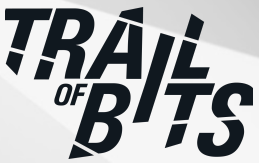

Drift Protocol

Security Assessment

February 15, 2023

Prepared for:
Drift Protocol

Prepared by: **Anders Helsing and Samuel Moelius**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

Drift Protocol engaged Trail of Bits to review the security of its decentralized exchange and smart contract. From November 7 to December 2, 2022, a team of two consultants conducted a security review of the client-provided source code, with six person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

From January 23 to January 25, 2023, Trail of Bits reviewed the fixes and mitigations implemented by Drift Protocol to resolve the issues described in this report. A detailed review of the current status of each issue is provided in Appendix E.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We had access to the source code and documentation. We conducted this audit with full knowledge of the target system, including access to the source code and documentation. We performed static and dynamic testing of the target system and its codebase, using both automated and manual processes.

Summary of Findings

The audit did not uncover any high-severity flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and details on notable findings are provided below.

EXPOSURE ANALYSIS

Severity	*Count*
Medium	1
Informational	15
Undetermined	4

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	3
Auditing and Logging	1
Configuration	1
Data Validation	3
Error Reporting	1
Patching	2
Testing	3
Undefined Behavior	6

Notable Findings

Notable flaws that could impact system confidentiality, integrity, or availability are listed below.

- **TOB-DRIFT-2**
  The Anchor tests are not run as part of Drift Protocol's CI process.
- **TOB-DRIFT-4**
  A race condition in the Drift SDK causes client programs to operate on non-existent or possibly stale data. The race condition affects many of the project's Anchor tests, making them unreliable.

- **TOB-DRIFT-12**
  The exchange's status is represented using an `enum`, which does not allow more than one individual operation to be paused. The exchange's status is checked in multiple, inconsistent ways (e.g., in function annotations and in function bodies).
- **TOB-DRIFT-13**
  Spot market access controls are only partially implemented.
- **TOB-DRIFT-16**
  The codebase uses integer types inconsistently; data of similar kinds is represented using differently sized types, or types with different signedness. There are nearly 700 casts from one integer type to another, each of which could cause its enclosing operation to fail.

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineers were associated with this project:

Anders Helsing, Consultant
anders.helsing@trailofbits.com

Samuel Moelius, Consultant
samuel.moelius@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
October 27, 2022	Pre-project kickoff call
November 16, 2022	Status update meeting #1
November 28, 2022	Status update meeting #2
December 5, 2022	Delivery of report draft
December 5, 2022	Report readout meeting
January 6, 2023	Delivery of revised draft report
February 15, 2023	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of the Drift Protocol decentralized exchange and smart contract. We conducted the assessment through a combination of manual and automated review, including applying a comprehensive suite of tools to automatically uncover bugs, review of the architecture of the system for design flaws, a detailed manual code review, and identification and evaluation of security and correctness properties. We used the following non-exhaustive list of questions to drive our investigation throughout the assessment:

- Can a malicious actor withdraw funds from the Drift Protocol contract in a manner other than intended?
- Can funds become frozen?
- Can math operations within the contract instructions result in overflow or underflow conditions?
- Is it possible to bypass the checks on accounts used by instructions?
- Can instructions use the wrong type of accounts?
- Does Drift Protocol use oracles safely?

Project Targets

The engagement involved a review and testing of the following target.

Drift Protocol v2

Repository	https://github.com/drift-labs/protocol-v2
Version	`57dd5c647253d6e4d9c12b75e17ce603eeb60315`
Type	Rust
Platform	Solana

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Static analysis.** We ran Clippy over the codebase with `-W pedantic` and reviewed the warnings that were produced.
- **Test review.** We ran the Cargo tests and verified that they passed. We attempted the same with the Anchor tests.
- **Manual review.** We manually reviewed the Drift Protocol contract with a focus on answering the questions listed under Project Goals.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We reviewed the authentication of all the user, IF staker, and admin instructions. However, the admin instructions are only cursorily examined to ensure that the provided admin account has signed the transaction and that it is tied to the state account.
- Much of the code is written in an inconsistent style. One would expect that if two functions fulfill similar roles, then they would exhibit similar structure. However, it is difficult to discern such patterns from the current codebase (see TOB-DRIFT-12 for an example). This fact considerably hindered manual review.
- We were unable to reliably run the Anchor tests, we suspect because of TOB-DRIFT-4. Thus, our ability to test the contract dynamically was limited to using the Cargo tests.

Codebase Maturity Evaluation

A codebase maturity evaluation is a holistic assessment that seeks to identify systemic issues and/or opportunities for improvement in a client's codebase and their overall approach to software development. While the Detailed Findings section provides information about specific issues along with tactical steps to remediate them, the codebase maturity evaluation offers strategic recommendations for widespread problems that will provide long-lasting value.

Trail of Bits indicates the maturity of the codebase across various categories using a traffic-light protocol (akin to standards such as TLP Version 2.0) that has been specifically tailored to provide a clear understanding of the areas in which the codebase is mature, underdeveloped, or somewhere in between. Deficiencies identified in the evaluation often stem from root causes within the software development life cycle (SDLC) that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Mature codebases align with industry best practices in software development and security; as a result, they tend to be better positioned to avoid security issues (e.g., they are less likely to introduce bugs) and to mitigate security issues that do arise (e.g., through robust controls and procedures that mitigate impact to the system and its users).

For more information about the Codebase Maturity Evaluation categories and rating criteria, refer to Appendix B.

Category	Summary	Result
Arithmetic	The code uses unchecked arithmetic and converts between integer types more often than seems necessary (TOB-DRIFT-11, TOB-DRIFT-16).	**Weak**
Auditing	The project emits events for most (if not all) critical operations.	**Satisfactory**
Authentication / Access Controls	Access controls are implemented inconsistently (TOB-DRIFT-12) and, in some places, are only partially implemented (TOB-DRIFT-13).	**Weak/ Missing**

Complexity Management	Code is duplicated both literally and logically, suggesting that it does not employ the right abstractions (TOB-DRIFT-15). Testing code is mixed with production code (TOB-DRIFT-10). Some instructions require accounts that are unused (TOB-DRIFT-18). The code does not employ common best practices, such as having build instructions (TOB-DRIFT-1), reliable tests (TOB-DRIFT-4), a reliable CI process (TOB-DRIFT-2), or a method for uncovering vulnerable dependencies (TOB-DRIFT-3).	**Weak**
Cryptography and Key Management	We found no issues related to cryptography or key management. Where signatures need to be performed, the platform appears to check them correctly.	**Satisfactory**
Decentralization	The platform is administered by a central authority. Drift Protocol has indicated that they plan to develop a DAO. Further investigation is required to determine whether any aspect of the administrative APIs would be unsuitable for control by a DAO.	**Further Investigation Required**
Documentation	The project has comprehensive user documentation. A "Drift v2 Instructions" document was shared with us. We recommend incorporating it into the project's documentation.	**Satisfactory**
Front-Running Resistance	Some amount of front-running risk is inherent to a platform of this kind. We found no issues that would elevate that risk.	**Satisfactory**
Low-Level Manipulation	The code uses several questionable low-level constructs that, even if they are not currently vulnerable, could lead to future problems. Examples include use of Anchor's experimental zero-copy feature (TOB-DRIFT-6), using offsets into serialized data (TOB-DRIFT-7), and expecting the in-memory size of data structures to be the same as their serialized size (TOB-DRIFT-20).	**Moderate**

Testing and Verification	Tests are not run as part of the project's CI process (TOB-DRIFT-2). The integration tests are unreliable as they depend on a library with race conditions (TOB-DRIFT-4). Many tests use opaque constants, making it difficult to tell whether the tests are checking for correct behavior (TOB-DRIFT-17). The code would benefit from more advanced testing methods, such as fuzzing and property-based testing. Drift Protocol does, however, have a simulation framework that was not considered as part of this assessment.	**Weak**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	No build instructions	Testing	**Informational**
2	Inadequate testing	Testing	**Informational**
3	Invalid audit.toml prevents cargo audit from being run	Auditing and Logging	**Informational**
4	Race condition in Drift SDK	Undefined Behavior	**Undetermined**
5	Loose size coupling between function invocation and requirement	Undefined Behavior	**Informational**
6	The zero-copy feature in Anchor is experimental	Undefined Behavior	**Informational**
7	Hardcoded indices into account data	Undefined Behavior	**Informational**
8	Missing verification of maker and maker_stats accounts	Data Validation	**Undetermined**
9	Panics used for error handling	Error Reporting	**Informational**
10	Testing code used in production	Patching	**Undetermined**
11	Inconsistent use of checked arithmetic	Data Validation	**Undetermined**
12	Inconsistent and incomplete exchange status checks	Access Controls	**Medium**

13	Spot market access controls are incomplete	Access Controls	**Informational**
14	Oracles can be invalid in at most one way	Data Validation	**Informational**
15	Code duplication	Patching	**Informational**
16	Inconsistent use of integer types	Undefined Behavior	**Informational**
17	Use of opaque constants in tests	Testing	**Informational**
18	Accounts from contexts are not always used by the instruction	Access Controls	**Informational**
19	Unaligned references are allowed	Undefined Behavior	**Informational**
20	Size of created accounts derived from in-memory representation	Configuration	**Informational**

Detailed Findings

1. Lack of build instructions	
Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-DRIFT-1
Target: `README.md`	

Description

The Drift Protocol repository does not contain instructions to build, compile, test, or run the project. The project's README should include at least the following information:

- Instructions for building the project
- Instructions for running the built artifacts
- Instructions for running the project's tests

The closest thing we have found to build instructions appears in a script in the `drift-sim` repository (figure 1.1). As shown in the figure below, building the project is non-trivial. Users should not be required to rediscover these steps on their own.

```
git submodule update --init --recursive
# build v2
cd driftpy/protocol-v2
yarn && anchor build
# build dependencies for v2
cd deps/serum-dex/dex && anchor build && cd ../../..
# go back to top-level
cd ../../
```

Figure 1.1: `drift-sim/setup.sh`

Additionally, the project relies on `serum-dex`, which currently has an open issue regarding outdated build instructions. Thus, if a user visits the `serum-dex` repository to learn how to build the dependency, they will be misled.

Exploit Scenario

Alice attempts to build and deploy her own copy of the Drift Protocol smart contract. Without instructions, Alice deploys it incorrectly. Users of Alice's copy of the smart contract suffer financial loss.

Recommendations

Short term, add the minimal information listed above to the project's `README`. This will help users to build, run, and test the project.

Long term, as the project evolves, ensure that the `README` is updated. This will help ensure that the `README` does not communicate incorrect information to users.

References

- Documentation points to do.sh

2. Inadequate testing

Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-DRIFT-2
Target: `.github/workflows/main.yml`, `test-scripts/run-anchor-tests.sh`	

Description

The Anchor tests are not run as part of Drift Protocol's CI process. Moreover, the script responsible for running the Anchor tests does not run all of them. Integrating all Anchor tests into the CI process and updating the script so it runs all tests will help ensure they are run regularly and consistently.

Figure 2.1 shows a portion of the project's `main` GitHub workflow, which runs the project's unit tests. However, the file makes no reference to the project's Anchor tests.

```
- name: Run unit tests
  run: cargo test --lib # run unit tests
```

Figure 2.1: `.github/workflows/main.yml#L52–L53`

Furthermore, the script used to run the Anchor tests runs only some of them. The relevant part of the script appears in figure 2.2. The `test_files` array contains the names of nearly all of the files containing tests in the `tests` directory. However, the array lacks the following entries, and consequently does not run their tests:

- `ksolver.ts`
- `tokenFaucet.ts`

```
test_files=(
  postOnlyAmmFulfillment.ts
  imbalancePerpPnl.ts
  ... # 42 entries
  cancelAllOrders.ts
)
```

Figure 2.2: `test-scripts/run-anchor-tests.sh#L7–L53`

Exploit Scenario

Alice, a Drift Protocol developer, unwittingly introduces a bug into the codebase. The test would be revealed by the Anchor tests. However, because the Anchor tests are not run in CI, the bug goes unnoticed.

Recommendations

Short term:

- Adjust the `main` GitHub workflow so that it runs the Anchor tests.
- Adjust the `run-anchor-tests.sh` script so that it runs all Anchor tests (including those in `ksolver.ts` and `tokenFaucet.ts`).

Taking these steps will help to ensure that all Anchor tests are run regularly and consistently.

Long term, revise the `run-anchor-tests.sh` script so that the `test_files` array is not needed. Move files that do not contain tests into a separate directory, so that only files containing tests remain. Then, run the tests in all files in the `tests` directory. Adopting such an approach will ensure that newly added tests are automatically run.

3. Invalid audit.toml prevents cargo audit from being run

Severity: **Informational**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: TOB-DRIFT-3
Target: `audit.toml`	

Description

The project's `anchor.toml` file contains an invalid key. This makes running `cargo audit` on the project impossible.

The relevant part of the `audit.toml` file appears in figure 3.1. The `packages` key is unrecognized by `cargo audit`. As a result, `cargo audit` produces the error in figure 3.2 when run on the `protocol-v2` repository.

```
[packages]
source = "all" # "all", "public" or "local"
```

Figure 3.1: `.cargo/audit.toml#L27–L28`

```
error: cargo-audit fatal error: parse error: unknown field `packages`, expected one
of `advisories`, `database`, `output`, `target`, `yanked` at line 30 column 1
```

Figure 3.2: Error produced by `cargo audit` when run on the `protocol-v2` repository

Exploit Scenario

A vulnerability is discovered in a `protocol-v2` dependency. A RUSTSEC advisory is issued for the vulnerability, but because `cargo audit` cannot be run on the repository, the vulnerability goes unnoticed. Users suffer financial loss.

Recommendations

Short term, either remove the `packages` table from the `anchor.toml` file or replace it with a table recognized by `cargo audit`. In the project's current state, `cargo audit` cannot be run on the project.

Long term, regularly run `cargo audit` in CI and verify that it runs to completion without producing any errors or warnings. This will help the project receive the full benefits of running `cargo audit` by identifying dependencies with RUSTSEC advisories.

4. Race condition in Drift SDK

Severity: **Undetermined**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-DRIFT-4
Target: `sdk` directory	

Description

A race condition in the Drift SDK causes client programs to operate on non-existent or possibly stale data. The race condition affects many of the project's Anchor tests, making them unreliable. Use of the SDK in production could have financial implications.

When running the Anchor tests, the error in figure 4.1 appears frequently. The `data` field that the error refers to is read by the `getUserAccount` function (figure 4.2). This function tries to read the `data` field from a `DataAndSlot` object obtained by calling `getUserAccountAndSlot` (figure 4.3). That `DataAndSlot` object is set by the `handleRpcResponse` function (figure 4.4).

```
TypeError: Cannot read properties of undefined (reading 'data')
 at User.getUserAccount (sdk/src/user.ts:122:56)
 at DriftClient.getUserAccount (sdk/src/driftClient.ts:663:37)
 at DriftClient.<anonymous> (sdk/src/driftClient.ts:1005:25)
 at Generator.next (<anonymous>)
 at fulfilled (sdk/src/driftClient.ts:28:58)
 at processTicksAndRejections (node:internal/process/task_queues:96:5)
```

Figure 4.1: Error that appears frequently when running the Anchor tests

```
public getUserAccount(): UserAccount {
	return this.accountSubscriber.getUserAccountAndSlot().data;
}
```

Figure 4.2: sdk/src/user.ts#L121–L123

```
public getUserAccountAndSlot(): DataAndSlot<UserAccount> {
	this.assertIsSubscribed();
	return this.userDataAccountSubscriber.dataAndSlot;
}
```

Figure 4.3: sdk/src/accounts/webSocketUserAccountSubscriber.ts#L72–L75

```
handleRpcResponse(context: Context, accountInfo?: AccountInfo<Buffer>): void {
	...
	if (newBuffer && (!oldBuffer || !newBuffer.equals(oldBuffer))) {
```

```
                this.bufferAndSlot = {
                        buffer: newBuffer,
                        slot: newSlot,
                };
                const account = this.decodeBuffer(newBuffer);
                this.dataAndSlot = {
                        data: account,
                        slot: newSlot,
                };
                this.onChange(account);
        }
}
```

Figure 4.4: sdk/src/accounts/webSocketAccountSubscriber.ts#L55–L95

If a developer calls `getUserAccount` but `handleRpcResponse` has not been called since the last time the account was updated, stale data will be returned. If `handleRpcResponse` has *never* been called for the account in question, an error like that shown in figure 4.1 arises.

Note that a developer can avoid the race by calling `WebSocketAccountSubscriber.fetch` (figure 4.5). However, the developer must manually identify locations where such calls are necessary. Errors like the one shown in figure 4.1 appear frequently when running the Anchor tests, which suggests that identifying such locations is nontrivial.

```
async fetch(): Promise<void> {
        const rpcResponse =
                await this.program.provider.connection.getAccountInfoAndContext(
                        this.accountPublicKey,
                        (this.program.provider as AnchorProvider).opts.commitment
                );
        this.handleRpcResponse(rpcResponse.context, rpcResponse?.value);
}
```

Figure 4.5: sdk/src/accounts/webSocketAccountSubscriber.ts#L46–L53

We suspect this problem applies to not just user accounts, but any account fetched via a subscription mechanism (e.g., state accounts or perp market accounts).

Note that despite the apparent race condition, Drift Protocol states that the tests run reliably for them.

Exploit Scenario

Alice, unaware of the race condition, writes client code that uses the Drift SDK. Alice's code unknowingly operates on stale data and proceeds with a transaction, believing it will result in financial gain. However, when processed with actual on-chain data, the transaction results in financial loss for Alice.

Recommendations

Short term, rewrite all account getter functions so that they automatically call `WebSocketAccountSubscriber.fetch`. This will eliminate the need for developers to deal with the race manually.

Long term, investigate whether using a subscription mechanism is actually needed. Another Solana RPC call could solve the same problem yet be more efficient than a subscription combined with a manual fetch.

5. Loose size coupling between function invocation and requirement

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-DRIFT-5
Target: `programs/drift/src/state/events.rs`	

Description

The implementation of the `emit_stack` function relies on the caller to use a sufficiently large buffer space to hold a Base64-encoded representation of the discriminator along with the serialized event. Failure to provide sufficient space will result in an out-of-bounds attempt on either the write operation or the in the `base64::encode_config_slice` call.

```
emit_stack::<_, 424>(order_action_record);
```

Figure 5.1: programs/drift/src/controller/orders.rs#L545

```
pub fn emit_stack<T: AnchorSerialize + Discriminator, const N: usize>(event: T) {
    let mut data_buf = [0u8; N];
    let mut out_buf = [0u8; N];

    emit_buffers(event, &mut data_buf[..], &mut out_buf[..])
}

pub fn emit_buffers<T: AnchorSerialize + Discriminator>(
    event: T,
    data_buf: &mut [u8],
    out_buf: &mut [u8],
) {
    let mut data_writer = std::io::Cursor::new(data_buf);
    data_writer
        .write_all(&<T as Discriminator>::discriminator())
        .unwrap();
    borsh::to_writer(&mut data_writer, &event).unwrap();
    let data_len = data_writer.position() as usize;

    let out_len = base64::encode_config_slice(
        &data_writer.into_inner()[0..data_len],
        base64::STANDARD,
        out_buf,
    );

    let msg_bytes = &out_buf[0..out_len];
    let msg_str = unsafe { std::str::from_utf8_unchecked(msg_bytes) };

    msg!(msg_str);
```

```
}
```

Figure 5.2: programs/drift/src/state/events.rs#L482–L511

Exploit Scenario

A maintainer of the smart contract is unaware of this implicit size requirement and adds a call to `emit_stack` using too small a buffer, or changes are made to a type without a corresponding change to all places where `emit_stack` uses that type. If the changed code is not covered by tests, the problem will manifest during contract operation, and could cause an instruction to panic, thereby reverting the transaction.

Recommendations

Short term, add a size constant to the type, and calculate the amount of space required for holding the respective buffers. This ensures that changes to a type's size can be made throughout the code.

Long term, create a trait to be used by the types with which `emit_stack` is intended to work. This can be used to handle the size of the type, and also any other future requirement for types used by `emit_stack`.

6. The zero-copy feature in Anchor is experimental

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-DRIFT-6
Target: `State structs`	

Description

Several structs for keeping state use Anchor's zero-copy functionality. The Anchor documentation states that this is still an experimental feature that should be used only when Borsh serialization cannot be used without hitting the stack or heap limits.

Exploit Scenario

The Anchor framework has a bug in the zero-copy feature, or updates it with a breaking change, in a way that affects the security model of the Drift smart contract. An attacker discovers this problem and leverages it to steal funds from the contract.

```
#[account(zero_copy)]
#[derive(Default, Eq, PartialEq, Debug)]
#[repr(C)]
pub struct User {
    pub authority: Pubkey,
    pub delegate: Pubkey,
    pub name: [u8; 32],
    pub spot_positions: [SpotPosition; 8],
    pub perp_positions: [PerpPosition; 8],
    pub orders: [Order; 32],
    pub last_add_perp_lp_shares_ts: i64,
    pub total_deposits: u64,
    pub total_withdraws: u64,
    pub total_social_loss: u64,
    // Fees (taker fees, maker rebate, referrer reward, filler reward) and pnl for
perps
    pub settled_perp_pnl: i64,
    // Fees (taker fees, maker rebate, filler reward) for spot
    pub cumulative_spot_fees: i64,
    pub cumulative_perp_funding: i64,
    pub liquidation_margin_freed: u64, // currently unimplemented
    pub liquidation_start_ts: i64,     // currently unimplemented
    pub next_order_id: u32,
    pub max_margin_ratio: u32,
    pub next_liquidation_id: u16,
    pub sub_account_id: u16,
    pub status: UserStatus,
    pub is_margin_trading_enabled: bool,
```

```
    pub padding: [u8; 26],
}
```

Figure 6.1: Example of a struct using zero copy

Recommendations

Short term, evaluate if it is possible to move away from using zero copy without hitting the stack or heap limits, and do so if possible. Not relying on experimental features reduces the risk of exposure to bugs in the Anchor framework.

Long term, adopt a conservative stance by using stable versions of packages and features. This reduces both risk and time spent on maintaining compatibility with code still in flux.

7. Hard-coded indices into account data

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-DRIFT-7
Target: `perp_market_map.rs, spot_market_map.rs`	

Description

The implementations for both `PerpMarketMap` and `SpotMarketMap` use hard-coded indices into the accounts data in order to retrieve the `marked_index` property without having to deserialize all the data.

```
// market index 1160 bytes from front of account
let market_index = u16::from_le_bytes(*array_ref![data, 1160, 2]);
```

Figure 7.1: programs/drift/src/state/perp_market_map.rs#L110–L111

```
let market_index = u16::from_le_bytes(*array_ref![data, 684, 2]);
```

Figure 7.2: programs/drift/src/state/spot_market_map.rs#L174

Exploit Scenario

Alice, a Drift Protocol developer, changes the layout of the structure or the width of the `market_index` property but fails to update one or more of the hard-coded indices. Mallory notices this bug and finds a way to use it to steal funds.

Recommendations

Short term, add `consts` that include the value of the indices and the type size. Also add comments explaining the calculation of the values. This ensures that by updating the constants, all code relying on the operation will retrieve the correct part of the unlying data.

Long term, add an implementation to the struct to unpack the `market_index` from the serialized state. This reduces the maintenance burden of updating the code that accesses data in this way.

8. Missing verification of maker and maker_stats accounts

Severity: **Undetermined**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-DRIFT-8
Target: `programs/drift/src/instructions/user.rs`	

Description

The `handle_place_and_take_perp_order` and `handle_place_and_take_spot_order` functions retrieve two additional accounts that are passed to the instruction: `maker` and `maker_stats`. However, there is no check that the two accounts are linked (i.e., that their `authority` is the same). Due to time constraints, we were unable to determine the impact of this finding.

```
pub fn get_maker_and_maker_stats<'a>(
    account_info_iter: &mut Peekable<Iter<AccountInfo<'a>>>,
) -> DriftResult<(AccountLoader<'a, User>, AccountLoader<'a, UserStats>)> {
    let maker_account_info =
        next_account_info(account_info_iter).or(Err(ErrorCode::MakerNotFound))?;

    validate!(
        maker_account_info.is_writable,
        ErrorCode::MakerMustBeWritable
    )?;

    let maker: AccountLoader<User> =

AccountLoader::try_from(maker_account_info).or(Err(ErrorCode::CouldNotDeserializeMak
er))?;

    let maker_stats_account_info =

next_account_info(account_info_iter).or(Err(ErrorCode::MakerStatsNotFound))?;

    validate!(
        maker_stats_account_info.is_writable,
        ErrorCode::MakerStatsMustBeWritable
    )?;

    let maker_stats: AccountLoader<UserStats> =
        AccountLoader::try_from(maker_stats_account_info)
            .or(Err(ErrorCode::CouldNotDeserializeMakerStats))?;

    Ok((maker, maker_stats))
}
```

Figure 8.1: programs/drift/src/instructions/optional_accounts.rs#L47–L74

Exploit Scenario

Mallory passes two unlinked accounts of the correct type in the places for `maker` and `maker_stats`, respectively. This causes the contract to operate outside of its intended use.

Recommendations

Short term, add a check that the `authority` of the accounts are the same.

Long term, add all code for authentication of accounts to the front of instruction handlers. This increases the clarity of the checks and helps with auditing the authentication.

9. Panics used for error handling

Severity: **Informational**	Difficulty: **High**
Type: Error Reporting	Finding ID: TOB-DRIFT-9
Target: Various files in `programs/drift`	

Description

In several places, the code panics when an arithmetic overflow or underflow occurs. Panics should be reserved for programmer errors (e.g., assertion violations). Panicking on user errors dilutes the utility of the panic operation.

An example appears in figure 9.1. The `adjust_amm` function uses both the question mark operator (?) and `unwrap` to handle errors resulting from "peg" related calculations. An overflow or underflow could result from an invalid input to the function. An error should be returned in such cases.

```
budget_delta_peg = budget_i128
    .safe_add(adjustment_cost.abs())?
    .safe_mul(PEG_PRECISION_I128)?
    .safe_div(per_peg_cost)?;

budget_delta_peg_magnitude = budget_delta_peg.unsigned_abs();
new_peg = if budget_delta_peg > 0 {
   ...
} else if market.amm.peg_multiplier > budget_delta_peg_magnitude {
    market
        .amm
        .peg_multiplier
        .safe_sub(budget_delta_peg_magnitude)
        .unwrap()
} else {
    1
};
```

Figure 9.1: programs/drift/src/math/repeg.rs#L349–L369

Running Clippy with the following command identifies 66 locations in the `drift` package where `expect` or `unwrap` is used:

```
cargo clippy -p drift -- -A clippy::all -W
    clippy::expect_used -W clippy::unwrap_used
```

Many of those uses appear to be related to invalid input.

Exploit Scenario

Alice, a Drift Protocol developer, observes a panic in the Drift Protocol codebase. Alice ignores the panic, believing that it is caused by user error, but it is actually caused by a bug she introduced.

Recommendations

Short term, reserve the use of panics for programmer errors. Have relevant areas of the code return `Result::Err` on user errors. Adopting such a policy will help to distinguish the two types of errors when they occur.

Long term, consider denying the following Clippy lints:

- `clippy::expect_used`
- `clippy::unwrap_used`
- `clippy::panic`

Although this will not prevent all panics, it will prevent many of them.

10. Testing code used in production

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Patching	Finding ID: TOB-DRIFT-10
Target: `programs/drift/src/state/{oracle_map.rs, perp_market.rs}`	

Description

In some locations in the Drift Protocol codebase, testing code is mixed with production code with no way to discern between them. Testing code should be clearly indicated as such and guarded by `#[cfg(test)]` to avoid being called in production.

Examples appear in figures 10.1 and 10.2. The `OracleMap` struct has a `quote_asset_price_data` field that is used only when `get_price_data` is passed a default `Pubkey`. Similarly, the AMM implementation contains functions that are used only for testing and are not guarded by `#[cfg(test)]`.

```
pub struct OracleMap<'a> {
    oracles: BTreeMap<Pubkey, AccountInfoAndOracleSource<'a>>,
    price_data: BTreeMap<Pubkey, OraclePriceData>,
    pub slot: u64,
    pub oracle_guard_rails: OracleGuardRails,
    pub quote_asset_price_data: OraclePriceData,
}

impl<'a> OracleMap<'a> {
    ...
    pub fn get_price_data(&mut self, pubkey: &Pubkey) ->
DriftResult<&OraclePriceData> {
        if pubkey == &Pubkey::default() {
            return Ok(&self.quote_asset_price_data);
        }
```

Figure 10.1: programs/drift/src/state/oracle_map.rs#L22–L47

```
impl AMM {
    pub fn default_test() -> Self {
        let default_reserves = 100 * AMM_RESERVE_PRECISION;
        // make sure tests dont have the default sqrt_k = 0
        AMM {
```

Figure 10.2: programs/drift/src/state/perp_market.rs#L490–L494

Drift Protocol has indicated that the `quote_asset_price_data` field (figure 10.1) *is* used in production. This raises concerns because there is currently no way to set the contents of

this field, and no asset's price is perfectly constant (e.g., even stablecoins' prices fluctuate). For this reason, we have changed this finding's severity from Informational to Undetermined.

Exploit Scenario

Alice, a Drift Protocol developer, introduces code that calls the `default_test` function, not realizing it is intended only for testing. Alice introduces a bug as a result.

Recommendations

Short term, to the extent possible, avoid mixing testing and production code by, for example, using separate data types and storing the code in separate files. When testing and production code must be mixed, clearly mark the testing code as such, and guard it with `#[cfg(test)]`. These steps will help to ensure that testing code is not deployed in production.

Long term, as new code is added to the codebase, ensure that the aforementioned standards are maintained. Testing code is not typically held to the same standards as production code, so it is more likely to include bugs.

11. Inconsistent use of checked arithmetic

Severity: **Undetermined**	Difficulty: **Undetermined**
Type: Data Validation	Finding ID: TOB-DRIFT-11
Target: Various files in `programs/drift`	

Description

In several locations, the Drift Protocol codebase uses unchecked arithmetic. For example, in `calculate_margin_requirement_and_total_collateral_and_liability_info` (figure 11.1), the variable `num_perp_liabilities` is used as an operand in both a checked and an unchecked operation. To protect against overflows and underflows, unchecked arithmetic should be used sparingly.

```
        num_perp_liabilities += 1;
    }

    with_isolated_liability &=
        margin_requirement > 0 && market.contract_tier == ContractTier::Isolated;
}

if num_spot_liabilities > 0 {
    validate!(
        margin_requirement > 0,
        ErrorCode::InvalidMarginRatio,
        "num_spot_liabilities={} but margin_requirement=0",
        num_spot_liabilities
    )?;
}

let num_of_liabilities = num_perp_liabilities.safe_add(num_spot_liabilities)?;
```

Figure 11.1: programs/drift/src/math/margin.rs#L499–L515

Note that adding the following to the crate root will cause Clippy to fail the build whenever unchecked arithmetic is used:

```
#![deny(clippy::integer_arithmetic)]
```

Exploit Scenario

Alice, a Drift Protocol developer, unwittingly introduces an arithmetic overflow bug into the codebase. The bug would have been revealed by the use of checked arithmetic. However, because unchecked arithmetic is used, the bug goes unnoticed.

Recommendations

Short term, add the `#![deny(clippy::integer_arithmetic)]` attribute to the `drift` crate root. Add `#[allow(clippy::integer_arithmetic)]` in rare situations where code is performance critical and its safety can be guaranteed through other means. Taking these steps will reduce the likelihood of overflow or underflow bugs residing in the codebase.

Long term, if additional Solana programs are added to the codebase, ensure the `#![deny(clippy::integer_arithmetic)]` attribute is also added to them. This will reduce the likelihood that newly introduced crates contain overflow or underflow bugs.

12. Inconsistent and incomplete exchange status checks

Severity: **Medium**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-DRIFT-12
Target: `programs/drift/src/instructions/{admin.rs, keeper.rs, user.rs}`, `programs/drift/src/state/state.rs`	

Description

Drift Protocol's representation of the exchange's status has several problems:

- The exchange's status is represented using an `enum`, which does not allow more than one individual operation to be paused (figures 12.1 and 12.2). As a result, an administrator could inadvertently unpause one operation by trying to pause another (figure 12.3).
- The `ExchangeStatus` variants do not map cleanly to exchange operations. For example, `handle_transfer_deposit` checks whether the exchange status is `WithdrawPaused` (figure 12.4). The function's name suggests that the function checks whether "transfers" or "deposits" are paused.
- The `ExchangeStatus` is checked in multiple inconsistent ways. For example, in `handle_update_funding_rate` (figure 12.5), both an `access_control` attribute and the body of the function include a check for whether the exchange status is `FundingPaused`.

```
pub enum ExchangeStatus {
    Active,
    FundingPaused,
    AmmPaused,
    FillPaused,
    LiqPaused,
    WithdrawPaused,
    Paused,
}
```

Figure 12.1: programs/drift/src/state/state.rs#L36–L44

```
#[account]
#[derive(Default)]
#[repr(C)]
pub struct State {
    pub admin: Pubkey,
    pub whitelist_mint: Pubkey,
    ...
```

```
    pub exchange_status: ExchangeStatus,
    pub padding: [u8; 17],
}
```

Figure 12.2: programs/drift/src/state/state.rs#L8–L33

```
pub fn handle_update_exchange_status(
    ctx: Context<AdminUpdateState>,
    exchange_status: ExchangeStatus,
) -> Result<()> {
    ctx.accounts.state.exchange_status = exchange_status;
    Ok(())
}
```

Figure 12.3: programs/drift/src/instructions/admin.rs#L1917–L1923

```
#[access_control(
    withdraw_not_paused(&ctx.accounts.state)
)]
pub fn handle_transfer_deposit(
    ctx: Context<TransferDeposit>,
    market_index: u16,
    amount: u64,
) -> anchor_lang::Result<()> {
```

Figure 12.4: programs/drift/src/instructions/user.rs#L466–L473

```
#[access_control(
    market_valid(&ctx.accounts.perp_market)
    funding_not_paused(&ctx.accounts.state)
    valid_oracle_for_perp_market(&ctx.accounts.oracle, &ctx.accounts.perp_market)
)]
pub fn handle_update_funding_rate(
    ctx: Context<UpdateFundingRate>,
    perp_market_index: u16,
) -> Result<()> {
    ...
    let is_updated = controller::funding::update_funding_rate(
        perp_market_index,
        perp_market,
        &mut oracle_map,
        now,
        &state.oracle_guard_rails,
        matches!(state.exchange_status, ExchangeStatus::FundingPaused),
        None,
    )?;
    ...
}
```

Figure 12.5: programs/drift/src/instructions/keeper.rs#L1027–L1078

The Medium post describing the incident that occurred around May 11, 2022 suggests that the exchange's pausing mechanisms contributed to the incident's subsequent fallout:

> *The protocol did not have a kill-switch where only withdrawals were halted. The protocol was paused in the second pause to prevent a further drain of user funds…*

This suggests that the pausing mechanisms should receive heightened attention to reduce the damage should another incident occur.

Exploit Scenario

Mallory tricks an administrator into pausing funding after withdrawals have already been paused. By pausing funding, the administrator unwittingly unpauses withdrawals.

Recommendations

Short term:

- Represent the exchange's status as a set of flags. This will allow individual operations to be paused independently of one another.
- Ensure exchange statuses map cleanly to the operations that can be paused. Add documentation where there is potential for confusion. This will help ensure developers check the proper exchange statuses.
- Adopt a single approach for checking the exchange's status and apply it consistently throughout the codebase. If an exception must be made for a check, explain why in a comment near that check. Adopting such a policy will reduce the likelihood that a missing check goes unnoticed.

Long term, periodically review the exchange status checks. Since the exchange status checks represent a form of access control, they deserve heightened scrutiny. Moreover, the exchange's pausing mechanisms played a role in past incidents.

13. Spot market access controls are incomplete

Severity: **Informational**	Difficulty: **Undetermined**
Type: Access Controls	Finding ID: TOB-DRIFT-13
Target: `programs/drift/src/instructions/{admin.rs, user.rs}`	

Description

Functions in `admin.rs` involving perpetual markets verify that the market is valid, i.e., not delisted (figure 13.1). However, functions involving spot markets do not include such checks (e.g., figure 13.2). Drift Protocol has indicated that the spot market implementation is incomplete.

```
#[access_control(
    market_valid(&ctx.accounts.perp_market)
)]
pub fn handle_update_perp_market_expiry(
    ctx: Context<AdminUpdatePerpMarket>,
    expiry_ts: i64,
) -> Result<()> {
```

Figure 13.1: programs/drift/src/instructions/admin.rs#L676–L682

```
pub fn handle_update_spot_market_expiry(
    ctx: Context<AdminUpdateSpotMarket>,
    expiry_ts: i64,
) -> Result<()> {
```

Figure 13.2: programs/drift/src/instructions/admin.rs#L656–L660

A similar example concerning whether the exchange is paused appears in figure 13.3 and 13.4.

```
#[access_control(
    exchange_not_paused(&ctx.accounts.state)
)]
pub fn handle_place_perp_order(ctx: Context<PlaceOrder>, params: OrderParams) ->
Result<()> {
```

Figure 13.3: programs/drift/src/instructions/user.rs#L687–L690

```
pub fn handle_place_spot_order(ctx: Context<PlaceOrder>, params: OrderParams) ->
Result<()> {
```

Figure 13.4: programs/drift/src/instructions/user.rs#L1022–L1023

Exploit Scenario

Mallory tricks an administrator into making a call that re-enables an expiring spot market. Mallory profits by trading against the should-be-expired spot market.

Recommendations

Short term, add the missing access controls to the spot market functions in `admin.rs`. This will ensure that an administrator cannot accidentally perform an operation on an expired spot market.

Long term, add tests to verify that each function involving spot markets fails when invoked on an expired spot market. This will increase confidence that the access controls have been implemented correctly.

14. Oracles can be invalid in at most one way

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-DRIFT-14
Target: `programs/drift/src/math/oracle.rs`	

Description

The Drift Protocol codebase represents oracle validity using an `enum`, which does not allow an oracle to be invalid in more than one way. Furthermore, the code that determines an oracle's validity imposes an implicit hierarchy on the ways an oracle could be invalid. This design is fragile and likely to cause future problems.

The `OracleValidity` `enum` is shown in figure 14.1, and the code that determines an oracle's validity is shown in figure 14.2. Note that if an oracle is, for example, both "too volatile" and "too uncertain," the oracle will be labeled simply `TooVolatile`. A caller that does not account for this fact and simply checks whether an oracle is `TooUncertain` could overlook oracles that are both "too volatile" and "too uncertain."

```
pub enum OracleValidity {
    Invalid,
    TooVolatile,
    TooUncertain,
    StaleForMargin,
    InsufficientDataPoints,
    StaleForAMM,
    Valid,
}
```

Figure 14.1: programs/drift/src/math/oracle.rs#L21–L29

```
pub fn oracle_validity(
    last_oracle_twap: i64,
    oracle_price_data: &OraclePriceData,
    valid_oracle_guard_rails: &ValidityGuardRails,
) -> DriftResult<OracleValidity> {
    ...
    let oracle_validity = if is_oracle_price_nonpositive {
        OracleValidity::Invalid
    } else if is_oracle_price_too_volatile {
        OracleValidity::TooVolatile
    } else if is_conf_too_large {
        OracleValidity::TooUncertain
    } else if is_stale_for_margin {
        OracleValidity::StaleForMargin
```

```
    } else if !has_sufficient_number_of_data_points {
        OracleValidity::InsufficientDataPoints
    } else if is_stale_for_amm {
        OracleValidity::StaleForAMM
    } else {
        OracleValidity::Valid
    };

    Ok(oracle_validity)
}
```

Figure 14.2: programs/drift/src/math/oracle.rs#L163–L230

Exploit Scenario

Alice, a Drift Protocol developer, is unaware of the implicit hierarchy among the `OracleValidity` variants. Alice writes code like `oracle_validity != OracleValidity::TooUncertain` and unknowingly introduces a bug into the codebase.

Recommendations

Short term, represent oracle validity as a set of flags. This will allow oracles to be invalid in more than one way, which will result in more robust and maintainable code.

Long term, thoroughly test all code that relies on oracle validity. This will help ensure the code's correctness following the aforementioned change.

15. Code duplication

Severity: **Informational**	Difficulty: **High**
Type: Patching	Finding ID: TOB-DRIFT-15
Target: Various files in `programs/drift`	

Description

Various files in the `programs/drift` directory contain duplicate code, which can lead to incomplete fixes or inconsistent behavior (e.g., because the code is modified in one location but not all).

As an example, the code in figure 15.1 appears nearly verbatim in the functions `liquidate_perp`, `liquidate_spot`, `liquidate_borrow_for_perp_pnl`, and `liquidate_perp_pnl_for_deposit`.

```
// check if user exited liquidation territory
let (intermediate_total_collateral, intermediate_margin_requirement_with_buffer) =
    if !canceled_order_ids.is_empty() || lp_shares > 0 {
        ... // 37 lines
        (
            intermediate_total_collateral,
            intermediate_margin_requirement_plus_buffer,
        )
    } else {
        (total_collateral, margin_requirement_plus_buffer)
    };
```

Figure 15.1: programs/drift/src/controller/liquidation.rs#L201–L246

In some places, the text itself is not obviously duplicated, but the logic it implements is clearly duplicated. An example appears in figures 15.2 and 15.3. Such "logical" code duplication suggests the code does not use the right abstractions.

```
// Update Market open interest
if let PositionUpdateType::Open = update_type {
    if position.quote_asset_amount == 0 && position.base_asset_amount == 0 {
        market.number_of_users = market.number_of_users.safe_add(1)?;
    }

    market.number_of_users_with_base =
market.number_of_users_with_base.safe_add(1)?;
} else if let PositionUpdateType::Close = update_type {
    if new_base_asset_amount == 0 && new_quote_asset_amount == 0 {
        market.number_of_users = market.number_of_users.safe_sub(1)?;
```

```
    }

    market.number_of_users_with_base =
market.number_of_users_with_base.safe_sub(1)?;
}
```

Figure 15.2: programs/drift/src/controller/position.rs#L162–L175

```
if position.quote_asset_amount == 0 && position.base_asset_amount == 0 {
    market.number_of_users = market.number_of_users.safe_add(1)?;
}

position.quote_asset_amount = position.quote_asset_amount.safe_add(delta)?;

market.amm.quote_asset_amount =
market.amm.quote_asset_amount.safe_add(delta.cast()?)?;

if position.quote_asset_amount == 0 && position.base_asset_amount == 0 {
    market.number_of_users = market.number_of_users.safe_sub(1)?;
}
```

Figure 15.3: programs/drift/src/controller/position.rs#L537–L547

Exploit Scenario

Alice, a Drift Protocol developer, is asked to fix a bug in `liquidate_perp`. Alice does not realize that the bug also applies to `liquidate_spot`, `liquidate_borrow_for_perp_pnl`, and `liquidate_perp_pnl_for_deposit`, and fixes the bug in only `liquidate_perp`. Eve discovers that the bug is not fixed in one of the other three functions and exploits it.

Recommendations

Short term:

- Refactor `liquidate_perp`, `liquidate_spot`, `liquidate_borrow_for_perp_pnl`, and `liquidate_perp_pnl_for_deposit` to eliminate the code duplication. This will reduce the likelihood of an incomplete fix for a bug affecting more than one of these functions.
- Identify cases where the code uses the same logic, and implement abstractions to capture that logic. Ensure that code that relies on such logic uses the new abstractions. Consolidating similar pieces of code will make the overall codebase easier to reason about.

Long term, adopt code practices that discourage code duplication. This will help to prevent this problem from recurring.

16. Inconsistent use of integer types

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-DRIFT-16
Target: Various files in `programs/drift`	

Description

The Drift Protocol codebase uses integer types inconsistently; data of similar kinds is represented using differently sized types or types with different signedness. Conversions from one integer type to another present an opportunity for the contracts to fail and should be avoided.

For example, the pow method expects a u32 argument. However, in some places u128 values must be cast to u32 values, even though those values are intended to be used as exponents (figures 16.1, 16.2, and 16.3).

```
let expo_diff = (spot_market.insurance_fund.shares_base -
insurance_fund_stake.if_base)
    .cast::<u32>()?;

let rebase_divisor = 10_u128.pow(expo_diff);
```

Figure 16.1: programs/drift/src/controller/insurance.rs#L154–L157

```
#[zero_copy]
#[derive(Default, Eq, PartialEq, Debug)]
#[repr(C)]
pub struct InsuranceFund {
    pub vault: Pubkey,
    pub total_shares: u128,
    pub user_shares: u128,
    pub shares_base: u128,      // exponent for lp shares (for rebasing)
    pub unstaking_period: i64, // if_unstaking_period
    pub last_revenue_settle_ts: i64,
    pub revenue_settle_period: i64,
    pub total_factor: u32, // percentage of interest for total insurance
    pub user_factor: u32,  // percentage of interest for user staked insurance
}
```

Figure 16.2: programs/drift/src/state/spot_market.rs#L352–L365

```
#[account(zero_copy)]
#[derive(Default, Eq, PartialEq, Debug)]
#[repr(C)]
```

```
pub struct InsuranceFundStake {
    pub authority: Pubkey,
    if_shares: u128,
    pub last_withdraw_request_shares: u128, // get zero as 0 when not in escrow
    pub if_base: u128,                      // exponent for if_shares decimal places
(for rebase)
    pub last_valid_ts: i64,
    pub last_withdraw_request_value: u64,
    pub last_withdraw_request_ts: i64,
    pub cost_basis: i64,
    pub market_index: u16,
    pub padding: [u8; 14],
}
```

Figure 16.3: programs/drift/src/state/insurance_fund_stake.rs#L10–L24

The following command reveals 689 locations where the `cast` method appears to be used:

```
grep -r -I '\.cast\>' programs/drift
```

Each such use could lead to a denial of service if an attacker puts the contract into a state where the `cast` always errors. Many of these uses could be eliminated by more consistent use of integer types.

Note that Drift Protocol has indicated that some of the observed inconsistencies are related to reducing rent costs.

Exploit Scenario

Mallory manages to put the contract into a state such that one of the nearly 700 uses of `cast` always returns an error. The contract becomes unusable for Alice, who needs to execute a code path involving the vulnerable `cast`.

Recommendations

Short term, review all uses of `cast` to see which might be eliminated by changing the types of the operands. This will reduce the overall number of `casts` and reduce the likelihood that one could lead to denial of service.

Long term, as new code is introduced into the codebase, review the types used to hold similar kinds of data. This will reduce the likelihood that new `casts` are needed.

17. Use of opaque constants in tests

Severity: **Informational**	Difficulty: **High**
Type: Testing	Finding ID: TOB-DRIFT-17
Target: `programs/drift/src/controller/liquidation/tests.rs`	

Description

Several of the Drift Protocol tests use constants with no explanation for how they were derived, which makes it difficult to assess whether the tests are functioning correctly.

Ten examples appear in figure 17.1. In each case, a variable or field is compared against a constant consisting of 6–12 random-looking digits. Without an explanation for how these digits were obtained, it is difficult to tell whether the constant expresses the correct value.

```
assert_eq!(user.spot_positions[0].scaled_balance, 45558159000);
assert_eq!(user.spot_positions[1].scaled_balance, 406768999);
...
assert_eq!(margin_requirement, 44744590);
assert_eq!(total_collateral, 45558159);
assert_eq!(margin_requirement_plus_buffer, 45558128);
...
assert_eq!(token_amount, 406769);
assert_eq!(token_value, 40676900);
assert_eq!(strict_token_value_1, 4067690); // if oracle price is more favorable than
twap
...
assert_eq!(liquidator.spot_positions[0].scaled_balance, 159441841000);
...
assert_eq!(liquidator.spot_positions[1].scaled_balance, 593824001);
```

Figure 17.1: programs/drift/src/controller/liquidation/tests.rs#L1618–L1687

Exploit Scenario

Mallory discovers that a constant used in a Drift Protocol test was incorrectly derived and that the tests were actually verifying incorrect behavior. Mallory uses the bug to siphon funds from the Drift Protocol exchange.

Recommendations

Short term, where possible, compute values using an explicit formula rather than an opaque constant. If using an explicit formula is not possible, include a comment explaining how the constant was derived. This will help to ensure that correct behavior is being tested for. Moreover, the process of giving such explicit formulas could reveal errors.

Long term, write scripts to identify constants with high entropy, and run those scripts as part of your CI process. This will help to ensure the aforementioned standards are maintained.

18. Accounts from contexts are not always used by the instruction

Severity: **Informational**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-DRIFT-18
Target: `programs/drift/src/instructions/admin.rs`	

Description

The context definition for the `initialize` instruction defines a `drift_signer` account. However, this account is not used by the instruction. It appears to be a remnant used to pass the address of the state PDA account; however, the need to do this was eliminated by the use of `find_program_address` to calculate the address. Also, in the `initialize_insurance_fund_stake` instruction, the `spot_market`, `user_stats`, and `state` accounts from the context are not used by the instruction.

```
#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,
    #[account(
        init,
        seeds = [b"drift_state".as_ref()],
        space = std::mem::size_of::<State>() + 8,
        bump,
        payer = admin
    )]
    pub state: Box<Account<'info, State>>,
    pub quote_asset_mint: Box<Account<'info, Mint>>,
    /// CHECK: checked in `initialize`
    pub drift_signer: AccountInfo<'info>,
    pub rent: Sysvar<'info, Rent>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
}
```

Figure 18.1: programs/drift/src/instructions/admin.rs#L1989–L2007

Exploit Scenario

Alice, a Drift Protocol developer, assumes that the `drift_signer` account is used by the instruction, and she uses a different address for the account, expecting this account to hold the contract state after the `initialize` instruction has been called.

Recommendations

Short term, remove the unused account from the context. This eliminates the possibility of confusion around the use of the accounts.

Long term, employ a process where a refactoring of an instruction's code is followed by a review of the corresponding context definition. This ensures that the context is in sync with the instruction handlers.

19. Unaligned references are allowed

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-DRIFT-19
Target: `programs/drift/src/lib.rs`	

Description
The Drift Protocol codebase uses the `#![allow(unaligned_references)]` attribute. This allows the use of unaligned references throughout the program and could mask serious problems in future updates to the contract.

```
#![allow(clippy::too_many_arguments)]
#![allow(unaligned_references)]
#![allow(clippy::bool_assert_comparison)]
#![allow(clippy::comparison_chain)]
```

Figure 19.1: programs/drift/src/lib.rs#L1–L4

Exploit Scenario
Alice, a Drift Protocol developer, accidentally introduces errors caused by the use of unaligned references, affecting the contract operation and leading to a loss of funds.

Recommendations
Short term, remove the attributes. This ensures that the check for unaligned references correctly flag such cases.

Long term, be conservative with the use of attributes used to suppress warnings or errors throughout the codebase. If possible, apply them to only the minimum possible amount of code. This minimizes the risk of problems stemming from the suppressed checks.

20. Size of created accounts derived from in-memory representation

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-DRIFT-20
Target: Files in `/programs/drift/src/state/`	

Description

When state accounts are initialized, the size of the account is set to `std::mem::size_of::<ACCOUNT_TYPE>() + 8`, where the eight extra bytes are used for the discriminator. The structs for the state types all have a trailing field with padding, seemingly to ensure the account size is aligned to eight bytes and to determine the size of the account. In other places, the code relies on the `size_of` function to determine the type of accounts passed to the instruction.

While we could not find any security-related problem with the scheme today, this does mean that every account's in-memory representation is inflated by the amount of padding, which could become a problem with respect to the limitation of the stack or heap size. Furthermore, if any of the accounts are updated in such a way that the `repr(C)` layout size differs from the Anchor space reference, it could cause a problem. For example, if the `SpotMarket` struct is changed so that its in-memory representation is smaller than the required Anchor size, the `initialize_spot_market` would fail because the created account would be too small to hold the serialized representation of the data.

```
#[account]
#[derive(Default)]
#[repr(C)]
pub struct State {
    pub admin: Pubkey,
    pub whitelist_mint: Pubkey,
    pub discount_mint: Pubkey,
    pub signer: Pubkey,
    pub srm_vault: Pubkey,
    pub perp_fee_structure: FeeStructure,
    pub spot_fee_structure: FeeStructure,
    pub oracle_guard_rails: OracleGuardRails,
    pub number_of_authorities: u64,
    pub number_of_sub_accounts: u64,
    pub lp_cooldown_time: u64,
    pub liquidation_margin_buffer_ratio: u32,
    pub settlement_duration: u16,
    pub number_of_markets: u16,
```

```
    pub number_of_spot_markets: u16,
    pub signer_nonce: u8,
    pub min_perp_auction_duration: u8,
    pub default_market_order_time_in_force: u8,
    pub default_spot_auction_duration: u8,
    pub exchange_status: ExchangeStatus,
    pub padding: [u8; 17],
}
```

Figure 20.1: The `State` struct, with corresponding padding

```
#[account(
    init,
    seeds = [b"drift_state".as_ref()],
    space = std::mem::size_of::<State>() + 8,
    bump,
    payer = admin
)]
pub state: Box<Account<'info, State>>,
```

Figure 20.2: The creation of the `State` account, using the in-memory size

```
if data.len() < std::mem::size_of::<UserStats>() + 8 {
    return Ok((None, None));
}
```

Figure 20.3: An example of the in-memory size used to determine the account type

Exploit Scenario

Alice, a Drift Protocol developer, unaware of the implicit requirements of the in-memory size, makes changes to a state account's structure or adds a state structure account such that the in-memory size is smaller than the size needed for the serialized data. As a result, instructions in the contract that save data to the account will fail.

Recommendations

Short term, add an implementation to each state struct that returns the size to be used for the corresponding Solana account. This avoids the overhead of the padding and removes the dependency on assumption about the in-memory size.

Long term, avoid using assumptions about in-memory representation of type within programs created in Rust. This ensures that changes to the representation do not affect the program's operation.

Summary of Recommendations

Drift Protocol is a distributed exchange with multiple planned iterations. Trail of Bits recommends that Drift Protocol address the findings detailed in this report as they continue to secure their exchange.

The following is a summary of this report's main recommendations:

- Ensure that the Anchor tests function reliably, and run them as part of the project's CI process. Require the tests to pass before merging changes into the codebase. This will greatly reduce the possibility of bugs being introduced into the codebase. (TOB-DRIFT-2, TOB-DRIFT-4)
- Reduce the use of opaque constants in tests; prefer explicit formulae instead. This will increase confidence that the tests are verifying correct behavior. (TOB-DRIFT-17)
- Enforce consistency in how the exchange's status is checked. This will make it easier to verify that the correct checks are performed for the correct operations. (TOB-DRIFT-12)
- Implement access controls for the spot markets. Try to use the same patterns used for the perp markets' access controls. The absence of access controls could allow the spot markets to be exploited. Using the same patterns used for the perp markets will make it easier to verify the spot market's access controls' correctness. (TOB-DRIFT-13)
- Establish greater consistency among the uses of integer types. Try to use the same integer types for the similar kinds of data (e.g., amounts, prices, conversion rates, etc.). This will reduce the number cast operations required, and reduce the likelihood that any operation using them could fail. (TOB-DRIFT-16)
- Look for opportunities to consolidate code. Identify and eliminate code that has been copied and pasted. When similar code resides in two different functions, try to determine whether that code belongs in a third function. These steps will produce code that is easier to maintain and reason about. (TOB-DRIFT-15)

We would like to emphasize the last bullet. As mentioned under Coverage Limitations, much of the code is written in an inconsistent style, and it is difficult to discern patterns from the codebase. We recommend that Drift Protocol consider what patterns they would like the code to exhibit (e.g., where/how various checks should be performed), and refactor the code so that it exhibits them. Appendix D contains additional recommendations for improving the codebase's readability.

Although we found no high-severity vulnerabilities, we recommend seeking a re-review of the code after it has been refactored/redesigned but before deployment, due to the nature and scope of the recommended changes.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Front-Running Resistance	The system's resistance to front-running attacks
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Non-Security-Related Findings

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- Some statements are missing an ending semicolon. Use `cargo clippy -- -A clippy::all -W clippy::semicolon_if_nothing_returned` to identify the lines.
- The `liquidate_perp` function contains a local variable named `user_order_id`. However, when instantiating the Order struct, this variable is not used for the `user_order_id` field of the struct, but for the `order_id` field. Rename the variable `order_id` to better match its use.
- In several places, variables with leading underscores are used. Per the following Clippy warning, "a leading underscore signals that a binding will not be used":

```
warning: used binding `_base_asset_amount` which is prefixed with an
underscore. A leading underscore signals that a binding will not be used
    --> programs/drift/src/controller/orders.rs:3050:56
     |
3050 |         base_asset_amount =
base_asset_amount.safe_add(_base_asset_amount)?;
     |
^^^^^^^^^^^^^^^^^
     |
     = note: `-W clippy::used-underscore-binding` implied by `-W
clippy::pedantic`
     = help: for further information visit
https://rust-lang.github.io/rust-clippy/master/index.html#used_underscore
_binding
```

- In several places, the following is pattern is used to change the type of an error:

```
match loader.load() {
    Ok(perp_market) => Ok(perp_market),
    Err(e) => {
        let caller = Location::caller();
        msg!("{:?}", e);
        msg!(
            "Could not load perp market {} at {}:{}",
            market_index,
            caller.file(),
            caller.line()
        );
        Err(ErrorCode::UnableToLoadPerpMarketAccount)
    }
}
```

Such code could be written more concisely using `map_err`, as follows:

```
loader.load().map_err(|e| {
    let caller = Location::caller();
    msg!("{:?}", e);
    msg!(
        "Could not load perp market {} at {}:{}",
        market_index,
        caller.file(),
        caller.line()
    );
    ErrorCode::UnableToLoadPerpMarketAccount
})
```

- In the following error message, “greater or equal to” should be “greater than”:

```
validate!(
    revenue_amount <= depositors_amount,
    ErrorCode::SpotMarketVaultInvariantViolated,
    "revenue_amount={} greater or equal to the depositors_amount={}
(depositors_claim={}, spot_market.deposit_balance={})",
    revenue_amount,
    depositors_amount,
    depositors_claim,
    spot_market.deposit_balance
)?;
```

- Within the Drift AMM documentation, the following formulae could be simplified:

```
bid_quote_reserve = quote_reserve - (quote_reserve / (100%/short_spread))
ask_quote_reserve = quote_reserve + (quote_reserve / (100%/long_spread))
```

Specifically, they could be rewritten as:

```
bid_quote_reserve = quote_reserve * (1 - short_spread)
ask_quote_reserve = quote_reserve * (1 + long_spread)
```

- In several places, a fraction is represented as a pair of fields, one for a numerator and one for a denominator. For example:

```
pub struct FeeTier {
    pub fee_numerator: u32,
    pub fee_denominator: u32,
    pub maker_rebate_numerator: u32,
    pub maker_rebate_denominator: u32,
    pub referrer_reward_numerator: u32,
    pub referrer_reward_denominator: u32,
    pub referee_fee_numerator: u32,
    pub referee_fee_denominator: u32,
}
```

The code would be clearer if each such pair of fields were replaced with a fraction data type.

D. Code Quality Recommendations

This appendix contains additional recommendations for improving the Drift Protocol codebase's readability. The appendix specifically addresses project layout, function comments, function complexity, and general inconsistencies. Functions involving a user's liquidation status are addressed in some detail.

Before performing any refactoring based on these recommendations, we strongly encourage taking the following steps:

- Ensure that the project's Anchor tests run reliably.
- Incorporate the Anchor tests into the project's CI process.
- Require all tests to pass on a code change before merging the change into the codebase.

Project Layout

The project's layout is unclear and undocumented. In particular, the purposes of the `controller`, `math`, and `verification` subdirectories are unclear. These subdirectories have several filenames in common. However, for a function X and file Y, it is unclear whether X should reside in `controller/Y`, `math/Y`, or `verification/Y`. The lack of a clear rationale for the layout makes navigating the project difficult, as it can be hard to locate any given component or function.

Within the `programs/drift` folder, several files are devoted to tests. Most of these are named `tests.rs`, but some are not, e.g.:

- `controller/pnl/delisting.rs`
- `controller/orders/amm_jit_tests.rs`

In at least two cases, a test file contains a redundant module named `test`:

- `math/amm_spread/tests.rs`
- `math/margin/tests.rs`

Function Comments

It is a common Rust convention to precede a function with a comment describing the function's purpose and how it works. However, the Drift Protocol codebase mostly does not follow this convention; the `drift` crate contains approximately 778 functions, and only 34 of those are preceded by descriptive comments.

Function Complexity

The following quote is from the book "The Pragmatic Programmer":

Design components that are self-contained, independent, and have a single, well-defined purpose.

We argue that the Drift Protocol codebase does not adhere to this principle.

A function's line count is one heuristic for its complexity. When run on the `drift` crate, Clippy's `too-many-lines` lint produces 29 warnings. Thus, there are at least 29 functions that should be considered for refactoring into smaller functions.

Another heuristic for function complexity is long function names. While descriptive function names are good, a long name can indicate a function whose operation is overly complex, or whose integration into the larger system has not been well thought out.

Several functions in the Drift Protocol codebase have exceedingly long names. The following table includes the ten longest lengths and the number of functions with a name of that length (with tests filtered out):[1]

Function name length	Number of functions
68	1
65	1
56	1
53	2
52	3
51	2
50	1
49	7
48	1
47	3

Table D.1: The ten longest function name lengths and the number of functions with a name of that length

The function with the longest name (68 characters) is the following:

[1] Following the removal of all test files: `find . -name '*.rs' -exec sed -n 's/^.*fn \([a-z0-9_]\+\).*$/\1/;T;p' {} \; | while read X; do echo -n "$X" | wc -c; done | sort -n | uniq -c | sort -k2 -n -r | head -n 10`

`calculate_margin_requirement_and_total_collateral_and_liability_info`

The function is rather long (277 lines). Its name begs whether there should instead exist functions to compute the following:

- Margin requirement
- Total collateral
- Total liability

Note that addressing this issue is *not* merely about renaming functions.

Example: user liquidation status
As examples of functions that are overly complex or whose integration into the larger system does not appear to have been well thought out, consider the three functions whose name includes `being_liquidated`:

- `User::is_being_liquidated`
- `is_user_being_liquidated`
- `validate_user_not_being_liquidated`

We address each of these functions individually.

`User::is_being_liquidated` returns `true` whenever the user's status is `UserStatus::BeingLiquidated` *or* `UserStatus::Bankrupt`. The function's name suggests that the function should check *only* for `UserStatus::BeingLiquidated`, and *not* `UserStatus::Bankrupt`. Also note that there is a `User::is_bankrupt` function, which (as its name suggests) checks *only* for `UserStatus::Bankrupt`.

`is_user_being_liquidated` performs a calculation independent of a user's status. Note the similarity to the just-described function, which suggests that the two functions should perform similar operations, though they do not.

`validate_user_not_being_liquidated` (figure D.1) first calls `User::is_being_liquidated`. If the call returns `false`, `validate_user_not_being_liquidated` returns `Ok(())`. Otherwise, the function calls `is_user_being_liquidated`. If the latter call returns `true`, the function returns an error. If the latter call returns `false`, the function updates the user's status to `UserStatus::Active` and returns `Ok(())`.

```
pub fn validate_user_not_being_liquidated(
    user: &mut User,
    market_map: &PerpMarketMap,
    spot_market_map: &SpotMarketMap,
    oracle_map: &mut OracleMap,
    liquidation_margin_buffer_ratio: u32,
) -> DriftResult {
```

```
    if !user.is_being_liquidated() {
        return Ok(());
    }

    let is_still_being_liquidated = is_user_being_liquidated(
        user,
        market_map,
        spot_market_map,
        oracle_map,
        liquidation_margin_buffer_ratio,
    )?;

    if is_still_being_liquidated {
        return Err(ErrorCode::UserIsBeingLiquidated);
    } else {
        user.status = UserStatus::Active;
    }

    Ok(())
}
```

Figure D.1: programs/drift/src/math/liquidation.rs#L217–L243

There are several problems with `validate_user_not_being_liquidated`:

- The verb "validate" suggests that the function performs only validation. However, the function actually performs a state change. A verb like "update" or "refresh" would more accurately reflect what the function does.
- The function has essentially no effect when the user's status is `UserStatus::Active`. This has the potential for confusion. A user that is active is not being liquidated. Thus, the function's name suggests such a status should be "validated." In fact, however, such a status is trusted as being accurate.
- `validate_user_not_being_liquidated` inherits the problems of `User::is_being_liquidated` by calling it. That is, `validate_user_not_being_liquidated` proceeds to call `is_user_being_liquidated` if the user's status is `UserStatus::BeingLiquidated` *or* `UserStatus::Bankrupt`.

To summarize, a developer must keep at least three pieces of information in mind when reviewing calls to `validate_user_not_being_liquidated`:

- It performs a state change.
- It short circuits (i.e., has essentially no effect) when the user's status is `UserStatus::Active`.
- It treats `UserStatus::BeingLiquidated` and `UserStatus::Bankrupt` the same, despite mentioning only "being liquidated" in the name.

Having to remember such facts creates undue cognitive load. Generally speaking, such cognitive load can be reduced by, e.g., ensuring each function performs a single, logically coherent task, and choosing function names that accurately reflect those tasks.

Finally, none of the functions discussed in this section are documented. The fact that these functions are complex and could be misused increases the need for them to be documented.

General Inconsistencies

As mentioned under Coverage Limitations, much of the code is written in an inconsistent style. This issue is more about consistency than style. That is, changes in style from one part of the code to the next can be jarring, thereby making the code harder to read.

To illustrate this point, consider the calls to `validate_user_not_being_liquidated` (mentioned in the previous section). The function is called in seven places. In each place, the context is significantly different. This can be seen, for example, in the variation in the calls' line offsets within their enclosing function bodies (table D.2).

Call to `validate_user_not_being_liquidated`	Line offset within enclosing function
`src/controller/orders.rs:100`	6
`src/controller/orders.rs:660`	59
`src/controller/orders.rs:2104`	41
`src/controller/orders.rs:2337`	6
`src/controller/orders.rs:2687`	56
`src/controller/orders.rs:3870`	41
`src/instructions/user.rs:1327`	20

Table D.2: The seven calls to `validate_user_not_being_liquidated` and their line offsets within their enclosing function bodies

The differences in context make it difficult to tell whether calls to `validate_user_not_being_liquidated` are missing, and whether

`validate_user_not_being_liquidated` is being used correctly. If, for example, `validate_user_not_being_liquidated` was always the first function called in an enclosing function, it would be easier to tell whether such a call was missing.

To add to the above:

- Three of the seven calls are *preceded* by a check of `user.is_bankrupt()`. Four of the seven calls are *followed* by a check of `user.is_bankrupt()`.

Recall from the previous section that `validate_user_not_being_liquidated` treats `UserStatus::Bankrupt` specially. Moreover, `validate_user_not_being_liquidated` performs a state change. Thus, deciding whether the seven calls to `validate_user_not_being_liquidated` could be moved before or after the call to `user.is_bankrupt()` is non-trivial.

Finally, for reasons that are unclear:

- Five of the seven calls return an error on failure. Two of the seven calls return `Ok(0)` on failure.

To summarize, the irregular way that `validate_user_not_being_liquidated` is now called makes it difficult to tell whether `validate_user_not_being_liquidated` is being used correctly, and whether it is used in all places where needed. Moreover, the fact that `validate_user_not_being_liquidated` performs non-obvious state changes makes imposing consistency on its uses more difficult.

Style Guides

The following style guides provide recommendations beyond those of this appendix:

- The Pragmatic Programmer documents "processes that are virtually universal, and ideas that are almost axiomatic" on design, project management, and coding.
- Rust API Guidelines is "a set of recommendations on how to design and present APIs for the Rust programming language."
- The Rust Reference contains many examples of idiomatic Rust code.
- Many of Clippy's pedantic lints flag code that could be written in a simpler or more idiomatic way.

E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From January 23 to January 25, 2023, Trail of Bits reviewed the fixes and mitigations implemented by Drift Protocol for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, Drift Protocol has resolved 11 of the issues described in this report, has partially resolved two issues, and has not resolved the remaining seven issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	No build instructions	Informational	Resolved
2	Inadequate testing	Informational	Resolved
3	Invalid audit.toml prevents cargo audit from being run	Informational	Resolved
4	Race condition in Drift SDK	Undetermined	Resolved
5	Loose size coupling between function invocation and requirement	Informational	Resolved
6	The zero-copy feature in Anchor is experimental	Informational	Unresolved
7	Hardcoded indices into account data	Informational	Resolved
8	Missing verification of maker and maker_stats accounts	Undetermined	Resolved
9	Panics used for error handling	Informational	Resolved

10	Testing code used in production	Undetermined	Partially Resolved
11	Inconsistent use of checked arithmetic	Undetermined	Unresolved
12	Inconsistent and incomplete exchange status checks	Medium	Resolved
13	Spot market access controls are incomplete	Informational	Partially Resolved
14	Oracles can be invalid in at most one way	Informational	Unresolved
15	Code duplication	Informational	Unresolved
16	Inconsistent use of integer types	Informational	Unresolved
17	Use of opaque constants in tests	Informational	Unresolved
18	Accounts from contexts are not always used by the instruction	Informational	Unresolved
19	Unaligned references are allowed	Informational	Resolved
20	Size of created accounts derived from in-memory representation	Informational	Resolved

Detailed Fix Review Results

TOB-DRIFT-1: No build instructions

Resolved in 5209b9bc17e81fe3c11e9817d98833d0eaf94fd1. The project's README now includes instructions for building it, and for running its Rust and Anchor tests. We verified that the provided commands work as described.

TOB-DRIFT-2: Inadequate testing

Resolved in af85e4c518dfeb70d80e30fb8544f53c116c973c (which also resolves TOB-DRIFT-4 below). The Anchor tests are now run as part of CI. We reviewed recent GitHub logs to verify that the tests are capable of passing in CI.

For reasons we did not investigate, some test files were removed (`adminWidthdraw.ts`, `ksolver,ts`, and `tokenFaucet.ts`).

Also, the `run-anchor-tests.sh` script still uses the `test_files` array (figure E.1). We continue to recommend that the script be revised so that the array is not needed.

```
test_files=(
  postOnlyAmmFulfillment.ts
  imbalancePerpPnl.ts
  ...
  cancelAllOrders.ts
)
```

Figure E.1: test-scripts/run-anchor-tests.sh#L7-L52

TOB-DRIFT-3: Invalid audit.toml prevents cargo audit from being run

Resolved in 0df896decaf21649fd772c709837d8c1b8d44452. The offending `audit.toml` entry was removed, and `cargo audit` now completes without error when run on the project.

TOB-DRIFT-4: Race condition in Drift SDK

Resolved in af85e4c518dfeb70d80e30fb8544f53c116c973c (which also resolves TOB-DRIFT-2 above). `forceGetUserAccount` functions were added to `user.ts` and `driftClient.ts`. These new functions call `WebSocketAccountSubscriber.fetch` before accessing `UserAccount` data. Calls to `WebSocketAccountSubscriber.fetch` were added in other places as well. The Anchor tests now pass, suggesting such calls were inserted where needed.

TOB-DRIFT-5: Loose size coupling between function invocation and requirement

Resolved in 5f25f2a2d5548d9e82933e3d907fd01354e04a7a (which also resolves TOB-DRIFT-20 below). A `Size` trait was added that allows a type to specify its storage size in a `SIZE` constant. Tests are used to verify that the values of types' `SIZE` constants match the sizes of the types' in-memory representations. The use of `emit_stack` was updated to use `OrderActionRecord::SIZE` instead of a hard-coded constant.

TOB-DRIFT-6: The zero-copy feature in Anchor is experimental

Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-7: Hardcoded indices into account data

Resolved in 8e4f15771cce51f6c74628c19b74c5e83c51ed69. A `MarketIndexOffset` trait was added that allows a type to specify its market index offset in a `MARKET_INDEX_OFFSET` constant. The trait is implemented for the `PerpMarket` and `SpotMarket` types. Tests are used to help verify that the named constants are set correctly (see figure E.2).

```
#[test]
fn spot_market() {
    let mut spot_market = SpotMarket {
        market_index: 11,
        ..SpotMarket::default()
    };
    create_anchor_account_info!(spot_market, SpotMarket, spot_market_account_info);

    let data = spot_market_account_info.try_borrow_data().unwrap();
    let market_index =
        u16::from_le_bytes(*array_ref![data, SpotMarket::MARKET_INDEX_OFFSET, 2]);
    assert_eq!(market_index, spot_market.market_index);
}
```

Figure E.2: programs/drift/src/state/traits/tests.rs#L76-L88

While we consider the issue resolved, the tests could be further improved to help increase confidence in the constants' values. Specifically, assertions could be added to verify that the market index changes during the test. For example, the following line could be added as the first line of the `spot_market` test:

```
assert_ne!(SpotMarket::default().market_index, 11);
```

TOB-DRIFT-8: Missing verification of maker and maker_stats accounts

Resolved in 40f0054799f786e113ec489e03167da8c929ad59. In both locations named in the finding, the code now checks that the `maker` and `maker_stats` accounts have the same authority.

TOB-DRIFT-9: Panics used for error handling

Resolved in f63b160024afb4001973a6a3ba80d2db149434e8. A `SafeUnwrap` trait was added to, e.g., convert `Options` into `Results`. Calls to `safe_unwrap` are not used where panics were used before. We verified that Clippy's `expect_used`, `unwrap_used`, and `panic` lints produce no warnings when applied to the `drift` crate.

While we consider the issue resolved, we recommend against implementing `SafeUnwrap` for `Result` (figure E.3). If `safe_unwrap` were accidentally called on a `Result` where no call

was needed, the call would effectively hide the original error by turning it into `ErrorCode::FailedUnwrap`.

```
impl<T, U> SafeUnwrap for Result<T, U> {
    type Item = T;

    #[track_caller]
    #[inline(always)]
    fn safe_unwrap(self) -> DriftResult<T> {
        match self {
            Ok(v) => Ok(v),
            Err(_) => {
                let caller = Location::caller();
                msg!("Unwrap error thrown at {}:{}", caller.file(), caller.line());
                Err(ErrorCode::FailedUnwrap)
            }
        }
    }
}
```

Figure E.3: programs/drift/src/math/safe_unwrap.rs#L28-L43

Instead, we recommend using `Result::map_err` wherever such a conversion is necessary. Thus, `safe_unwrap` would be used only for `Options`, and `map_err` would be used for `Results`. Adopting this strategy would make it impossible to accidentally call `safe_unwrap` on a `Result` where no call was needed.

Also, we noticed that a Git hook was added to run Clippy before each commit (figure E.4). However, the commands in the hook fail when run. In particular, running Clippy with `-D warnings` fails. This suggests that either the hook is not being set correctly, or the failure is not being caught.

```
cargo +stable clippy -p drift -- -D warnings -D clippy::unwrap_used -D
clippy::expect_used -D clippy::panic
```

Figure E.4: .husky/pre-commit#L5

TOB-DRIFT-10: Testing code used in production
Partially resolved in 39f7292a2084cd0d275d5671dc58aa2e648787d8. The implementation of the `default_test` function in figure 10.2 is now guarded by `#[cfg(test)]`. Other functions are now guarded by `#[cfg(test)]` as well.

However, as noted in TOB-DRIFT-10, the `quote_asset_price_data` field (figure 10.1) is used in production, contrary to our initial understanding. This raises concerns because there is currently no way to set the contents of this field, and no asset's price is perfectly constant. Commit 39f7292a2084cd0d275d5671dc58aa2e648787d8 does not introduce a way to set the field. Hence, we consider this issue only partially resolved.

TOB-DRIFT-11: Inconsistent use of checked arithmetic

Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-12: Inconsistent and incomplete exchange status checks

Resolved in 7f7a04d5dc87962f3fa511139c06e699d312c738. The exchange's status is now represented using a `u8`. Individual bits are set using an `enum` generated by the `enumflags2` crate. Furthermore, some consistency was imposed between statuses and pausable operations. For example, `handle_transfer_deposit` now checks whether the `DepositPaused` flag is set.

The flags are still checked in multiple ways. For example, `handle_resolve_perp_bankruptcy` (figure E.5) and `handle_update_spot_market_cumulative_interest` (figure E.6). However, in the former case, the check within the body appears redundant, and thus could be removed. In the latter case, it appears the check could not be moved into an `access_control` annotation, because an action is performed even when the exchange is paused.

```
#[access_control(
    perp_market_valid(&ctx.accounts.perp_market)
    funding_not_paused(&ctx.accounts.state)
    valid_oracle_for_perp_market(&ctx.accounts.oracle, &ctx.accounts.perp_market)
)]
pub fn handle_update_funding_rate(
    ctx: Context<UpdateFundingRate>,
    perp_market_index: u16,
) -> Result<()> {
    ...
    let is_updated = controller::funding::update_funding_rate(
        perp_market_index,
        perp_market,
        &mut oracle_map,
        now,
        &state.oracle_guard_rails,
        state.funding_paused()?,
        None,
    )?;

    if !is_updated {
        return Err(ErrorCode::InvalidFundingProfitability.into());
    }

    Ok(())
}
```

Figure E.5: programs/drift/src/instructions/keeper.rs#L1027-L1078

```
#[access_control(
    spot_market_valid(&ctx.accounts.spot_market)
    exchange_not_paused(&ctx.accounts.state)
```

```
    valid_oracle_for_spot_market(&ctx.accounts.oracle, &ctx.accounts.spot_market)
)]
pub fn handle_update_spot_market_cumulative_interest(
    ctx: Context<UpdateSpotMarketCumulativeInterest>,
) -> Result<()> {
    ...
    if !state.funding_paused()? {
        controller::spot_balance::update_spot_market_cumulative_interest(
            spot_market,
            Some(oracle_price_data),
            now,
        )?;
    } else {
        // even if funding is paused still update twap stats
        controller::spot_balance::update_spot_market_twap_stats(
            spot_market,
            Some(oracle_price_data),
            now,
        )?;
    }

    Ok(())
}
```

Figure E.6: programs/drift/src/instructions/keeper.rs#L1150-L1188

Note that the original finding was about the exchange's status, and we therefore consider it resolved. However, we recommend incorporating `enumflags2` into the representation of the markets' statuses as well (see figure E.7).

```
pub enum MarketStatus {
    Initialized,     // warm up period for initialization, fills are paused
    Active,          // all operations allowed
    FundingPaused,   // perp: pause funding rate updates | spot: pause interest
updates
    AmmPaused,       // amm fills are prevented/blocked
    FillPaused,      // fills are blocked
    WithdrawPaused, // perp: pause settling positive pnl | spot: pause withdrawing
asset
    ReduceOnly,      // fills only able to reduce liability
    Settlement, // market has determined settlement price and positions are expired
must be settled
    Delisted,   // market has no remaining participants
}
```

Figure E.7: programs/drift/src/state/perp_market.rs#L32-L42

TOB-DRIFT-13: Spot market access controls are incomplete

Partially resolved in cb0cb29049fe9feae18cf62bfdf060e53bcc3422. `access_control` annotations were added to most functions involving spot markets. Functions involving perpetual and spot markets now largely coincide in terms of the exchange statuses they

check. However, there still appear to be some discrepancies. In particular, the example from figures 13.3 and 13.4 (repeated in figures E.8 and E.9) still applies to the updated code.

```
#[access_control(
    exchange_not_paused(&ctx.accounts.state)
)]
pub fn handle_place_perp_order(ctx: Context<PlaceOrder>, params: OrderParams) ->
Result<()> {
```

Figure E.8: programs/drift/src/instructions/user.rs#L688-L691

```

pub fn handle_place_spot_order(ctx: Context<PlaceOrder>, params: OrderParams) ->
Result<()> {
```

Figure E.9: programs/drift/src/instructions/user.rs#L1023-L1024

TOB-DRIFT-14: Oracles can be invalid in at most one way
Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-15: Code duplication
Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-16: Inconsistent use of integer types
Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-17: Use of opaque constants in tests
Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-18: Accounts from contexts are not always used by the instruction
Unresolved. Drift Protocol has not resolved this issue.

TOB-DRIFT-19: Unaligned references are allowed
Resolved in 8d0b518b765879cb6fbff8eced30bdaa397006ce. The `#![allow(unaligned_references)]` crate level attribute was removed.

TOB-DRIFT-20: Size of created accounts derived from in-memory representation
Resolved in 5f25f2a2d5548d9e82933e3d907fd01354e04a7a (which also resolves TOB-DRIFT-5 above). A `Size` trait was added that allows a type to specify its storage size in a `SIZE` constant. Tests are used to verify that the values of types' `SIZE` constants match the sizes of the types' in-memory representations. These `SIZE` constants are now used where expressions involving `std::mem::size_of` were used before.