



Project: Anonymous DeFi Protocol
Backend: Solana - rust
Audit Date: 10 Sep 2021 - 14 Nov 2021

SecureContract

Contact: hello@securecontract.io

	DOCUMENT REVISION HISTORY	4
	CONTACTS	4
1	EXECUTIVE OVERVIEW	5
1.1	INTRODUCTION	6
1.2	AUDIT SUMMARY	6
1.3	TEST APPROACH & METHODOLOGY	7
	RISK METHODOLOGY	7
1.4	SCOPE	9
2	ASSESSMENT SUMMARY & FINDINGS OVERVIEW	10
3	FINDINGS & TECH DETAILS	11
3.1	(VULN-01) OUTDATED DEPENDENCY - LOW	13
	Description	13
	Code Location	13
	Risk Level	13
	Recommendation	13
	Remediation Plan	14
3.2	(VULN-02) ARITHMETIC ERRORS - LOW	15
	Description	15
	Code Location	15
	Recommendation	16
	Reference	16
	Remediation Plan	16
3.3	(VULN-03) UNSAFE RUST CODE USAGE - INFORMATIONAL	17

	Description	17
	Result	18
	Risk Level	19
	Recommendation	19
	Remediation Plan	19
3.4	(VULN-04) LOW TEST COVERAGE - INFORMATIONAL	20
	Description	20
	Result	20
	Risk Level	20
	Recommendation	21
	Remediation Plan	21
4	MANUAL TESTING	22
	Description	23
4.1	WITHDRAW AMOUNT ON BEHALF OF OTHER USER	23
	Description	23
	Code Location	23
	Results	24
4.2	BORROW AMOUNT ON BEHALF OF OTHER USER	25
	Description	25
	Code Location	25
	Results	26
5	FUZZING	26
5.1	FUZZING UNSAFE RUST DEPENDENCIES	28

	Description	28
	Results	28
6	AUTOMATED TESTING	29
6.1	VULNERABILITIES AUTOMATIC DETECTION	31
	Description	31
	Results	32
6.2	UNSAFE RUST CODE DETECTION	33
	Description	33
	Results	34
6.3	RUST UNDEFINED BEHAVIOUR TESTING	35
	Description	35
	Results	35
	References	36

DOCUMENT REVISION

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/29/2021	Mo Ashouri
0.5	Document Edit	10/02/2021	Florian Bujack
0.9	Document Edit	10/07/2021	Doreen Ahr
1.0	Final Version	10/11/2021	Mo Ashouri
1.1	Remediation Plan	10/14/2021	Florian Bujack

CONTACT	COMPANY	EMAIL
Mo Ashouri	SecureContract	ashourics@securecontract.com
Florian Bujack	SecureContract	florian@securecontract.io
Doreen Ahr	SecureContract	doreenahr@securecontract.io



EXECUTIVE OVERVIEW



1.1 INTRODUCTION

The X Protocol is a DeFi network built on Solana that will include the stablecoin XAI, a non-custodial lending market, and a margin trading vAMM. These are all use cases designed to solve one single problem: making value locked in DeFi systems accessible.

X engaged SecureContract to conduct a security assessment on their Smart contracts beginning on September 10, 2021. The security assessment was scoped to the smart contract provided in a Github repository and an audit of the security risk and implications regarding the changes introduced by the development team at X prior to its production release shortly following the assessments deadline.

1.2 AUDIT SUMMARY

The team at SecureContract was provided several weeks for the engagement and assigned two full time security engineers to audit the security of the smart contract. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart contract development.

1.3 TEST APPROACH & METHODOLOGY

SecureContract 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 smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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 review and walkthrough.
- Manual assessment of use and safety for the critical Rust variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual assessment to determine access control issues such as missing ownership checks, missing signer checks, and solana account confusions.
- Fuzz testing. (SecureContract custom fuzzing tool)
- Checking the test coverage. (cargo tarpaulin)
- Scanning of Rust files for vulnerabilities. (cargo audit)
- Detecting usage of unsafe Rust code. (cargo-geiger)
- Detecting Rust undefined behavior. (Miri)

RISK METHODOLOGY:

Vulnerabilities or issues observed by SecureContract 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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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

Code into: <https://github.com/—>

[aff638e60c4fe72d2393cfa4c9f24d14b8386f1](#)/programs/x/src folder.

Specific Commit Of Platform: commit [aff638e60c4fe72d2393cfa4c9f24d14b8386f1](#)

OUT-OF-SCOPE:

Other smart contracts in the repository and economics attacks.

Third party connections/connectivity.

2. ASSESSMENT SUMMARY & FINDINGS

		CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
		0	0	0	2	2
		LIKELIHOOD				
IMPACT						
	Vuln -03					
	Vuln -04					

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
OUTDATED DEPENDENCY	Low	RISK ACCEPTED
ARITHMETIC ISSUES	Low	RISK ACCEPTED
UNSAFE RUST CODE USAGE	Informational	ACKNOWLEDGED
LOW TEST COVERAGE	Informational	ACKNOWLEDGED



FINDINGS & TECH DETAILS

3.1 (VULN-01) OUTDATED DEPENDENCY LOW

Description:

PartyX is using `solana` dependency version `1.6.3` which is not the last `solana` version. Also `anchor` version is too old compare to the latest one `0.13.x`. Crate for fixed point number which is `fixed` is also old. It is always recommended to use the latest solana program version to avoid already fixed issues.

Code Location:

Listing 1: Cargo.toml (Lines 22,23,24,25)

```
21 [dependencies]
2   fixed = "1.7.0"
2
2   anchor-lang = "0.5.0"
3
2   anchor-spl = "0.5.0"
4
2   solana-program = "1.6.3"
5
```

Risk Level:

Likelihood - 2

Impact - 3

Recommendation:

SecureContract recommends to use the latest possible version of solana dependency (`0.7.4` at the moment of this audit) unless rust programs are bounded to specific versions. Also use the latest version for `fixed` and `anchor` crates.

Remediation Plan:

RISK ACCEPTED: X team is not considering to upgrade the anchor crate at the moment.

3.2 (VULN-02) ARITHMETIC ERRORS

LOW

Description:

The most serious arithmetic errors include **integer overflow/underflow**. In computer programming, integer overflow/underflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum representable value. Although integer overflows and underflows do not cause Rust to panic in the release mode, the consequences could be dire if the result of those operations is used in financial calculations.

Code Location:

Integer Overflow/Underflow

Listing 2: math.rs

```
27         let delta_decimal =
28             price.price_decimal as i16 + price.token_decimal as i16 -
                price.bid_token_decimal as i16;
```

Listing 3: math.rs

```
63         let delta_decimal =
64             price.price_decimal as i16 + price.token_decimal as i16 -
                price.bid_token_decimal as i16;
```

Division

Listing 4: math.rs

```
34         if delta_decimal > 0 {
35             collateral_amount_in_debt_token =
                collateral_amount_in_debt_token.div(ten_exponent)
```

```
36     } else {
```

Listing 5: math.rs


```

75         } else {
76             fp_repay_collateral_amount = fp_repay_collateral_amount.
                div(ten_exponent);
77         }
78         fp_repay_collateral_amount = fp_repay_collateral_amount.div(
            Fix::from_num(price.price));

```

Recommendation:

It is recommended to use vetted safe math libraries (like `checked_add`, `checked_div`) for arithmetic operations consistently throughout the smart contract system. Consider using Rust safe arithmetic functions for primitives rather than standard arithmetic operators.

Reference:

Safe arithmetic operations for primitives: `u8`, `u32`, `u64`

Remediation Plan:

RISK ACCEPTED: `X team` considers acceptable the arithmetic in this context, because the debt type owners chooses the asset types, so decimals will not overflow.

3.3 (VULN-03) UNSAFE RUST CODE USAGE - INFORMATIONAL

Description:

Rust code that uses the `unsafe` keyword is considered unsafe since all of the memory safety guarantees of Rust are not enforced there. It means that the code might be prone to vulnerabilities that would've been prevented by the compiler such as Buffer overflow, Double free, Use After free, and more.

Result:

Metric output format: x/y
 x = unsafe code used by the build
 y = total unsafe code found in the crate

Symbols:
 🛡️ = No `unsafe` usage found, declares `#![forbid(unsafe_code)]`
 ? = No `unsafe` usage found, missing `#![forbid(unsafe_code)]`
 ☹️ = `unsafe` usage found

Functions	Expressions	Impls	Traits	Methods	Dependency
0/1	0/7	0/0	0/0	0/0	? parrot 0.1.0
0/0	0/0	0/0	0/0	0/0	? anchor-lang 0.5.0
0/0	0/0	0/0	0/0	0/0	? anchor-attribute-access-control 0.5.0
0/0	8/8	0/0	0/0	0/0	☹️ anchor-syn 0.5.0
15/18	432/439	3/3	0/0	11/11	☹️ anyhow 1.0.40
0/0	1/1	0/0	0/0	0/0	☹️ bs58 0.3.1
0/0	0/0	0/0	0/0	0/0	? heck 0.3.2
0/0	0/0	0/0	0/0	0/0	? unicode-segmentation 1.7.1
0/0	0/0	0/0	0/0	0/0	? proc-macro2 1.0.24
0/0	0/0	0/0	0/0	0/0	? unicode-xid 0.2.1
0/0	0/0	0/0	0/0	0/0	? quote 1.0.9
0/0	0/0	0/0	0/0	0/0	? proc-macro2 1.0.24
0/0	4/4	0/0	0/0	0/0	☹️ serde 1.0.125
0/0	0/0	0/0	0/0	0/0	? serde_derive 1.0.125
0/0	0/0	0/0	0/0	0/0	? proc-macro2 1.0.24
0/0	0/0	0/0	0/0	0/0	? quote 1.0.9
0/0	45/45	3/3	0/0	2/2	☹️ syn 1.0.67
0/0	0/0	0/0	0/0	0/0	? proc-macro2 1.0.24
0/0	0/0	0/0	0/0	0/0	? quote 1.0.9
0/0	0/0	0/0	0/0	0/0	? unicode-xid 0.2.1
0/0	4/6	0/0	0/0	0/0	☹️ serde_json 1.0.64
0/0	1/1	0/0	0/0	0/0	☹️ itoa 0.4.7
8/12	674/921	0/0	0/0	2/2	☹️ ryu 1.0.5
0/0	4/4	0/0	0/0	0/0	☹️ serde 1.0.125
2/2	73/74	0/0	0/0	0/0	☹️ sha2 0.9.3
0/0	6/6	0/0	0/0	0/0	☹️ block-buffer 0.9.0
0/0	3/3	0/0	0/0	0/0	☹️ block-padding 0.2.1
1/1	295/295	20/20	8/8	5/5	☹️ generic-array 0.14.4
0/0	4/4	0/0	0/0	0/0	☹️ serde 1.0.125
0/0	0/0	0/0	0/0	0/0	☹️ typenum 1.13.0
0/0	0/0	0/0	0/0	0/0	? cfg-if 1.0.0
0/0	0/0	0/0	0/0	0/0	? cpuid-bool 0.1.2
0/0	0/0	0/0	0/0	0/0	? digest 0.9.0
1/1	295/295	20/20	8/8	5/5	☹️ generic-array 0.14.4
0/0	0/0	0/0	0/0	0/0	? opaque-debug 0.3.0
0/0	45/45	3/3	0/0	2/2	☹️ syn 1.0.67
0/0	0/0	0/0	0/0	0/0	? thiserror 1.0.24
0/0	0/0	0/0	0/0	0/0	? thiserror-impl 1.0.24
0/0	0/0	0/0	0/0	0/0	? proc-macro2 1.0.24
0/0	0/0	0/0	0/0	0/0	? quote 1.0.9
0/0	45/45	3/3	0/0	2/2	☹️ syn 1.0.67
15/18	432/439	3/3	0/0	11/11	☹️ anyhow 1.0.40
0/0	0/0	0/0	0/0	0/0	? proc-macro2 1.0.24
0/0	0/0	0/0	0/0	0/0	? quote 1.0.9
0/0	34/34	1/2	0/0	2/2	☹️ regex 1.4.5
19/19	678/678	0/0	0/0	22/22	☹️ aho-corasick 0.7.15
26/27	1823/1896	0/0	0/0	0/0	☹️ memchr 2.3.4
0/19	10/311	0/0	0/0	5/27	☹️ libc 0.2.91
26/27	1823/1896	0/0	0/0	0/0	☹️ memchr 2.3.4
0/0	0/0	0/0	0/0	0/0	? regex-syntax 0.6.23
0/0	45/45	3/3	0/0	2/2	☹️ syn 1.0.67

After cloning the repository, SecureContract installed and executed `cargo geiger` on the in scope program. The results show that many core components

contain unsafe Rust code.

Risk Level:

Likelihood - 1

Impact - 2

Recommendation:

It is recommended to always double check unsafe Rust code in your own codebase and monitor any core dependencies that contain unsafe Rust in case of any found vulnerabilities.

Remediation Plan:

ACKNOWLEDGED: **X team** claims that the use of certain dependencies is out of their control.

3.4 (VULN-04) LOW TEST COVERAGE INFORMATIONAL

Description:

Checking the code by automated testing (unit testing or functional testing) is a good practice to be sure all lines of the code work correctly. SecureContract used an automatic tool to discover the test coverage. This is also known as “code coverage”. The tool used by the auditors is a rust utility called **cargo tarpaulin**.

Result:

```
running 9 tests
test math::tests::example_inaccurate_float_calculation ... ok
test math::tests::test_calculate_collateral_ratio_precision_lost ... ok
test math::tests::test_calculate_collateral_repay_amount ... ok
test math::tests::test_calculate_collateral_ratio ... ok
test math::tests::test_rpy ... ok
test stables_oracle::test_id ... ok
test math::tests::tool_calculate_interest_rate ... ok
test tests::test_vault_decrease_collateral ... ok
test tests::test_vault_type_update_interest_accum ... ok

test result: ok. 9 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.04s

Aug 19 03:10:24.694 INFO cargo_tarpaulin::report: Coverage Results:
|| Tested/Total Lines:
|| node_modules/@solana/web3.js/examples/bpf-rust-noop/src/lib.rs: 0/1
|| programs/parrot/src/lib.rs: 115/476
|| programs/parrot/src/math.rs: 135/136
||
|| 40.78% coverage, 250/613 lines covered
```

After cloning the repository, SecureContract installed and executed `cargo tarpaulin` on the libraries on the in scope components. The coverage results ended up determining that **40.78%** of the lines of rust code were covered with unit/function tests. Details on which components/libraries have coverage are provided in the output on the next page.

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to have the developers enhance the code coverage to have as much possible tests to check all the functionalities of the ledger platform. This will ensure the production release functions as intended.

Remediation Plan:

ACKNOWLEDGED: **X team** claims that most of the tests are integration tests, and not captured by rust tests. So they consider that the test coverage is acceptable.



MANUAL TESTING

Description:

Custom tests are useful for developers to check if functions and permissions work correctly. Furthermore, they are also useful for security auditors to perform security tests behaving like a malicious user. Then, auditors manually manipulated inputs to check the security in the smart contracts.

4.1 WITHDRAW AMOUNT ON BEHALF OF OTHER USER

Description:

`unstake()` method requires just amount as an argument. Since `unstake()` is public function, one can pass `Unstake` object and amount value to withdraw the amount on behalf of other users. But it is not possible due to derived macro `account` for `anchor_lang` makes sure correct value for vault owner, debt token, collateral token holder etc in passed `Unstake` object.

Code Location:

Listing 6

```
1      #[derive(Accounts)]
2      pub struct Unstake<'info> {
3          debt_type: ProgramAccount<'info, DebtType>,
4
5          #[account(mut, has_one = debt_type)]
6          vault_type: ProgramAccount<'info, VaultType>,
7
8          #[account(mut, "&debt_type.debt_token == debt_token.
              to_account_info().key")]
9          debt_token: CpiAccount<'info, Mint>, //to get token decimal
10
11         #[account(mut, has_one = vault_type)]
12         vault: ProgramAccount<'info, Vault>,
13
14         #[account("&vault_type.price_oracle == oracle.key")]
```

```

15     oracle : AccountInfo <' info >,
16
17     #[ account ( mut, signer, "& vault . owner == vault_owner . key ") ]
18     vault_owner : AccountInfo <' info >,
19
20     #[ account ( " token_program . key == &token :: ID ") ]
21     token_program : AccountInfo <' info >,
22
23     #[ account (
24         mut,
25         "&vault_type . collateral_token == collateral_token .
26             to_account_info (). key "
27     )]
28     collateral_token : CpiAccount <' info , Mint >,
29
30     #[ account (
31         mut,
32         "&vault_type . collateral_token_holder ==
33             collateral_token_holder . key "
34     )]
35     collateral_token_holder : AccountInfo <' info >,
36
37     // PDA of vault_type
38     collateral_token_holder_authority : AccountInfo <' info >,
39
40     #[ account ( mut )]
41     receiver : AccountInfo <' info >,
42
43     clock : Sysvar <' info , Clock >,
44 }

```

Results:

Struct **Unstake** checks vault owner, debt token, collateral tokens, collateral token holder account etc, so it is not possible to perform unstake on behalf of another user.

4.2 BORROW AMOUNT ON BEHALF OF OTHER USER

Description:

`borrow()` method requires just amount as an argument. Since `borrow()` is public function, one can pass `Borrow` object and amount value to borrow the amount on behalf of other users. But it is not possible due to implemented checked for vault owner, vault token, debt originator etc in passed `Borrow` object.

Code Location:

Listing 7

```
1      pub struct Borrow<'info> {
2          debt_type: ProgramAccount<'info, DebtType>,
3
4          #[account(mut, has_one = debt_type)]
5          vault_type: ProgramAccount<'info, VaultType>,
6
7          #[account("&vault_type.collateral_token ==
              collateral_token_mint.to_account_info().key")]
8          collateral_token_mint: CpiAccount<'info, Mint>, //to get token
              decimal
9
10         #[account(mut, has_one = vault_type)]
11         vault: ProgramAccount<'info, Vault>,
12
13         #[account(signer, "&vault.owner == vault_owner.key")]
14         vault_owner: AccountInfo<'info>,
15
16         #[account("token_program.key == &token::ID")]
17         token_program: AccountInfo<'info>,
18
19         #[account("&debt_type.debt_token == debt_token.
              to_account_info().key")]
20         debt_token: CpiAccount<'info, Mint>, //use Mint to get token
              decimal
21
```

```

22     #[ account ( mut, "& debt_type . debt_originator == debt_originator .
        key ") ]
23     debt_originator : AccountInfo <' info >,
24

```

Results:

Struct **Borrow** checks vault owner, vault token, debt originator etc, so it is not possible to perform borrow on behalf of other user.

Also method **increase_debt()** checks the **debt_ceiling** to prevent unlimited borrow.



FUZZING

5.1 FUZZING UNSAFE RUST DEPENDENCIES

Description:

Since the program uses some core dependencies that contain unsafe Rust code, SecureContract used some custom fuzzing tools and industry standard tools like `libfuzzer`, `honggfuzz` and `fzero_fuzzer` to fuzz some of those dependencies for a certain period.

Results:

Due to the time constraints, only two handpicked dependencies were fuzzed for a certain amount of time. All fuzzing tests were positive ie. `no issues were detected at this time`.

- `serde-json`: Fuzz Code

```
#![no_main]

use libfuzzer_sys::fuzz_target;
use serde_json::*;

fn json_fuzz(data:&str) -> serde_json::Result<()> {
    // Some JSON input data as a &str. Maybe this comes from the user.

    // Parse the string of data into serde_json::Value.
    let v: serde_json::Value = serde_json::from_str(data)?;

    Ok(())
}

fuzz_target!(|data: &[u8]| {
    if let Ok(data) = std::str::from_utf8(data) {
        json_fuzz(data);
    }
});
```

- `anyhow`: Fuzz Code

```

#[macro_use]
extern crate honggfuzz;
extern crate anyhow;

use anyhow::{Result, anyhow};

fn err (data: &str) → Result<> {
    anyhow::ensure!(data.len() > 0, "only user {} is allowed", data);

    return Err(anyhow!("some error {:?}", data));
}

► Run | Debug
fn main() → Result<>{
    println!("Starting fuzzer");

    loop {

        fuzz!(|data: &str| {

            err(data);

        });

    }
}

```



AUTOMATED TESTING

6.1 VULNERABILITIES AUTOMATIC DETECTION

Description:

SecureContract 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 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:

```
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 323 security advisories (from /home/ethsec/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (131 crate dependencies)
Crate:      cpuid-bool
Version:    0.1.2
Warning:    unmaintained
Title:      `cpuid-bool` has been renamed to `cpufeatures`
Date:      2021-05-06
ID:         RUSTSEC-2021-0064
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0064
Dependency tree:
cpuid-bool 0.1.2
├── sha2 0.9.3
│   ├── solana-program 1.6.10
│   │   ├── spl-token 3.1.0
│   │   │   ├── serum_dex 0.3.0
│   │   │   │   ├── anchor-spl 0.5.0
│   │   │   │   │   └── parrot 0.1.0
│   │   │   ├── flux-aggregator 0.1.0
│   │   │   │   └── parrot 0.1.0
│   │   │   └── anchor-spl 0.5.0
│   │   ├── serum_dex 0.3.0
│   │   ├── parrot 0.1.0
│   │   ├── flux-aggregator 0.1.0
│   │   ├── anchor-spl 0.5.0
│   │   ├── anchor-lang 0.5.0
│   │   │   └── parrot 0.1.0
│   │   │   └── anchor-spl 0.5.0
│   │   └── solana-frozen-abi 1.6.10
│   │       └── solana-program 1.6.10
│   └── anchor-syn 0.5.0
│       ├── anchor-derive-accounts 0.5.0
│       │   └── anchor-lang 0.5.0
│       ├── anchor-attribute-state 0.5.0
│       │   └── anchor-lang 0.5.0
│       ├── anchor-attribute-program 0.5.0
│       │   └── anchor-lang 0.5.0
│       ├── anchor-attribute-interface 0.5.0
│       │   └── anchor-lang 0.5.0
│       ├── anchor-attribute-event 0.5.0
│       │   └── anchor-lang 0.5.0
│       ├── anchor-attribute-error 0.5.0
│       │   └── anchor-lang 0.5.0
│       ├── anchor-attribute-account 0.5.0
│       │   └── anchor-lang 0.5.0
│       └── anchor-attribute-access-control 0.5.0
│           └── anchor-lang 0.5.0
warning: 1 allowed warning found
```


6.2 UNSAFE RUST CODE DETECTION

Description:

SecureContract used automated security scanners to assist with the detection of well-known security issues and vulnerabilities. Among the tools used was `cargo-geiger`, a security tool that lists statistics related to the usage of unsafe Rust code in a core Rust codebase and all its dependencies.

Results:

Metric output format: x/y
 x = unsafe code used by the build
 y = total unsafe code found in the crate

Symbols:
 🛡️ = No 'unsafe' usage found, declares #![forbid(unsafe_code)]
 ? = No 'unsafe' usage found, missing #![forbid(unsafe_code)]
 ⚠️ = 'unsafe' usage found

Functions	Expressions	Impls	Traits	Methods	Dependency
0/1	0/7	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	8/8	0/0	0/0	0/0	⚠️
15/18	432/439	3/3	0/0	11/11	⚠️
0/0	1/1	0/0	0/0	0/0	⚠️
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	4/4	0/0	0/0	0/0	⚠️
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	45/45	3/3	0/0	2/2	⚠️
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	🛡️
0/0	4/6	0/0	0/0	0/0	⚠️
0/0	1/1	0/0	0/0	0/0	⚠️
8/12	674/921	0/0	0/0	2/2	⚠️
0/0	4/4	0/0	0/0	0/0	⚠️
2/2	73/74	0/0	0/0	0/0	⚠️
0/0	6/6	0/0	0/0	0/0	⚠️
0/0	3/3	0/0	0/0	0/0	⚠️
1/1	295/295	20/20	8/8	5/5	⚠️
0/0	4/4	0/0	0/0	0/0	⚠️
0/0	0/0	0/0	0/0	0/0	🛡️
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	🛡️
1/1	295/295	20/20	8/8	5/5	⚠️
0/0	0/0	0/0	0/0	0/0	?
0/0	45/45	3/3	0/0	2/2	⚠️
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	45/45	3/3	0/0	2/2	⚠️
15/18	432/439	3/3	0/0	11/11	⚠️
0/0	0/0	0/0	0/0	0/0	?
0/0	0/0	0/0	0/0	0/0	?
0/0	34/34	1/2	0/0	2/2	⚠️
19/19	678/678	0/0	0/0	22/22	⚠️
26/27	1823/1896	0/0	0/0	0/0	⚠️
0/19	10/311	0/0	0/0	5/27	⚠️
26/27	1823/1896	0/0	0/0	0/0	⚠️
0/0	0/0	0/0	0/0	0/0	🛡️

0.1.0
 anchor-lang 0.5.0
 anchor-attribute-access-control 0.5.0
 anchor-syn 0.5.0
 anyhow 1.0.40
 bs58 0.3.1
 heck 0.3.2
 unicode-segmentation 1.7.1
 proc-macro2 1.0.24
 unicode-xid 0.2.1
 quote 1.0.9
 proc-macro2 1.0.24
 serde 1.0.125
 serde_derive 1.0.125
 proc-macro2 1.0.24
 quote 1.0.9
 syn 1.0.67
 proc-macro2 1.0.24
 quote 1.0.9
 unicode-xid 0.2.1
 serde_json 1.0.64
 itoa 0.4.7
 ryu 1.0.5
 serde 1.0.125
 sha2 0.9.3
 block-buffer 0.9.0
 block-padding 0.2.1
 generic-array 0.14.4
 serde 1.0.125
 typenum 1.13.0
 cfg-if 1.0.0
 cpuid-bool 0.1.2
 digest 0.9.0
 generic-array 0.14.4
 opaque-debug 0.3.0
 syn 1.0.67
 thiserror 1.0.24
 thiserror-impl 1.0.24
 proc-macro2 1.0.24
 quote 1.0.9
 syn 1.0.67
 anyhow 1.0.40
 proc-macro2 1.0.24
 quote 1.0.9
 regex 1.4.5
 aho-corasick 0.7.15
 memchr 2.3.4
 libc 0.2.91
 memchr 2.3.4
 regex-syntax 0.6.23

6.3 RUST UNDEFINED BEHAVIOR TESTING

Description:

SecureContract used automated security scanners to assist with detection of different classes of undefined behaviour in Rust. Among the tools used was `Miri`, an experimental interpreter for Rust's mid-level intermediate representation (MIR). It can run binaries and test suites of cargo projects and detect certain classes of undefined behavior, for example:

- Out-of-bounds memory accesses and use-after-free
- Invalid use of uninitialized data
- Violation of intrinsic preconditions (an `unreachable_unchecked` being reached, calling `copy_nonoverlapping` with overlapping ranges, ...)
- Not sufficiently aligned memory accesses and references
- Violation of some basic type invariants (a bool that is not 0 or 1, for example, or an invalid enum discriminant)
- Experimental: Violations of the Stacked Borrows rules governing aliasing for reference types
- Experimental: Data races (but no weak memory effects)

On top of that, `Miri` will also tell you about memory leaks: when there is memory still allocated at the end of the execution, and that memory is not reachable from a global `static`, `Miri` will raise an error.

SecureContract ran predefined tests that exist in the program through `Miri` to catch any undefined behavior.

Results:

All tests are passing without any issues raised by [Miri](#)

```
running 9 tests
test math::tests::example_inaccurate_float_calculation ... ok
test math::tests::test_calculate_collateral_ratio ... ok
test math::tests::test_calculate_collateral_ratio_precision_lost ... ok
test math::tests::test_calculate_collateral_repay_amount ... ok
test math::tests::test_rpy ... ok
test math::tests::tool_calculate_interest_rate ... ok
test stables_oracle::test_id ... ok
test tests::test_vault_decrease_collateral ... ok
test tests::test_vault_type_update_interest_accum ... ok

test result: ok. 9 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

References:

[Rust Undefined Behavior](#)

THANK YOU FOR CHOOSING

SecureContract