Phantasia Sports – NTF Store

Solana Program Security Audit

Prepared by: **Halborn**

Date of Engagement: **January 16th, 2022 - January 25th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/16/2022	Piotr Cielas
0.2	Document Edits	01/17/2022	Piotr Cielas
0.3	Document Edits	01/24/2022	Piotr Cielas
0.4	Final Draft	01/25/2022	Piotr Cielas
0.5	Final Draft Review	01/27/2022	Gabi Urrutia
1.0	Remediation Plan	01/31/2022	Isabel Burruezo
1.1	Remediation Plan Review	02/01/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Phantasia Sports is a Fantasy Sports platform that is built on top of the Solana Blockchain. On Phantasia Sports, anyone can earn tokens by contributing to the ecosystem through skilled gameplay. Players can join public or private tournaments to play fantasy sports against other users.

Phantasia Sports engaged Halborn to conduct a security assessment on their NTF Store program beginning on January 16th, 2022 and ending January 25th, 2022. This security assessment was scoped to the phantasia-nft-store-program repository and an audit of the security risk and implications regarding the changes introduced by the development team at Phantasia Sports prior to its production release shortly following the assessment's deadline.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned one full-time security engineer to audit the security of the program. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that program functions are intended.
- Identify potential security issues with the program.

In summary, Halborn identified some security risks that were mostly addressed by the Phantasia Sports team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of the platform.
- Manual code read and walkthrough.
- Manual Assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Fuzz testing. (Halborn custom fuzzing tool).
- Checking the test coverage. (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities.(cargo audit).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

This review was scoped to the NFT Store Solana program.

1. NFT Store program
    (a) Repository: phantasia-nft-store-program
    (b) Commit ID: 59c18ce9cfb4fa58ecf503c6bdbfb7e22996c9b9

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	2	4

LIKELIHOOD

IMPACT

				(HAL-01)
		(HAL-02)		
	(HAL-03) (HAL-04)			
(HAL-05) (HAL-06) (HAL-07) (HAL-08)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) ANONYMOUS SELL ORDER CANCELLING	High	SOLVED - 01/17/2022
(HAL-02) HARDCODED VAULT ADDRESS	Medium	ACKNOWLEDGED
(HAL-03) NFT ORDER TYPE MISMATCH	Low	SOLVED - 01/29/2022
(HAL-04) FANT AMOUNT CASTING OVERFLOW	Low	SOLVED - 01/29/2022
(HAL-05) EDITION SELL ORDER PARAMETER SANITY CHECK MISSING	Informational	SOLVED - 01/27/2022
(HAL-06) POSSIBLE MISUSE OF HELPER METHODS	Informational	SOLVED - 12/28/2021
(HAL-07) BUYING NFTS AS A DELEGATE	Informational	ACKNOWLEDGED
(HAL-08) ALL NFT EDITION SALE PROFITS ARE TRANSFERRED TO THE FEE VAULT	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) ANONYMOUS SELL ORDER CANCELLING - HIGH

Description:

Owners can place sell orders on their NFTs by sending either a ListNftForSale instruction or a ListEditionForSale instruction to the program. Either instruction handler transfers the token's authority to an account address derived from the program's ID and a static seed and creates an order account. This account stores the ask price, NFT mint and several other parameters.

Sellers may choose to cancel their orders before they're filled. To accomplish that, they can send either a CancelNftSale or a CancelEditionSale. Either instruction handler transfers the NFT's authority back to the seller (NFT owner) and closes the order account, sending the rent back to the seller.

However, because neither the CancelNftSale instruction handler nor the CancelEditionSale instruction handler verifies if the NFT owner is, in fact, a transaction signer, an anonymous attacker may cause a DoS of the program by cancelling all sell orders.

Code Location:

Listing 1: processor/cancel_listing.rs (Lines 22)

```
20 pub fn process_cancel_listing(accounts: &[AccountInfo], program_id
       : &Pubkey) -> ProgramResult {
21     let account_info_iter = &mut accounts.iter();
22     let seller_wallet_account = next_account_info(
           account_info_iter)?;
23     let selling_nft_token_account = next_account_info(
           account_info_iter)?;
24     let sell_order_data_storage_account = next_account_info(
           account_info_iter)?;
25     let nft_store_signer_pda_account = next_account_info(
           account_info_iter)?;
```

```
26      let token_program = next_account_info(account_info_iter)?;
```

Listing 2: processor/cancel_listing.rs (Lines 63)

```
63 if sell_order_data.seller_wallet != *seller_wallet_account.key {
64     msg!("PhantasiaError::SellerMismatched");
65     return Err(PhantasiaError::SellerMismatched.into());
66 }
```

Listing 3: processor/cancel_edition_listing.rs (Lines 25)

```
20 pub fn process_cancel_edition_listing(
21     accounts: &[AccountInfo],
22     program_id: &Pubkey,
23 ) -> ProgramResult {
24     let account_info_iter = &mut accounts.iter();
25     let seller_wallet_account = next_account_info(
           account_info_iter)?;
26     let selling_nft_token_account = next_account_info(
           account_info_iter)?;
27     let sell_order_data_storage_account = next_account_info(
           account_info_iter)?;
28     let nft_store_signer_pda_account = next_account_info(
           account_info_iter)?;
29     let token_program = next_account_info(account_info_iter)?;
```

The only check performed on the seller account by the instruction handler.

Listing 4: processor/cancel_edition_listing.rs (Lines 66)

```
66 if sell_order_data.seller_wallet != *seller_wallet_account.key {
67     msg!("PhantasiaError::SellerMismatched");
68     return Err(PhantasiaError::SellerMismatched.into());
69 }
```

Risk Level:

Likelihood - 5
Impact - 4

Recommendation:

Verify if the NFT owner (seller) is indeed the signer of CancelNftSale and CancelEditionSale instructions before updating the state.

Remediation Plan:

SOLVED: The Phantasia Sports team fixed this issue in commit 5a1b332897736200f6d793852891ac179144c48d: the transaction signer public key is verified to match the seller_wallet account address saved in the sale order.

3.2 (HAL-02) HARDCODED VAULT ADDRESS - MEDIUM

Description:

Rust compiler requires the program developer to set the TREASURY_ATA_PUBKEY environment variable to build the program. The BuyNft and BuyEdition instruction handlers use the value of this variable to get the fee vault address. Since this address is hardcoded at compile time, it cannot be modified without redeploying the program if the account compromised.

Code Location:

Listing 5: processor/buy_nft.rs (Lines 46)

```
45 let transaction_fee_fant_ata_id = Pubkey::from_str(env!(
46     "TREASURY_ATA_PUBKEY",
47     "Must specify a treasury ATA account public key!"
48 ))
49 .map_err(|_| PhantasiaError::StringToPubkeyConversionFailed)?;
50 if *transaction_fee_fant_ata.key != transaction_fee_fant_ata_id {
51     msg!(
52         "PhantasiaError::TransactionFeeFantAtaMismatched EXPECTED:
                {:?} ACTUAL: {:?}",
53         transaction_fee_fant_ata_id,
54         transaction_fee_fant_ata.key
55     );
56     return Err(PhantasiaError::TransactionFeeFantAtaMismatched.
           into());
57 }
```

Listing 6: processor/buy_edition.rs (Lines 58)

```
57 let transaction_fee_fant_ata_id = Pubkey::from_str(env!(
58     "TREASURY_ATA_PUBKEY",
59     "Must specify a treasury ATA account public key!"
60 ))
61 .map_err(|_| PhantasiaError::StringToPubkeyConversionFailed)?;
```

```
62 if *transaction_fee_fant_ata.key != transaction_fee_fant_ata_id {
63     msg!(
64         "PhantasiaError::TransactionFeeFantAtaMismatched EXPECTED:
               {:?} ACTUAL: {:?}",
65         transaction_fee_fant_ata_id,
66         transaction_fee_fant_ata.key
67     );
68     return Err(PhantasiaError::TransactionFeeFantAtaMismatched.
          into());
69 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Implement a governance function to update the fee vault address.

Remediation Plan:

ACKNOWLEDGED: The Phantasia Sports team accepts the risk of this finding.

3.3 (HAL-03) NFT ORDER TYPE MISMATCH - LOW

Description:

The program creates two types of NFT sell orders: SellOrder and LimitedEditionSale. NFTs may be either single- or multiple edition. Edition ID is stored in NFT metadata as edition_nonce. Editions are identified with u8 integer, but for single edition NFTs this property is set to None.

Neither the ListNftForSale nor the ListEditionForSale instruction handler checks the value of the edition_nonce property, which means the NFT owner may place a SellOrder order for a multiple edition NFT and a LimitedEditionSale order for a single edition NFT.

Code Location:

Listing 7: processor/list_nft.rs (Lines 158,161)

```
150 if !rent.is_exempt(
151     selling_nft_token_account.lamports(),
152     selling_nft_token_account.data_len(),
153 ) {
154     msg!("PhantasiaError::NotRentExempt");
155     return Err(PhantasiaError::NotRentExempt.into());
156 }
157
158 validate_nft(
159     selling_nft_mint_account,
160     selling_nft_token_account,
161     selling_nft_metadata_account,
162     seller_wallet_account.key,
163 )?;
```

Listing 8: processor/list_nft.rs (Lines 57)

```
43 pub fn validate_nft(
44     nft_mint: &AccountInfo,
45     nft_token_account: &AccountInfo,
46     metadata_info: &AccountInfo,
47     seller_wallet_pubkey: &Pubkey,
48 ) -> ProgramResult {
49     let verified_creators: [Pubkey; NUM_VERIFIED_CREATORS] = [
50         vc1_id::id(), // This is our main wallet
51         vc2_id::id(),
52         vc3_id::id(),
53         vc4_id::id(),
54         vc5_id::id(),
55         vc6_id::id(), // Phanbot creator
56     ];
57     validate_metadata_account(metadata_info, nft_mint.key)?;
58
59     if *nft_token_account.owner != spl_token::id() {
60         msg!("PhantasiaError::AccountOwnerShouldBeTokenProgram");
61         return Err(PhantasiaError::
               AccountOwnerShouldBeTokenProgram.into());
62     }
```

Listing 9: processor/list_nft.rs (Lines 57)

```
43 pub fn validate_nft(
44     nft_mint: &AccountInfo,
45     nft_token_account: &AccountInfo,
46     metadata_info: &AccountInfo,
47     seller_wallet_pubkey: &Pubkey,
48 ) -> ProgramResult {
49     let verified_creators: [Pubkey; NUM_VERIFIED_CREATORS] = [
50         vc1_id::id(), // This is our main wallet
51         vc2_id::id(),
52         vc3_id::id(),
53         vc4_id::id(),
54         vc5_id::id(),
55         vc6_id::id(), // Phanbot creator
56     ];
57     validate_metadata_account(metadata_info, nft_mint.key)?;
58
59     if *nft_token_account.owner != spl_token::id() {
60         msg!("PhantasiaError::AccountOwnerShouldBeTokenProgram");
```

```
61          return Err(PhantasiaError::
                AccountOwnerShouldBeTokenProgram.into());
62      }
```

Listing 10: processor/list_nft.rs

```
252 pub fn validate_metadata_account(metadata_info: &AccountInfo,
        nft_mint: &Pubkey) -> ProgramResult {
253     verify_metadata_account_owner(metadata_info.owner)?;
254     let metadata_program_id = metadata_program_id::id();
255     let metadata_seeds = &[
256         metaplex_state::PREFIX.as_bytes(),
257         metadata_program_id.as_ref(),
258         nft_mint.as_ref(),
259     ];
260     let (metadata_address, _) =
261         Pubkey::find_program_address(metadata_seeds, &
                metadata_program_id::id());
262
263     if metadata_address != *metadata_info.key {
264         msg!("PhantasiaError::MetadataAccountMismatched");
265         return Err(PhantasiaError::MetadataAccountMismatched.into
                ());
266     }
267     Ok(())
268 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Check the edition_nonce value before creating and placing sell orders.

Remediation Plan:

SOLVED: The Phantasia Sports team fixed this issue in commit 6b36c56025ae1ed8bfc869dc7015413a25e1b8b3: the value of edition_nonce is checked in the validate_nft function, used by ListNftForSale and ListEditionForSale, to verify that it is expected sales order type.

3.4 (HAL-04) FANT AMOUNT CASTING OVERFLOW - LOW

Description:

The program calculates the FANT amount the seller receives with the `get_fants_to_receive_from_basis_points` function. This function takes a `u64 selling_price` and `u16 basis_points` and performs some arithmetic operations on them. The result is cast to `u64` to match the type of the `amount` field of SPL Token's `Account` struct.

Before casting however the result is not verified to not exceed the maximum value allowed by the `u64` type which can lead to integer overflow.

Code Location:

Listing 11: processor/buy_nft.rs (Lines 206)

```
198 pub fn get_fants_to_receive_from_basis_points(
199     selling_price: u64,
200     basis_points: u16,
201 ) -> Result<u64, ProgramError> {
202     let amount_to_receive = (selling_price as u128)
203         .checked_mul(basis_points as u128)
204         .ok_or(PhantasiaError::MathOverflow)?
205         .checked_div(10000u128)
206         .ok_or(PhantasiaError::MathOverflow)? as u64;
207     return Ok(amount_to_receive);
208 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Verify amount_to_receive does not exceed u64::MAX before casting it to u64.

Remediation Plan:

SOLVED: The Phantasia Sports team fixed this issue in commit 6b36c56025ae1ed8bfc869dc7015413a25e1b8b3: the amount_to_receive is verified to not exceed the maximum value allowed by the u64 type to avoid an integer overflow.

3.5 (HAL-05) EDITION SELL ORDER PARAMETER SANITY CHECK MISSING - INFORMATIONAL

Description:

The ListEditionForSale instruction handler requires the NFT owner to provide some parameters, including number_to_sell and num_sold. The former denotes the total number of NTFs available, and the latter stores the number of NFTs sold to date.

The handler does not verify if num_sold is lower than number_to_sell which means it is possible to place a sell order on an NFT edition where the number of sold tokens exceeds the number of tokens available for sale.

Code Location:

Listing 12: processor/list_edition.rs (Lines 23,24)

```
19 pub fn process_list_edition_nft(
20     accounts: &[AccountInfo],
21     program_id: &Pubkey,
22     sale_price: u64,
23     number_to_sell: u16,
24     num_sold: u16
25 ) -> ProgramResult {
26     let account_info_iter = &mut accounts.iter();
27     let seller_wallet_account = next_account_info(
           account_info_iter)?;
```

Listing 13: processor/list_edition.rs (Lines 118,119)

```
113 let sell_order_data: LimitedEditionSale = LimitedEditionSale {
114     acc_type: AccTypesWithVersion::LimitedEditionSaleDataV1 as u8,
115     seller_wallet: *seller_wallet_account.key,
116     nft_token: *selling_nft_token_account.key,
117     selling_price: sale_price,
118     number_purchased: num_sold,
```

```
119     number_to_sell: number_to_sell,
120     nonce: bump_seed,
121 };
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Verify num_sold is a lower number than number_to_sell before placing the sell order.

Remediation Plan:

SOLVED: The Phantasia Sports team fixed this issue in commit 32ff199c83cdedf061aaeef706c75c9a4c7ed510:
the num_sold is verified not to exceed the number of available tokens, number_to_sell.

3.6 (HAL-06) POSSIBLE MISUSE OF HELPER METHODS - INFORMATIONAL

Description:

The intention and use of helper methods in Rust, like unwrap, is very useful for testing environments because a value is forcibly demanded to get an error (aka panic!) if the Option the methods is called on doesn't have Some value or Result. Nevertheless, leaving unwrap functions in production environments is a bad practice because not only will this cause the program to crash out, or panic!. In addition, no helpful messages are shown to help the user solve, or understand the reason of the error.

Code Location:

Listing 14: processor/list_nft.rs (Lines 103)

```
102 let metadata = Metadata::from_account_info(metadata_info)?;
103 let nft_creators = metadata.data.creators.unwrap();
104 let mut is_fake_nft = true;
105 for creator in nft_creators {
106     if creator.verified && is_verified_creator(&creator.address, &
        verified_creators)? {
107         is_fake_nft = false;
108         break;
109     }
110 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended not use the unwrap function in production environment because this use provokes panic! and may crash the contract without verbose error messages. Crashing the system will result in a loss of availability, and in some cases, even private information stored in the state. Some alternatives are possible, such as propagating the error with ? instead of unwrap, or using the error-chain crate for errors.

Remediation Plan:

SOLVED: The Phantasia Sports team fixed this issue in commit 68463a47fdd1a9be18b14ead9891806cb96be9d9: any use of the unwrap function has been removed and replaced by more secure methods such as error propagation.

3.7 (HAL-07) BUYING NFTS AS A DELEGATE - INFORMATIONAL

Description:

Both BuyNft and BuyEdition instruction handlers expect the transaction sender to provide the buyer_nft_fant_ata FANT token account to charge and buyer_wallet_account to authorize the transfer.
If it is successful, the NFT's authority is transferred to buyer_wallet_account.

In the SPL Token program, an account delegate is defined as an account authorized to transfer an owner-approved share of tokens out of the account.

Neither the BuyNft nor BuyEdition instruction handler verifies if buyer_wallet_account is the owner of the buyer_nft_fant_ata account rather than just a delegate which means if the transaction is signed by a delegate, the NFT's authority will be transferred to the delegate instead of the buyer_nft_fant_ata account owner.

Code Location:

Listing 15: processor/buy_nft.rs (Lines 29,30)

```
27 pub fn process_buy_nft(accounts: &[AccountInfo], program_id: &
       Pubkey) -> ProgramResult {
28     let account_info_iter = &mut accounts.iter();
29     let buyer_wallet_account = next_account_info(account_info_iter
           )?;
30     let buyer_nft_fant_ata = next_account_info(account_info_iter)
           ?;
```

Listing 16: processor/buy_nft.rs (Lines 40)

```
40 if !buyer_wallet_account.is_signer {
41         msg!("ProgramError::MissingRequiredSignature");
42         return Err(ProgramError::MissingRequiredSignature);
43     }
```

the only check on buyer_wallet_account

Listing 17: processor/buy_nft.rs (Lines 40)

```
40 if !buyer_wallet_account.is_signer {
41         msg!("ProgramError::MissingRequiredSignature");
42         return Err(ProgramError::MissingRequiredSignature);
43     }
```

Listing 18: processor/buy_edition.rs (Lines 36,47)

```
35 let mint_authority_info = next_account_info(account_info_iter)?;
36 let buyer_wallet_account = next_account_info(account_info_iter)?;
37 let nft_store_signer_pda_account = next_account_info(
       account_info_iter)?;
38 let master_edition_token_account_info = next_account_info(
       account_info_iter)?;
39 let update_authority_info = next_account_info(account_info_iter)?;
40 let master_metadata_account_info = next_account_info(
       account_info_iter)?;
41 let token_program_account_info = next_account_info(
       account_info_iter)?;
42 let system_account_info = next_account_info(account_info_iter)?;
43 let rent_info = next_account_info(account_info_iter)?;
44 let metadata_program_id_account = next_account_info(
       account_info_iter)?;
45 let metadata_mint_info = next_account_info(account_info_iter)?;
46
47 let buyer_nft_fant_ata = next_account_info(account_info_iter)?;
```

the only check on buyer_wallet_account

Listing 19: processor/buy_edition.rs (Lines 52)

```
52 if !buyer_wallet_account.is_signer {
53         msg!("ProgramError::MissingRequiredSignature");
54         return Err(ProgramError::MissingRequiredSignature);
55     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Verify the buyer_wallet_account to be the owner of the buyer_nft_fant_ata account before transferring NFT authority.

Remediation Plan:

ACKNOWLEDGED: The Phantasia Sports team do not believe that any of the platform users are going to ever use delegation to purchase NFTs from the store.

3.8 (HAL-08) ALL NFT EDITION SALE PROFITS ARE TRANSFERRED TO THE FEE VAULT - INFORMATIONAL

Description:

NFT owners can post sell orders on NFTs or NFT editions. When a user buys a single-edition NFT, a fixed percentage is deducted from the ask price and transferred to the fee vault. When a user buys an NFT edition, however, 100% of the ask price is transferred to the fee vault and the NFT owner receives nothing.

Code Location:

Listing 20: processor/buy_edition.rs (Lines 135)

```
128 let seller_to_receive_fants = sell_order_data.selling_price;
129
130 msg!("Calling the token program to transfer FANT to Treasury...");
131 invoke(
132     &spl_token::instruction::transfer(
133         token_program_account_info.key,
134         buyer_nft_fant_ata.key,
135         transaction_fee_fant_ata.key,
136         buyer_wallet_account.key,
137         &[],
138         seller_to_receive_fants,
139     )?,
140     &[
141         buyer_nft_fant_ata.clone(),
142         transaction_fee_fant_ata.clone(),
143         buyer_wallet_account.clone(),
144         token_program_account_info.clone(),
145     ],
146 )?;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Restrict access to the ListEditionForSale instruction handler to a wallet that owns the fee vault.

Remediation Plan:

ACKNOWLEDGED: The Phantasia Sports team states that the idea is that edition sales will be only performed by the team and would like all proceeds to be sent directly to the treasury.

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Results:

ID	package	Short Description
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations

THANK YOU FOR CHOOSING

// HALBORN