized governance. This unlocks the potential to earn passive income on cryptocurrency holdings, which completely invalidates the argument that cryptocurrencies are unproductive assets. Currently, the vEmpire platform supports staking for six Metaverse-based cryptocurrencies; ETH, SAND, MANA, AXS, STARL, and BNB.

Visit https://v-empire.io/ to know more about it.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 125+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-ups with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The vEMPIRE team has provided the following doc for the purpose of audit:

1. **vEmpire Solana Flow.pdf**

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: vEMPIRE
- Contract Name: context.rs, error.rs, lib.rs, state.rs, utils.rs
- GitHub Address: https://github.com/0xblock-co/v-empire-solana-audit
- Commit Hash for Initial Audit: 47c6356c27ee3a40029144b2da7c11866717f71d
- Commit Hash for Final Audit: 7285b95faf20ee9976cfea14b7f92c7e7f887878
- Languages: RUST(Smart contract)

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level Reference

Every issue in this report were assigned a severity level from the following:

Admin/Owner Privileges can be misused either intentionally or unintentionally.
Critical severity issues can result in leakage of funds or not work according to business logic.
High severity issues will bring problems and should be fixed.
Medium severity issues could potentially bring problems and should eventually be fixed
Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

ISSUES	Admin	Critical	High	Medium	Low
OPEN	-	-	-	-	-
CLOSE	-	1	1	1	1

Admin/Owner Privileges

No issues were found

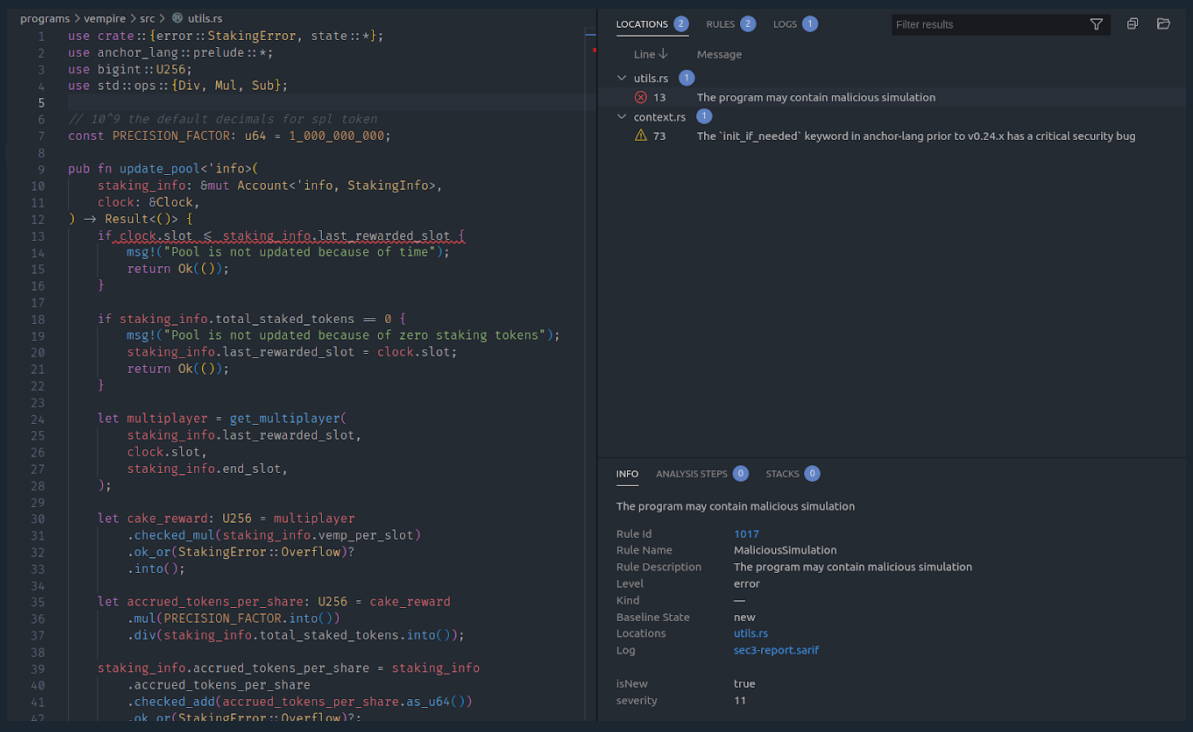
Critical Severity Issues

1. Detecting Simulations in a Solana Program:

Description:
As reported by the Sec3 Pro tool, an automated analyzer to run simulations and detect vulnerabilities in Rust/ Anchor based programs, we were able to find and report a critical issue the details for which can be found in the link.

https://opcodes.fr/en/publications/2022-01/detecting-transaction-simulation/

Recommendation:
A proper long term solution to the issue along with a considerable short term patch can be found in the link above so as to overcome the problem.



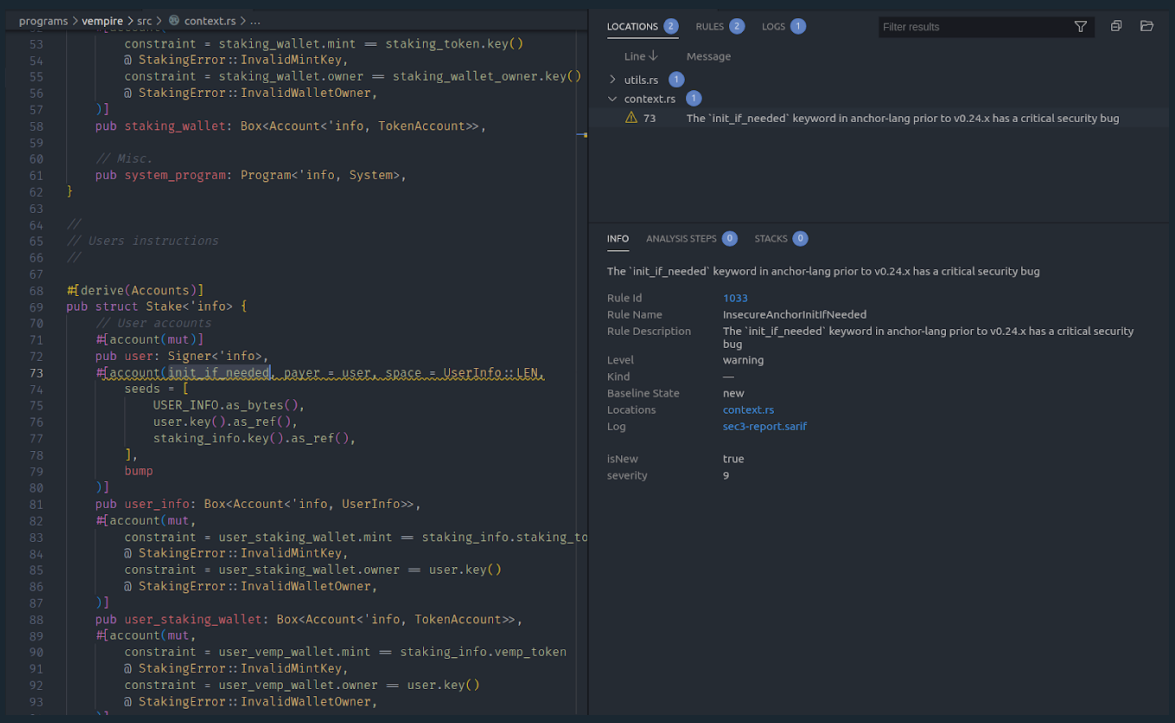
Amended (June 08th, 2022): The issue has been fixed by the vEmpire team and is no longer present in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

High Severity Issues

1. Anchor version <0.24.x containing exploitable handler

Description:
In the file programs/vempire/cargo.toml; on line 20 it is restricted that anchor-lang will use the version 0.23.0 and the feature of 'init-if-needed'. As reported by the Sec3 Pro tool, an automated analyzer to run simulations and detect vulnerabilities in Rust/ Anchor based programs, we were able to find and report a high risk issue the details for which can be found in the link.
https://twitter.com/armaniferrante/status/1516425854183690257

Recommendation:
The basic recommendation is to upgrade the anchor version to 0.24.x with the reasoning mentioned in the link above.



Amended (June 08th, 2022): The issue has been fixed by the vEmpire team and is no longer present in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

Medium Severity Issues

1. Insufficient error handling for the function initialize()

Description:
In the lib.rs [Lines 19-26] file; vempire contract, the function initialize() does not ensure that any invalid arbitrary values are sent as arguments, especially for start_slot and end_slot variables.
The test with invalid values for both variables passed initially, ultimately failing when calling the staking, unstaking, and withdrawing rewards functions.

Recommendation:
Place checks to ensure that always the valid values for both above-mentioned variables are supplied for the initialize() function. So that the following functions of Stake, unstake, and claiming rewards work as expected

Amended (June 08th, 2022): The issue has been fixed by the vEmpire team and is no longer present in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

Low Severity Issues

1. Missing overflow/underflow checks in cargo.toml

Description:
The arithmetics in utils.rs can be overflowed and/or underflowed pertaining to basic math. This can cause the rewards of vemp as primary and other tokens as secondary to be calculated wrongly.

Recommendation:
Use [overflow-checks = true] in cargo.toml to ensure the arithmetics in utils.rs stay protected.

Amended (June 08th, 2022): The issue has been fixed by the vEmpire team and is no longer present in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Recommendation / Informational

1. Outdated cargo dependencies

Description:

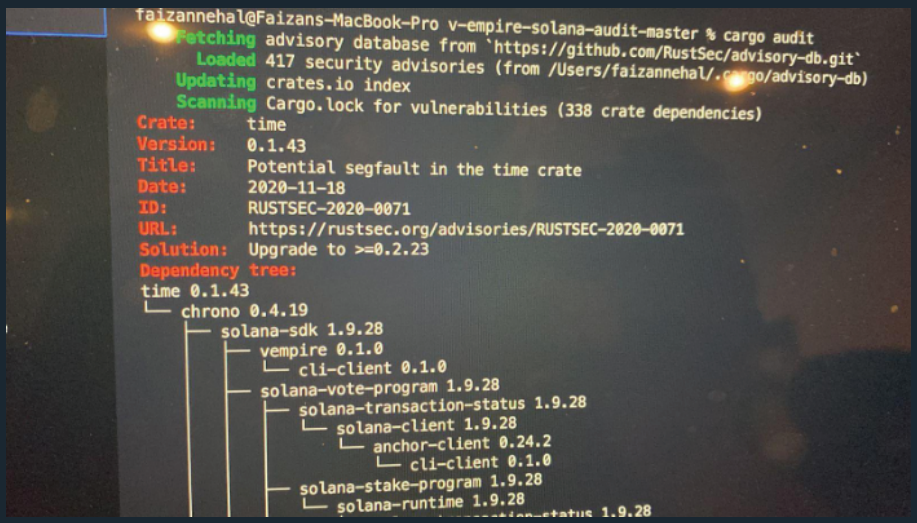
We used the security auditing automated tool named as cargo-audit that audits Cargo.lock files for crates with security vulnerabilities reported to the RustSec Advisory Database. The tool reported the following dependency issue.

Recommendation:
Use the latest cargo packages version for the reported issue

```
> cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 416 security advisories (from /Users/faizannehal/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (385 crate dependencies)
Crate:     nix
Version:   0.20.0
Title:     Out-of-bounds write in nix::unistd::getgrouplist
Date:      2021-09-27
ID:        RUSTSEC-2021-0119
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0119
Solution:  Upgrade to ^0.20.2 OR ^0.21.2 OR ^0.22.2 OR >=0.23.0
Dependency tree:
nix 0.20.0
├── solana-perf 1.8.14
│   └── solana-clap-utils 1.8.14
│       ├── solana-net-utils 1.8.14
│       │   └── solana-client 1.8.14
│       │       └── anchor-client 0.23.0
│       │           └── cli-client 0.1.0
│       ├── solana-faucet 1.8.14
│       │   └── solana-client 1.8.14
│       ├── solana-client 1.8.14
│       └── cli-client 0.1.0
└── solana-net-utils 1.8.14
```

Partially Closed (June 08th, 2022): The issue is partially fixed by the vEmpire team in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

Error shown during final audit.



2. Redundant Error Handling

Description:

It is redundant error handling in vempire/src/utils.rs. In line 30 there is a cake_reward variable. The code is utilizing checked_mul() function which already protects from the integer overflows and underflows but on the very next line the code is using the Overflow Error Handling .ok_or(StakingError::Overflow)?
This error handling is redundant because checked_mul() is already used. It is making the code less optimized.

Recommendation:
Omit the .ok_or(StakingError::Overflow)? Line from the code.

```
let cake_reward: U256 = multiplayer
        .checked_mul(staking_info.vemp_per_slot)
        .ok_or(StakingError::Overflow)?
        .into();
```

Amended (June 08th, 2022): The issue has been fixed by the vEmpire team and is no longer present in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

3. Lack Of Error Handling in Complex Math

Description:

There is a lack of underflow/overflow error handling in vempire/src/utils.rs. In line 35 in the accrued_tokens_per_share the code is utilizing the mul() function but there is no error handling in case of overflow

Recommendation:
We recommend either using the checked_mul() function or use the error handling .ok_or(StakingError::Overflow)?

```
let accrued_tokens_per_share: U256 = cake_reward
        .mul(PRECISION_FACTOR.into())
        .div(staking_info.total_staked_tokens.into());
```

4. Spelling mistakes and typos reducing code readability

Description:

In the files mentioned there are some typo errors that should be corrected to make the code as per the standard. In line 41, 62, 75 and 85 of the vempire/src/ state.rs contract there is a typo. It is written as DESCRIMINATOR_LEN, the correct word should be DISCRIMINATOR_LEN.

Recommendation:
Use the latest cargo packages version for the reported issue.

```
pub const LEN: usize = DESCRIMINATOR_LEN + 8 + 8;
```

Amended (June 08th, 2022): The issue has been fixed by the vEmpire team and is no longer present in the commit: 7285b95faf20ee9976cfea14b7f92c7e7f887878

Code Coverage Report

Back	/home/jarir/Work/audits/immunebytes/solana-audit-master/v-empire-solana-audit-master/programs/vempire/src	Covered: 154 of 365 (42.19%)

Path	Coverage
context.rs	0 / 0
error.rs	0 / 0
lib.rs	1 / 211 (0.47%)
state.rs	0 / 0
utils.rs	153 / 154 (99.35%)

Concluding Remarks

While conducting the audits of the vEmpire smart contract(s), it was observed that the contracts contain Critical, High, Medium, and Low severity issues.

Our auditors suggest that severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the vEmpire platform or its product nor this audit is investment advice.

Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes