marginfi

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Akash Gurugunti Sud0u53r.ak@osec.io

Contents

Appendices

01 | Executive Summary

Overview

mrgn labs engaged OtterSec to perform an assessment of the `marginfi` program.

This assessment was conducted between August 16th and September 5th, 2022.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team over to streamline patches and confirm remediation.

We delivered final confirmation of the patches **[not yet delivered]**.

Key Findings

The following is a summary of the major findings in this audit.

- 6 findings total
- 1 vulnerability which could lead to loss of funds
  - OS-MGF-ADV-00: The Mango on-chain observer does not verify whether accounts are associated with a given marginfi account.

02 | Scope

The source code was delivered to us in a git repository at github.com/mrgnlabs/marginfi. This audit was performed against commit `e155585`.

There was 1 program included in this audit. A brief description for each program is given below. A full list of program files and hashes can be found in Appendix A.

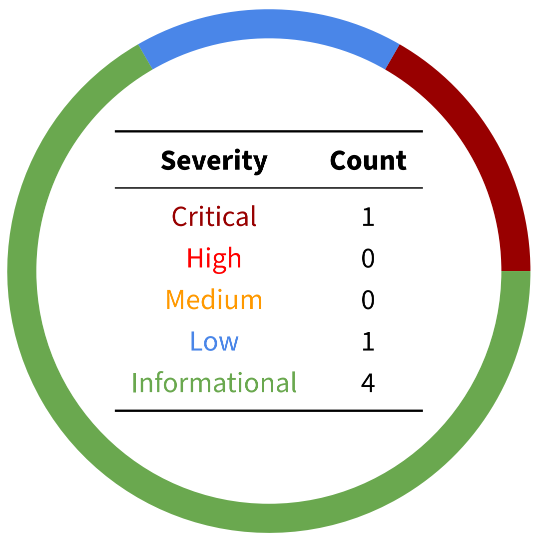
Name	Description
marginfi	Decentralized portfolio margining protocol.

03 | Findings

Overall, we report 6 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.

The below chart displays the findings by severity.



Severity	Count
Critical	1
High	0
Medium	0
Low	1
Informational	4

04 | Vulnerabilities

Here we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix D.

ID	Severity	Status	Description
OS-MGF-ADV-00	Critical	Resolved	The Mango on-chain observer does not verify whether accounts are associated with a given marginfi account.
OS-MGF-ADV-01	Low	TODO	UTP accounts can be deactivated if their balance is below the 1 USD dust threshold, which subsequently disappears from the protocol's point of view.

OS-MGF-ADV-00 [crit] [resolved] | Missing mango account check

Description

In mango_state.rs, a MangoObserver struct is instantitated from a list of Mango accounts. However, there is no constraint validating that the provided addresses are actually associated with the marginfi account. An attacker can abuse this by passing in arbitrary Mango accounts; this would allow them to take under collateralized loans or unfairly liquidate other users.

Proof of Concept

Consider the following scenario:

1. An attacker invokes the InitMarginfiAccount instruction to create a marginfi account.
2. They invoke the UtpMangoActivate instruction to activate their Mango UTP account.
3. They invoke the UtpMangoDeposit instruction with a different set of Mango accounts, in particular with more equity than expected, for their marginfi account. This allows them to bypass marginfi_account.check_margin_requirement and gain an under collateralized loan.
4. They invoke the Liquidate instruction with a different set of Mango accounts, in particular with less equity than expected, for the liquidatee's marginfi account. This allows them to bypass meets_margin_requirement and liquidate a healthy loan.

Remediation

Add a constraint to validate the mango account with the address in utp_config.address.

src/state/mango_state.rs DIFF

```diff
    let [mango_account_ai, mango_group_ai, mango_cache_ai] = ais;

+   check!(
+       utp_config.address.eq(mango_account_ai.key),
+       MarginfiError::InvalidObserveAccounts
+   );
```

Patch

Fixed in #200.

OS-MGF-ADV-01 [low] | Premature UTP deactivation

This finding was raised by mrgn labs in the course of the assessment.

The `DeactivateUTP` instruction is used to remove empty UTP accounts from a marginfi account's state. In particular, a Mango/01 account is considered empty if it has less than 1 USD worth of assets. The issue is that after deactivation, the protocol loses access to any remaining assets in the UTP account.

src/state/mango_state.rs RUST

```rust
pub fn is_empty<'a>(
    health_cache: &'a mut HealthCache,
    mango_group: &'a MangoGroup,
) -> MarginfiResult<bool> {
    let (assets, _) = health_cache.get_health_components(mango_group,
    ↪  HealthType::Equity);
    Ok(assets < DUST_THRESHOLD_F)
}
```

Proof of Concept

Consider the following scenario:

1. An attacker initializes a marginfi account and activates UTP accounts for Mango and 01.
2. They deposit less than 1 USD into Mango so that it can be deactivated at any time.
3. They additionally borrow and deposit a smaller amount into 01.
4. They deactivate the Mango UTP so that their account does not meet margin requirements.
5. They self-liquidate, and the missing funds are automatically covered by marginfi's insurance fund.

05 | General Findings

Here we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they do represent antipatterns and could introduce a vulnerability in the future.

ID	Status	Description
OS-MGF-SUG-00	TODO	UTP deactivation has duplicate code which can be refactored.
OS-MGF-SUG-01	TODO	Vault token accounts should be initialized with a PDA.
OS-MGF-SUG-02	TODO	Adding constraints to accounts from external protocols would improve robustness.
OS-MGF-SUG-03	TODO	It is possible to reach an illegal marginfi group state through configuration.

OS-MGF-SUG-00 | Duplicate code

Description

In the `DeactivateUTP` instruction, the code which clears a marginfi account's UTP duplicates logic which has already been abstracted into the `MarginfiAccount::deactivate_utp` method.

Remediation

Replace the code with a function call.

src/instructions/utp_deactivate.rs DIFF

```diff
pub fn process(ctx: Context<DeactivateUTP>, utp_index: usize) -> MarginfiResult {
    let mut marginfi_account = ctx.accounts.marginfi_account.load_mut()?;
    let utp_active = marginfi_account.active_utps[utp_index];

    check!(utp_active, MarginfiError::IllegalUtpDeactivation);

    let utp_observer = UtpObserver::new(ctx.remaining_accounts);

    check!(
        utp_observer
            .observation(&marginfi_account, utp_index)?
            .is_empty()?,
        MarginfiError::IllegalUtpDeactivation
    );

-   marginfi_account.active_utps[utp_index] = false;
-   marginfi_account.utp_account_config[utp_index] = UTPAccountConfig::default();
+   marginfi_account.deactivate_utp(utp_index);

    Ok(())
}
```


OS-MGF-SUG-01 | Use PDAs for vault accounts

Description

In the `InitMarginfiGroup` instruction, vault token accounts should be initialized with a PDA instead of allowing the admin to pass in arbitrary accounts. This applies to the bank, fee, and insurance vaults.

Similarly, the `UtpMangoDeposit` and `UtpZoDeposit` instructions should use PDAs for the temporary collateral account. This ensures the account cannot be initialized by anyone but the program.

Remediation

Initialize accounts with PDA seeds; this can be accomplished with Anchor constraints. An example is given for the `InitMarginfiGroup` instruction's `bank_vault` token account.

src/instructions/init_marginfi_group.rs DIFF

```diff
-   #[account(
-       constraint = bank_vault.mint.eq(&collateral_mint.key()),
-       constraint = bank_vault.owner.eq(&bank_authority.key()),
-       constraint = bank_vault.delegate == COption::None,
-       constraint = bank_vault.close_authority == COption::None
-   )]
+   #[account(
+       init,
+       payer = admin,
+       token::mint = collateral_mint,
+       token::authority = bank_authority,
+       seeds = [
+           "VAULT",
+           PDA_BANK_VAULT_SEED,
+           &marginfi_group.to_account_info().key.to_bytes()
+       ],
+       bump
+   )]
    pub bank_vault: Account<'info, TokenAccount>,
```

Another example is given for the `UtpMangoDeposit` instruction's `temp_collateral_account` token account. In this code snippet, Anchor will initialize and close the account during each invocation.

src/instructions/mango/deposit.rs DIFF

```diff
-   #[account(
-       mut,
-       constraint = temp_collateral_account.amount == 0
-   )]
+   #[account(
+       init,
+       payer = signer,
+       close = signer,
+       token::mint = margin_collateral_vault.load()?.mint,
+       token::authority = mango_authority,
+       seeds = [
+           "TEMP_ACC",
+           &marginfi_account.to_account_info().key.to_bytes()
+       ],
+       bump
+   )]
    pub temp_collateral_account: Account<'info, TokenAccount>,
```


OS-MGF-SUG-02 | UTP account constraints

Description

In the `UtpMangoActivate` and `UtpZoActivate` instructions, there are no constraints that checks that the passed in accounts are under ownership of respective protocols and data is empty. It is better to check them here to avoid re-initialization of accounts instead of relying on the underlying protocols.

Remediation

A simple form of validation is to check program ownership. Here, `mango_account` and `zo_margin` are PDAs which will be created within the CPI call. On the other hand, `zo_control` is expected to be zero-initialized in advance.

src/instructions/mango/activate.rs DIFF

```diff
-   #[account(mut)]
+   #[account(
+       mut,
+       owner = system_program.key(),
+   )]
    pub mango_account: AccountInfo<'info>,
```

src/instructions/zo/activate.rs DIFF

```diff
-   #[account(mut)]
+   #[account(
+       mut,
+       owner = system_program.key()
+   )]
    /// CHECK: Defer verification to UTP
    pub zo_margin: AccountInfo<'info>,
-   #[account(mut)]
+   #[account(
+       mut,
+       owner = zo_program.key()
+   )]
    /// CHECK: Defer verification to UTP
    pub zo_control: AccountInfo<'info>,
```

OS-MGF-SUG-03 | Configuration bypass

The `ConfigMarginfiGroup` instruction is used to configure the parameters of a marginfi group. Notice that the bank's `maint_margin_ratio` value is validated to be at most `init_margin_ratio`. However, this invariant may be bypassed by later reducing `init_margin_ratio`, which has no corresponding check.

src/state/marginfi_group.rs RUST

```rust
if let Some(val) = config.init_margin_ratio {
    let val = downscale_uint_to_fixed(val);
    msg!("Setting {} to {}", stringify!(self.init_margin_ratio), val);
    check!(I80F48::ZERO <= val, MarginfiError::IllegalConfig);
    self.init_margin_ratio = val.into();
}

if let Some(val) = config.maint_margin_ratio {
    let val = downscale_uint_to_fixed(val);
    msg!("Setting {} to {}", stringify!(self.maint_margin_ratio),
    ↪  val);
    check!(
        I80F48::ZERO <= val && val <=
    ↪   I80F48::from(self.init_margin_ratio),
        MarginfiError::IllegalConfig
    );
    self.maint_margin_ratio = val.into();
}
```

Remediation

Updating the `init_margin_ratio` parameter should maintain the desired invariant. To do so, validate that the proposed value is at least `maint_margin_ratio`.

A | Program Files

Below are the files in scope for this audit and their corresponding SHA256 hashes.

```
Cargo.toml                                  1cc763c9fcbb4bb67882631456ec093ba47feb7c2095ec8557c4a9278deed6cb
Xargo.toml                                  815f2dfb6197712a703a8e1f75b03c6991721e9eb7c40dfaec8b0b49da4aa629
src
  constants.rs                              44702ef15c4ed8a83469e8e328e6783985dace124b07227e3063bba22be88069
  errors.rs                                 d64d3d7471700d1bed312b6868db253dd0cf91a1c344ce557d7d1e6a0c648c3b
  events.rs                                 c93c37bb856d8cdcf88875261e3e03c77608a169d71efdbff72935e0e500420b
  lib.rs                                    055a3e864f09bd420ef26626f706a6279e6f95a81dedce43f1a626ed17dc0b01
  macros.rs                                 81247190750efa26bd77114b9ee2a538c2cebd1c1261e3eaed15f0f5d9b6e463
  prelude.rs                                60f8f2b210f54b22d509a9686a0fd16e713d033cc1fa3f0432932f95f0dcf648
  instructions
    bank_fee_vault_withdraw.rs              ab9561255955c952ecda2c5940d522480035528c4396c670e3b49dd3add9650e
    bank_insurance_vault_withdraw.rs        21710526ddcfcb56e36b5be0bef4d3eb1d169b7d9d642c7d0bdd3695b62cd839
    configure_marginfi_group.rs             4075c70c4b77b7c2b2a3235df23c7aba8baaf28bec7bf34893f92a96dcdf3547
    handle_bankruptcy.rs                    5568d88ac8d9d9bafc1e8f808cdfbbafdd88b6f6c10de7c8d05e0531518e22fd
    init_marginfi_account.rs                b21d269f67b8b59df5ca20aa15ce0d1df80149b5349df26f5205b04595d4cf1a
    init_marginfi_group.rs                  122f2cf1303b704857df82b2e45e35e8536d3195051cd76c126292dfe55f5784
    liquidate.rs                            21bc97a0b6508375c5e0374f07ec78cf51914e0cc0a2d87904c1077b9d26787a
    margin_deposit_collateral.rs            8e049d98d8ad76986493cfe0da91d4afe697eb0586655d0d56d763cf9ff71343
    margin_withdraw_collateral.rs           e972cc0905f8078941d47f97d975950e890e9ebf8de32b4b5cd0268378e89146
    marginfi_account_configure.rs           3e11cef105c32e27833d3eb9bb981d4a0a5b7f074d3407003ee9fa3990fa18bd
    mod.rs                                  cad0cdf286996242f58adbf02d9b561b611368717e82f75136cb2981c5c80b6b
    update_interest_accumulator.rs          4a629f1ed238fc94d18ab3fd96d1528ac09dcf425e15d31fc38487d10a433784
    utp_deactivate.rs                       02cbd51e5fa5c3fce539287194c24c4dfdfbcba1cd4a7b678f52d60a3e58c8ce
    mango
      activate.rs                           0097c76f61cd5dceb0836e0af7b9a4ec6ad436b5289bb406ae7199f1516b5480
      deposit.rs                            7e98747dd56501d28592cc6496dc3aa951cb285ee96a4b1a691b34bc917446e4
      mod.rs                                953e61299adc4a1cb4166a1edd5deada685c6f19eae7989b982c0fe4e76fc432
      trade.rs                              3f5e33207a79da228af357ffeb34401276a3d05e167e86b44a56680efc2a8d60
      withdraw.rs                           f3d6c58497e66c455b1fa3d3f78a74a5c59393545f10ad0b59f4893f05d94ff7
    zo
      activate.rs                           e9c7f3308bd69b532e66dc0cb92264ba8770f64c964caa6aa7ed9194d70241f2
      deposit.rs                            8b95b2677b83cfdebb4ea26bade5fcee830baa9211c948e4d389813223dee6b6
      mod.rs                                953e61299adc4a1cb4166a1edd5deada685c6f19eae7989b982c0fe4e76fc432
      trade.rs                              6b1d971f2265219c16994780a9affe264ec849f03dceab77e65eacfbdfa1dc80
      withdraw.rs                           55ccc4d9fc37a36e4babb4ec0eb5e9509d018e3d004dd4efdd97845790686270
  state
    decimal.rs                              483d0b429c14d25c9abbce692b9cfd96d87f43878e72b1103021f7550ed7afd4
    mango_state.rs                          12f025986433378c0b7e7fd67593bed653e22dc4ef334110718b1d5eb34306d9
    marginfi_account.rs                     df7d4f945aede435b6e3ff2c956963299fc0fee9cce6a30691710d85fdd7b01d
    marginfi_group.rs                       1adb9a42b57dda74a5813279111255ba3c971c20fb02b1f7fb50fa515db6874d
    mod.rs                                  1ea615a660acefef933f416226dffa72f96366964908e1e32b91db7460c4c3fa
    risk_engine.rs                          75d5520c6fb957a37e94acf6c6ee0afdf1f3d12afff319b65fd381beb314571e
    utp_observation.rs                      9699577f3e279344213ccde6b0a728f1f0b6eec57123e3072fbd01c313693fd2
    zo_state.rs                             eb7c921818669253a1eed35b99a656e86b4ace851d0f0345823b3ddcca3ead4d
  utils
    access_controls.rs                      7585cd996701ab04b2ad46c3939b7922e38eb4021bb7dabb2f1c74fc11e1b27a
    mod.rs                                  a27beac8f9288e56e4249fd307f76d75c724b8ede89015f53bd9261ae321db7a
    utp_dummy_ixs.rs                        34b53f48e176b4189f2751bfa0594fecc6b8d7ece2353a1a5650852539762161
    utp_helpers.rs                          08c3dd87601cfd44acf69ffa0a394ba78b11e35fcbd4fa54d153a2f4616e66d9
tests
  lending.rs                                c929fbadeac042ea33fd9ddbd280fb92559d1b18659ecfdbea28fadf9c8c4492
  marginfi_account.rs                       183dd0817b124a06aab4f42c23d5fa58550d6eb22987483e0d8a8052cdf657bc
  marginfi_group.rs                         8784e9c4302630ed48c8e0683444a85c7f94311be302cab478a99f1cce47715a
  utp.rs                                    91738e94d10143d49d19a5c9dad1b93421582eb04cf21f820aee2082a7fec7c9
  utp_trading.rs                            0dd8a55a49543f22f9a60c903740d6cede9a5922ac9b600e28a16ed5c0b70452
  fixtures
    marginfi_account.rs                     025860d112f907baa17570056730ef4a624aa0fad29b0c57274b55dd8e79266a
    marginfi_group.rs                       eab023f90ee7bb5759c71401cd8481d1a8e98a8584b7a394a2a7cf0acbf9ce12
    mod.rs                                  c03e3d443792f61531b0649a5cfe2a7c1ed2d55daf593d50e42e9df2b3c5ba2d
    prelude.rs                              95845c3176f585f7957b54a2103cb0d0ab9befc8da4d4333860cd25c4608c63e
    spl.rs                                  59ce2fcddb0a27737582ed113c210e33642e3ebb7096639d5f4a7dd679da1537
    test.rs                                 582b8cf4bb8e14c1748fc88e12493c69b6b81c612e3a60389db4ccb08d80b042
    utils.rs                                5a287a40922f1a111dea4e98dceac45ef95d20f148b73fd899db62886dce8372
```

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an onchain program. In other words, there is no way to steal tokens or deny service, ignoring any Solana specific quirks such as account ownership issues. An example of a design vulnerability would be an onchain oracle which could be manipulated by flash loans or large deposits.

On the other hand, auditing the implementation of the program requires a deep understanding of Solana's execution model. Some common implementation vulnerabilities include account ownership issues, arithmetic overflows, and rounding bugs. For a non-exhaustive list of security issues we check for, see Appendix C.

Implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach any target in a team of two. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

C | Implementation Security Checklist

Unsafe arithmetic

Integer underflows or overflows	Unconstrained input sizes could lead to integer over or underflows, causing potentially unexpected behavior. Ensure that for unchecked arithmetic, all integers are properly bounded.
Rounding	Rounding should always be done against the user to avoid potentially exploitable off-by-one vulnerabilities.
Conversions	Rust as conversions can cause truncation if the source value does not fit into the destination type. While this is not undefined behavior, such truncation could still lead to unexpected behavior by the program.

Account security

Account Ownership	Account ownership should be properly checked to avoid type confusion attacks. For Anchor, the safety of unchecked accounts should be clearly justified and immediately obvious.
Accounts	For non-Anchor programs, the type of the account should be explicitly validated to avoid type confusion attacks.
Signer Checks	Privileged operations should ensure that the operation is signed by the correct accounts.
PDA Seeds	PDA seeds are uniquely chosen to differentiate between different object classes, avoiding collision.

Input validation

Timestamps	Timestamp inputs should be properly validated against the current clock time. Timestamps which are meant to be in the future should be explicitly validated so.
Numbers	Sane limits should be put on numerical input data to mitigate the risk of unexpected over and underflows. Input data should be constrained to the smallest size type possible, and upcasted for unchecked arithmetic.
Strings	Strings should have sane size restrictions to prevent denial of service conditions
Internal State	If there is internal state, ensure that there is explicit validation on the input account's state before engaging in any state transitions. For example, only open accounts should be eligible for closing.

Miscellaneous

Libraries	Out of date libraries should not include any publicly disclosed vulnerabilities
Clippy	cargo clippy is an effective linter to detect potential anti-patterns.

D | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical

Vulnerabilities which immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority/token account validation
- Rounding errors on token transfers

High

Vulnerabilities which could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities which could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input cause computation limit exhaustion
- Forced exceptions preventing normal use

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation
- Uncaught Rust errors (vector out of bounds indexing)