Reign Of Terror

NFT Game

PROGRAM AUDIT

11.05.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of PROPELLENTS VENTURES PTE. LTD. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (17.04.2022)	Layout
0.4 (20.04.2022)	Automated Security Testing Manual Security Testing
0.5 (21.04.2022)	Verify Claims and Test Deployment
0.9 (24.04.2022)	Summary and Recommendation
1.0 (26.04.2022)	Final document
1.1 (04.05.2022)	Re-Check 948fb2d1c2e0e56b8afd69c8ef193599a6779fcf
1.2 (11.05.2022)	Re-Check 2cbc307bcbb5baabfd601ea7e58613d1d3515ecc

2. About the Project and Company

Company address:
PROPELLENTS VENTURES PTE. LTD.
10 ANSON ROAD #17-07
INTERNATIONAL PLAZA
SINGAPORE 079903

Website: https://reignofterror.io

Twitter: https://twitter.com/RoTTheGame

Medium: https://medium.com/@ReignofTerror

Discord: http://discord.gg/8AwGrpWBKA

Telegram: https://t.me/reignofterrornews

E-Mail: info@reddoor.digital

2.1 Project Overview

Red Door Digital focuses on developing great games that integrate blockchain to bring its players and the community more utility and value. Through this, they want to help gamers from around the world earn a living through P2E and bring about deeply engaged metaverse communities who play for fun and have shared social and financial incentives.

Reign of Terror is a cyberpunk metaverse P2E blockchain game created to immerse players in a dreamlike, futuristic world while weaving graspable decentralized finance mechanisms. Get lost playing in a jaw dropping new reality where missions become investment bearing activities. RoT utilizes NFTs in every facet of the game, is full of P2E missions, and also introduces players to DeFi mechanics as part of the gameplay, such as staking. SocialFi mechanics are also strong here with Guilds, Alliances, a DAO governance system and NFT lending system, creating a seamless layer of blockchain beneath the skin of a cyberpunk realm.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the program functioning in a number of scenarios, or creates a risk that the program may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a program, or provides the opportunity to use a program in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the program in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the program and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and rust program developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the program
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the programs to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your programs.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Program Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./rot-metagame-contract/src/lib.rs	32bbd9146f5aa443e0e5c7893e3e1df4
./rot-metagame-contract/src/anchor_metaplex.rs	6b77e9d3353e547af568eef0cca06b71

5.2 Used Code from other Frameworks/Dependencies

Dependency / Import Path	Source
anchor-lang	https://docs.rs/anchor-lang/latest/anchor_lang/
anchor-spl	https://docs.rs/anchor-spl/latest/anchor_spl/
mpl-token-metadata	https://docs.rs/mpl-token-metadata/latest/mpl_token_metadata/
static_assertions	https://docs.rs/static_assertions/latest/static_assertions/

6. Scope of Work

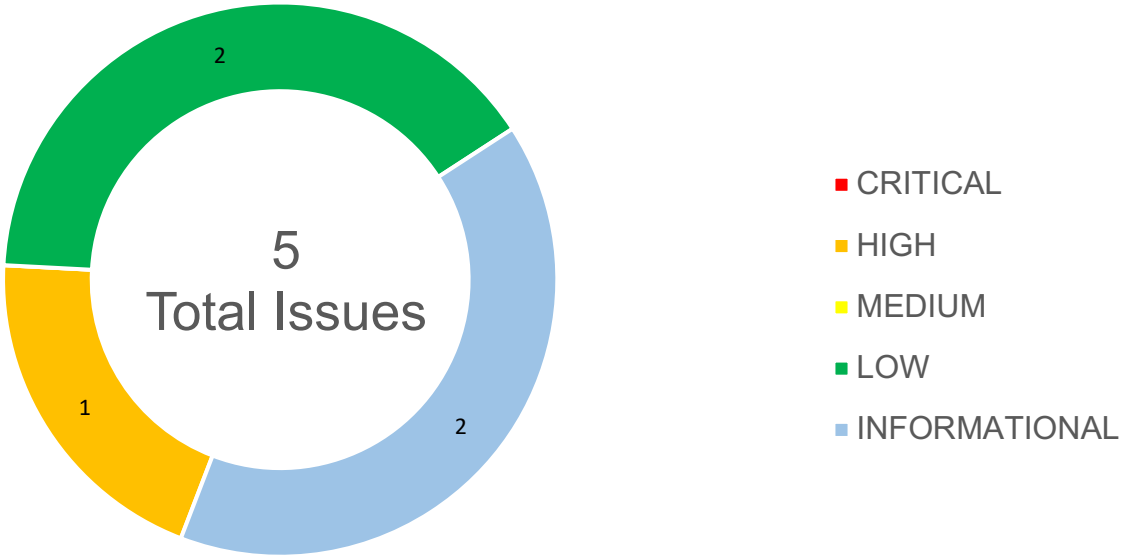
The Red Door Digital developer team, of the game Reign Of Terror provided us with the files that needs to be tested. The scope of the audit is the metagame program.

The team put forward the following assumptions regarding the security, usage of the program:

- The program is coded according to the newest standards and in a secure way.
- Rewards are correctly distributed
- Lock and unlock of NFTs is correctly working (10 days auto-unlock)

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	User Cannot Claim Rewards After claim_only_nft	HIGH	FIXED
6.2.2	Centralized Control Over Rewards	LOW	FIXED
6.2.3	Missing Account Verification	LOW	FIXED
6.2.4	Create transfer/close_account Functions To Avoid Code Duplication	INFORMATIONAL	FIXED
6.2.5	Dead Code	INFORMATIONAL	FIXED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting's experts found **0 Critical issues** in the code of the program.

HIGH ISSUES
During the audit, Chainsulting's experts found **1 High issue** in the code of the program.

6.2.1 User Cannot Claim Rewards After claim_only_nft
Severity: HIGH
Status: FIXED
Code: NA
File(s) affected: lib.rs
Update: ClaimOnlyNft has the Anchor constraint locked_nft_account.reward_mint == None while ClaimNftAndRewards has the Anchor constraint locked_nft_account.reward_mint.unwrap() == user_reward_account.mint (reward_mint is a Option<Pubkey>). So, the user will never be able to claim only the NFT if there are rewards. Nor will the user be able to claim rewards if there are none.

Attack / Description	The users can claim rewards after the admin unlocks their NFT or the auto_unlock_timestamp_secs reach the end. There is a functionality that allows the user to only claim the NFT, which gives them the NFT back and closes the escrow account. The issue occurs when the user calls claim_only_nft and tries to claim the rewards by calling. Especially in the transfer back NFT step, the program will not be able to find the nft_escrow_account. This will revert the transaction and the user will not be able to claim the rewards.
Code	Line 48 – 77 (lib.rs) `pub fn claim_only_nft(ctx: Context<ClaimOnlyNft>) -> Result<()> {` `  // Transfer NFT back to user` `  let transfer_cpi = Transfer {`

```
            from: ctx.accounts.nft_escrow_account.to_account_info().clone(),
            to: ctx.accounts.user_nft_account.to_account_info().clone(),
            authority: ctx.accounts.locked_nft_account.to_account_info().clone(),
        };
        let transfer_cpi_program = ctx.accounts.token_program.to_account_info();
        let seeds = [
            ctx.accounts.user.key.as_ref(),
            &[ctx.accounts.locked_nft_account.bump],
        ];
        token::transfer(
            CpiContext::new(transfer_cpi_program, transfer_cpi).with_signer(&[&seeds]),
            1,
        )?;

        // Close the escrow NFT associated token account and refund the rent to the user
        let close_cpi_accounts = CloseAccount {
            account: ctx.accounts.nft_escrow_account.to_account_info().clone(),
            destination: ctx.accounts.user.to_account_info().clone(),
            authority: ctx.accounts.locked_nft_account.to_account_info().clone(),
        };
        let close_cpi_program = ctx.accounts.token_program.to_account_info().clone();
        token::close_account(
            CpiContext::new(close_cpi_program, close_cpi_accounts).with_signer(&[&seeds]),
        )?;

        Ok(())
    }
```

Result/Recommendation	It is recommended to add a new boolean attribute, called nft_claimed to the LockedNft struct, that defaults to false and change it to true, after the user claims the NFT.

MEDIUM ISSUES
During the audit, Chainsulting's experts found **0 Medium issues** in the code of the program.

LOW ISSUES
During the audit, Chainsulting's experts found **2 Low issues** in the code of the program.

6.2.2 Centralized Control Over Rewards
Severity: LOW
Status: FIXED
Code: NA
File(s) affected: lib.rs
Update: This is by design. Users will lock an NFT before the game, play the game then get a (potential) reward at the end of the game. The specific rewards are decided by server-side at the end of the game. Note that the locking period is not necessarily constant. The 10 day auto-unlock is just a fail-safe. Games should last much less.

Attack / Description	The user rewards are not specified by default or calculated using the locking period. Instead, the admin is the one responsible to set the reward amount manually, by calling the setReward function. This represents a critical centralization issue, due to the super control that the admin has over the rewards.
Code	Line 134 – 151 (lib.rs) `pub fn set_reward(ctx: Context<SetReward>, reward_amount: u64) -> Result<()> {` `  require!(reward_amount > 0, Errors::RewardMustBeGreaterThanZero);` `  // Transfer reward_amount into reward escrow account` `  let transfer_cpi = Transfer {` `  from: ctx` `  .accounts` `  .reward_treasury_account`

	.to_account_info() .clone(), to: ctx.accounts.reward_escrow_account.to_account_info().clone(), authority: ctx.accounts.authority.to_account_info().clone(), }; let transfer_cpi_program = ctx.accounts.token_program.to_account_info(); token::transfer(CpiContext::new(transfer_cpi_program, transfer_cpi), reward_amount,)?;
Result/Recommendation	It is recommended to automate the process of reward distribution by setting a fixed value that every user will get, since the locking period is constant.

6.2.3 Missing Account Verification
Severity: LOW
Status: FIXED
Code: NA
File(s) affected: lib.rs
Update: It does not look to us that an address constraint on Program helps except as documentation. Anchor defines Program<'a, T> where T: Id and implements the anchor_lang::Id trait for Token/AssociatedToken/System. Program::try_from has an explicit check comparing info.key and &T::id() and will err with InvalidProgramId. Thus, any invalid key will fail with InvalidProgramId before it reaches a constaint check.

Attack / Description	Certain instructions lack a safety check in the account addresses, accounts that are passed to the transaction should all be verified. Otherwise, the program's functionality may get interpreted in a different way.
Code	Line 207 – 245 (lib.rs) pub struct LockNft<'info> { #[account(mut)]

```
    pub user: Signer<'info>,
    pub admin_settings: Account<'info, AdminSettings>,
    #[account(
        mut,
        associated_token::mint = nft_mint,
        associated_token::authority = user,
        constraint = user_nft_account.amount > 0 @ Errors::NftNotOwned,
    )]
    pub user_nft_account: Account<'info, TokenAccount>,
    pub nft_mint: Account<'info, Mint>,
    #[account(
        // Verify incoming metadata is for the NFT being locked
        constraint = nft_metadata.key() == find_metadata_account(&user_nft_account.mint).0 @
Errors::NftMetadataAndMintMismatch,
        // Check that incoming NFT is from proper collection and is verified
        constraint = nft_metadata.collection == Some(Collection {verified: true, key:
admin_settings.nft_collection}) @ Errors::NftFailedVerifiedCollectionCheck,
    )]
    pub nft_metadata: Box<Account<'info, MetaplexMetadata>>,
    #[account(
        init,
        payer = user,
        seeds = [user.key().as_ref()],
        bump,
        space = 8 + LOCKED_NFT_SIZE
        )]
    pub locked_nft_account: Account<'info, LockedNft>,
    #[account(
        init,
        payer = user,
        associated_token::mint = nft_mint,
        associated_token::authority = locked_nft_account
    )]
```

	`` pub nft_escrow_account: Account<'info, TokenAccount>, pub system_program: Program<'info, System>, pub token_program: Program<'info, Token>, pub associated_token_program: Program<'info, AssociatedToken>, pub rent: Sysvar<'info, Rent>, } ```
Result/Recommendation	It is recommended to verify all the accounts passed to the transaction, including the token and system accounts. The risk can be remediated using the following code : src/lib.rs #[account(address = token::ID)} pub token_program: Program<'info, Token>, #[account(address = system_program::ID)] pub system_program: Program<'info, System>,

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **2 Informational issues** in the code of the program.

6.2.4 Create transfer/close_account Functions To Avoid Code Duplication
Severity: INFORMATIONAL
Status: FIXED
Code: NA
File(s) affected: lib.rs

Update: We agree duplicate code is ugly and Rust is surprisingly verbose there. We will look into creating a helper function.

Attack / Description	Transfer and close_account instructions are used in almost every function.
Code	Line 30 - 35 (lib.rs) `let transfer_cpi = Transfer {` `from: ctx.accounts.user_nft_account.to_account_info().clone(),` `to: ctx.accounts.nft_escrow_account.to_account_info().clone(),` `authority: ctx.accounts.user.to_account_info().clone(),` `};` `let transfer_cpi_program = ctx.accounts.token_program.to_account_info();` `token::transfer(CpiContext::new(transfer_cpi_program, transfer_cpi), 1)?;`
Result/Recommendation	As best practice, it is recommended to create functions and call them as needed, to avoid code duplication and maintain a clean codebase.

6.2.5 Dead Code
Severity: INFORMATIONAL
Status: FIXED
Code: CWE-561
File(s) affected: lib.rs
Update: We agree. The final version should not have dead code.

Attack / Description	Commented-out code bloats programs and reduces readability.
Code	Line 188 - 203 (lib.rs) `/////` `/* Uncomment snippet to enforce only deployer can initialize the program` `// Note this makes testing on localnet more cumbersome, as cannot use`

	`` // --bpf-program flag to push program with set address on chain startup. // One could argue it is safer to enforce, but in practice it should make // no difference if we deploy and immediately call Initialize() ourselves. ///// #[account(constraint = this_program.programdata_address() == Some(program_data.key()))] pub this_program: Program<'info, crate::program::RotMetagameContract>, #[account(constraint = program_data.upgrade_authority_address == Some(signer.key()))] pub program_data: Account<'info, ProgramData> ///// End snippet */ ///// ```
Result/Recommendation	It is recommended to remove the commented-out or unused code; it can be retrieved from source control history if required.

6.3. Verify Claims

6.3.1 The program is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

6.3.2 Rewards are correctly distributed
Status: tested and verified ✅

6.3.3 Lock and unlock of NFTs is correctly working (10 days auto-unlock)
Status: tested and verified ✅

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the rust program.

The main goal of the audit was to verify the claims regarding the security of the program. During the audit, no critical, 1 high, 2 low and 2 informational issues have been found, after the manual and automated security testing. We advise Red Door Digital team to implement the recommendations contained in all our findings, to further enhance the code's security and readability.

Update (04th of May): The RedDoor Team addressed all issues and Chainsulting has provided further recommendation to issue 6.2.3

8. About the Auditor

Chainsulting is a professional software development firm based in Germany that provides comprehensive distributed ledger technology (DLT) solutions. Some of their services include blockchain development, smart contract audits and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Hyperledger, Tezos, Ethereum, Binance Smart Chain, and Solana to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, Amun, Furucombo among numerous other top DeFi projects.

Chainsulting currently secures $100 billion in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the blockchain sector to deliver top-notch smart contract audit solutions tailored to the clients' evolving business needs.

The blockchain security provider brings the highest security standards to crypto and blockchain platforms, helping to foster growth and transparency within the fast-growing ecosystem.

Check our website for further information: https://chainsulting.de