Bonfida - SNS

Solana Program Security Assessment

Prepared by: **Halborn**

Date of Engagement: **October 30th, 2023 - December 8th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	12/08/2023
0.2	Draft Version	12/08/2023
0.3	Draft Review	12/11/2023
0.4	Draft Review	12/11/2023
1.0	Remediation Plan	12/11/2023
1.1	Remediation Plan Review	12/14/2023
1.2	Remediation Plan Review	12/15/2023

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

The Bonfida program suite provides rich functionality related to the creation, tokenization, purchasing, and selling of domain names. Cross-chain functionality is also enabled via Wormhole with Binance smart chain, which allows for domains information to be used beyond the Solana blockchain.

Bonfida engaged Halborn to conduct a security assessment on their Solana programs, beginning on October 30th, 2023 and ending on December 8th, 2023 . The security assessment was scoped to the programs provided in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided 6 weeks for the engagement and assigned 1 full-time security engineer to review the security of the programs in scope. The security 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 assessment is to:

- Identify potential security issues within the programs

In summary, Halborn identified some security risks that were addressed and acknowledged by Bonfida .

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, 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 assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Finding unsafe Rust code usage (cargo-geiger)
- Scanning dependencies for known vulnerabilities (cargo audit).
- Local runtime testing (solana-test-framework)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope (s)	Changed (S:C)	1.25
	Unchanged (S:U)	1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Project Name

- Repository: name-tokenizer
- Commit ID: 9f922f39b
- Programs in scope:
    1. name-tokenizer (name-tokenizer/program/..)

1. Project Name

- Repository: sns-categories
- Commit ID: 36b6eeeac4
- Programs in scope:
    1. sns-categories (sns-categories/program/..)

1. Project Name

- Repository: name-offers
- Commit ID: b60700f732
- Programs in scope:
    1. name-offers (name-offers/program/..)

1. Project Name

- Repository: sns-registrar
- Commit ID: e2a81db7b2
- Programs in scope:
    1. sns-registrar (sns-registrar/program/..)

1. Project Name

- Repository: sns-warp
- Commit ID: 2b7e8194c
- Programs in scope:
    1. sns-warp (sns-warp/emitter/program..)

Out-of-scope:

- third-party libraries and dependencies
- financial-related attacks

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) FRONTRUNNING IN THE BuyFixedPrice INSTRUCTION HANDLER	Medium (6.7)	SOLVED - 11/21/2023
(HAL-02) LACK OF TOKEN BALANCE CHECKS IN WITHDRAWTOKEN COULD LEAD TO LOSS OF FUNDS	Informational (1.2)	ACKNOWLEDGED
(HAL-03) HARDCODED AUTHORITY ADDRESS WITHOUT UPDATE FUNCTIONALITY	Informational (1.2)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) FRONTRUNNING IN THE BuyFixedPrice INSTRUCTION HANDLER - MEDIUM (6.7)

Description:

In the MakeFixedPrice and BuyFixedPrice instructions, a FixedPriceOffer account is used to store data for a given listing. The amount and mint of tokens the purchaser will pay for the domain is included in this account. When the MakeFixedPrice instruction handler is called with a FixedPriceOffer account that already exists, the offer owner can modify the offer details. This allows the offer owner to change pricing data for the domain listing.

When BuyFixedPrice is called to purchase a domain account, the same FixedPriceOffer account is used. The caller pays offer_amount of quote_mint in exchange for the domain in question. However, this call can be frontrun by the offer_owner. Specifically, the offer_owner can frontrun the purchaser and change the offer_amount such that they are requesting way more tokens than before. Since the same account will be used for the purchasers transaction and there are no pricing checks, the transaction will succeed even with the unexpected new price. The result is a offer_owner who can steal funds from the purchaser of their domain listing via an unexpected price increase.

Code Location:

Listing 1: program/src/processor/buy_fixed_price.rs (Line 31)

```
27 #[derive(InstructionsAccount)]
28 pub struct Accounts<'a, T> {
29     /// Account of the fixed price offer
30     #[cons(writable)]
31     pub fixed_price_offer: &'a T,
32
33     /// Buyer of the fixed price offer
34     #[cons(signer)]
```

```rust
    pub buyer: &'a T,

    /// The domain name bought
    #[cons(writable)]
    pub name_account: &'a T,

    /// The destination token account
    #[cons(writable)]
    pub token_destination: &'a T,

    /// The source token account
    #[cons(writable)]
    pub token_source: &'a T,

    /// The fee account
    #[cons(writable)]
    pub fee_account: &'a T,

    /// The SPL token program ID
    pub spl_token_program: &'a T,

    /// The name service program ID
    pub name_service_program: &'a T,

    /// The system program ID
    pub system_program: &'a T,

    /// FIDA discount token account (belongs to the seller!)
    pub discount_account: Option<&'a T>,
}
```

Listing 2: program/src/processor/buy_fixed_price.rs (Line 160)

```rust
156 // Transfer native SOL
157 let ix = system_instruction::transfer(
158     accounts.buyer.key,
159     accounts.token_destination.key,
160     fixed_price_offer.offer_amount.checked_sub(fees).unwrap(),
161 );
162 invoke(
163     &ix,
164     &[
165         accounts.system_program.clone(),
166         accounts.buyer.clone(),
```

```
            accounts.token_destination.clone(),
        ],
)?;
```

Proof Of Concept:

The setup for this vulnerability includes a FixedPriceOffer account which has been created by some offer_owner. A purchaser is also required, who makes the call to buy-fixed-price. When the purchaser calls buy_fixed_price, the seller makes a second call to make_fixed_price, and alters the offer_amount field to a higher value. If the block is produced by a Jito validator, the seller can pay a bribe to get their transaction performed first. Since the same fixed_price_offer account is used in the purchasers call to buy_fixed_price, their call will succeed, even though the offer_amount has increased without their knowledge

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (6.7)

Recommendation:

Include expected price functionality in the buy_fixed_price function, so the purchaser cannot spend more than expected_amount in their transaction to purchase the domain.

Remediation Plan:

SOLVED: The Bonfida team solved this issue in commit 6d451eb by introducing an expected_price parameter, such that the buyer does not pay more than they intended for a tokenized domain.

4.2 (HAL-02) LACK OF TOKEN BALANCE CHECKS IN WITHDRAWTOKEN COULD LEAD TO LOSS OF FUNDS - INFORMATIONAL (1.2)

Description:

The name-tokenizer program includes functionality that allows an NFT holder to withdraw tokens that have been sent to the record account of the NFT they own. However, this introduces edge cases that could leave a tokenized domain holder to accidentally transfer ownership of these tokens during the sale of their NFT. In the event a tokenized domain is being sold, it would not be possible to check the account balances of all possible associated token accounts within the instruction itself. This leads to the edge case where a user is sent tokens, but accidentally sells or transfers their NFT before the tokens are realized. These tokens could then be claimed by the new owner of the NFT.

Code Location:

Listing 3: name-tokenizer/program/src/processor/withdraw_tokens.rs (Lines 109-110)

```
// NFT record is active -> Correct owner is the token holder
// NFT record is inactive -> Correct owner is the latest person
↳ who redeemed

pub fn process(program_id: &Pubkey, accounts: &[AccountInfo]) ->
↳ ProgramResult {
    let accounts = Accounts::parse(accounts, program_id)?;

    let mut NFT_record = NftRecord::from_account_info(accounts.
↳ NFT_record, Tag::ActiveRecord)
        .or_else(|_| NftRecord::from_account_info(accounts.
↳ NFT_record, Tag::InactiveRecord))?;

    let NFT = Account::unpack(&accounts.NFT.data.borrow())?;
```

```

    if NFT.mint != NFT_record.NFT_mint {
        msg!("+ NFT mint mismatch");
        return Err(ProgramError::InvalidArgument);
    }

    if NFT_record.is_active() {
        check_account_key(accounts.NFT_owner, &NFT.owner)?;
        if NFT.amount != 1 {
            msg!("+ Invalid NFT amount, received {}", NFT.amount);
            return Err(ProgramError::InvalidArgument);
        }

```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:L/Y:N/R:P/S:U (1.2)

Recommendation:

Ensure this feature is brought to the attention of users, such that NFTs capable of withdrawing tokens are not accidentally sold during NFT transfer to an address not under their control.

Remediation Plan:

ACKNOWLEDGED: The Bonfida team acknowledged this finding.

4.3 (HAL-03) HARDCODED AUTHORITY ADDRESS WITHOUT UPDATE FUNCTIONALITY - INFORMATIONAL (1.2)

Description:

The sns-warp change_fee instruction allows the program authority to update fee settings. However, this account is hardcoded as a constant, and there is no functionality to update this authority beyond deploying an updated program.

Code Location:

Listing 4: sns-warp/emitter/program/processor/change_fee.rs (Line 69)

```
65 check_account_key(a.central_state, &crate::central_state::KEY)?;
66 check_account_key(a.system_program, &system_program::ID)?;
67
68 #[cfg(not(feature = "no-authority"))]
69 check_account_key(a.authority, &AUTHORITY)?;
70
71 check_signer(a.authority)?;
72 check_signer(a.payer)?;
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:P/S:U (1.2)

Recommendation:

Include functionality to update the AUTHORITY address such that a program redeployment is not necessary.

Remediation Plan:

ACKNOWLEDGED: The Bonfida team acknowledged this finding.

MANUAL TESTING

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

Scenario	Expected Result	Evaluation
sns-registrar create accepts price data even if pyth account status field != trading	Tx fails	Pass
Can pass in a category-offer pda that corresponds to a fixed-price-offer pda (aka derivation using same seeds)	Tx fails	Pass
Possible to submit same instruction twice to sns-warp	Tx fails	Pass
Possible to pass in offer account with unexpected nonce in name-offers	Tx fails	Pass
Account other than initial offer creator can reinitialize offer	Tx fails	Pass
Account other than owner of domain can make an offer on their behalf	Tx fails	Pass
Possible to accept an offer that has been cancelled	Tx fails	Pass
Possible for account other than authority to change fees in sns-warp	Tx fails	Pass
Possible to create multiple simultaneous listings, each for different mint, that are not cancelled upon sale	Tx fails to create multiple listings	Pass
Possible to accept the same offer more than one time	Tx fails	Pass
Possible to provide invalid attestation signatures in SNS.sol	Tx fails	Pass
Possible to transfer NFT after it has been revoked	Tx fails	Pass

Possible to tokenize name_account on behalf of another user	Tx fails	Pass
Possible to create domain name with . seperator as part of name	Tx fails	Pass
Possible to create name account using name that is already in use	Tx fails	Pass
Possible to call sns-warp/post with domain that is not .sol	Tx fails	Pass
Possible to set favorite_account on behalf of another user	Tx fails	Pass
Can create both the domains Halborn and halborn	Tx fails to create second domain	Pass

5.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was cargo-audit, a security scanner for vulnerabilities reported to the Rust-Sec 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 reviewers 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-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on ‘ed25519-dalek‘
RUSTSEC-2023-0063	quinn-proto	Denial of service in Quinn servers
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2020-0072	tokio	Task dropped in wrong thread when aborting ‘LocalSet‘ task
RUSTSEC-2021-0124	tokio	Data race when sending and receiving after closing a ‘oneshot‘ channel
RUSTSEC-2023-0052	webpki	webpki: CPU denial of service in certificate path building
RUSTSEC-2023-0065	tungstenite	Tungstenite allows remote attackers to cause a denial of service

THANK YOU FOR CHOOSING

HALBORN