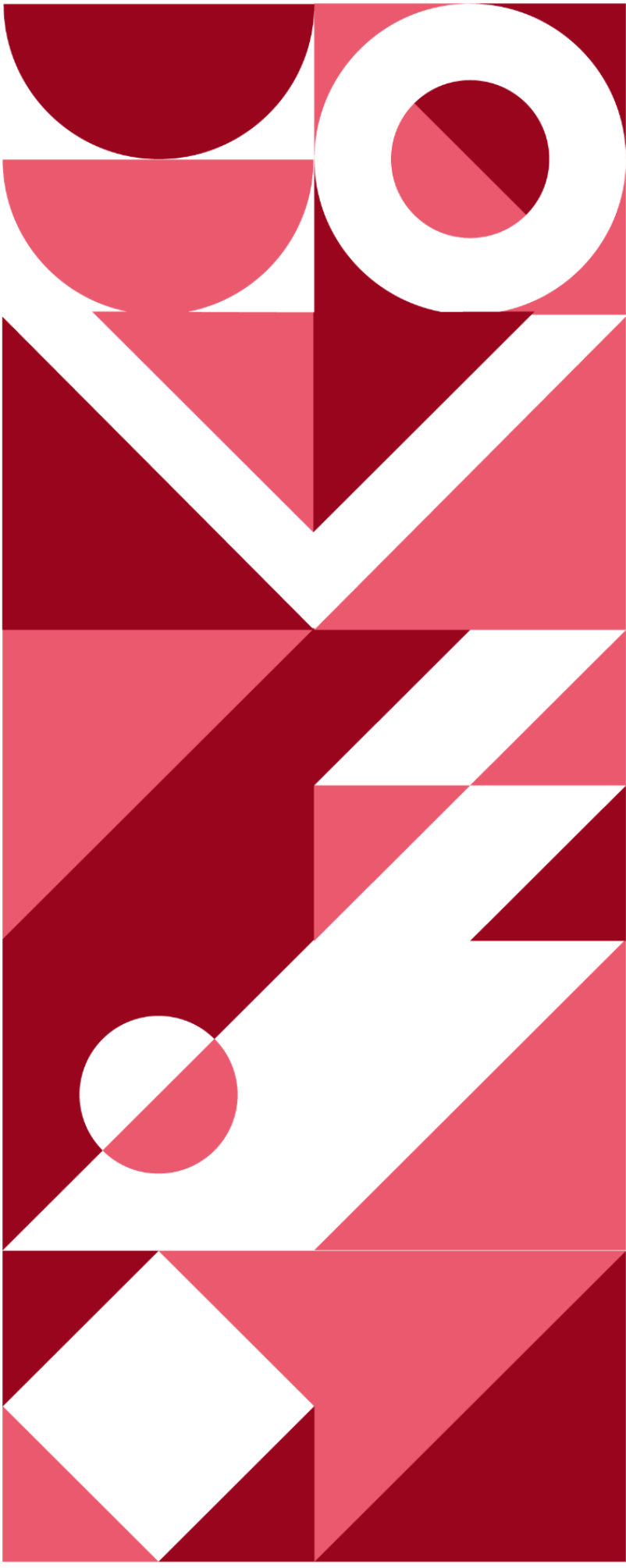


Pyth Oracle

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

William Wang defund@osec.io

Table of Contents

01 | Executive Summary

Overview

Pyth Data Association engaged OtterSec to perform an assessment of the `pyth-oracle` program. This assessment was conducted between April 25th and May 20th, with a focus on the following:

- Ensuring the integrity of the programs at an implementation and design level.
- Analyzing the program attack surface and providing meaningful recommendations to mitigate future vulnerabilities.

All issues were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the Pyth Data Association team to streamline patches and confirm remediation.

Key Findings

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

- 7 findings total, 2 of which we classify as vulnerabilities

We also observed the following:

- The codebase was well-structured and easy to understand.
- The team was very knowledgeable and responsive to our feedback.
- Updating time-weighted metrics and the associated PD arithmetic is relatively fragile.

As a part of this audit, we implemented proofs of concept for each vulnerability. They are available at https://osec.io/pocs/pyth-oracle, and a full list can be found in Appendix C.

02 | Scope

We received the program and began the audit April 25th. The source code was delivered to us in a git repository at https://github.com/pyth-network/pyth-client. This audit was performed against commit 6aadd6f.

A brief description of the program is as follows. A full list of program files and hashes can be found in Appendix A.

Name	Description
oracle	Program which estimates prices for various products; they are used by other on-chain programs as ground truth. Aggregates data from multiple publishers for robustness.

03 | Procedure

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

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

On the other hand, auditing the implementation of the program requires a deep understanding of Solana's execution model. Some common implementation vulnerabilities include account ownership issues, arithmetic overflows, and rounding bugs. For a non-exhaustive list of security issues we check for, see Appendix B. One issue we observed was that providing larger-than-required accounts bypassed the program's rent exemption checks (OS-PYO-ADV-01).

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

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

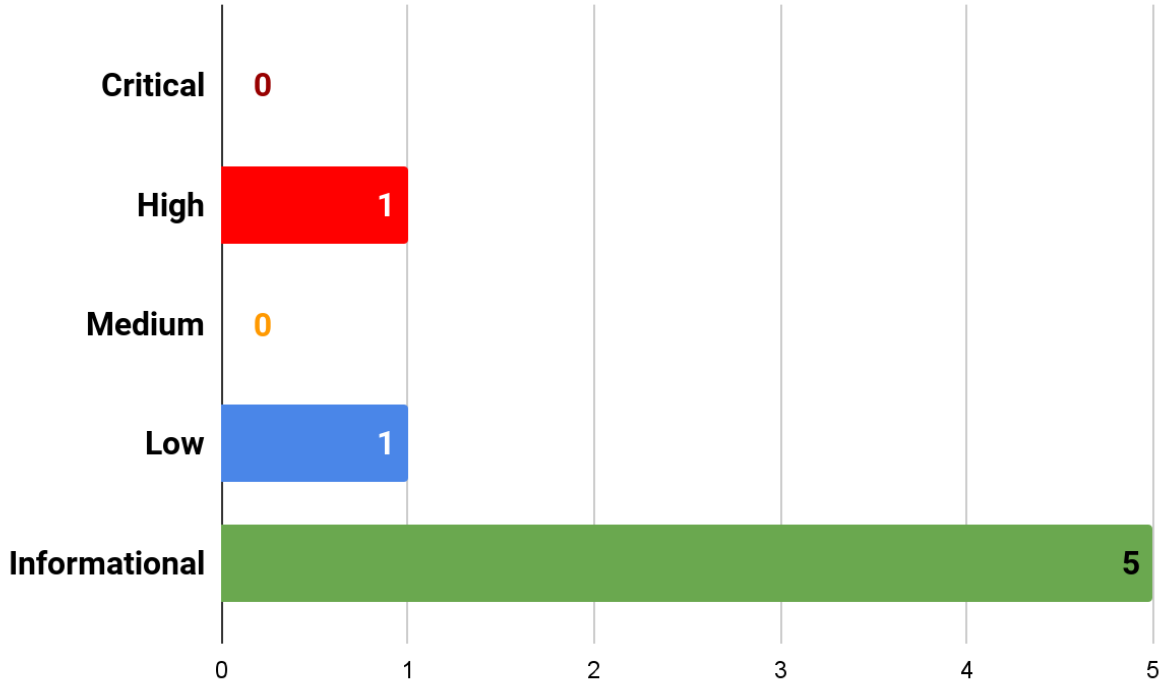
While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.

04 | Findings

In total, we report 7 findings.

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

The following chart displays our findings by severity.



Proofs of Concept

For each vulnerability, we implemented proofs of concept to demonstrate exploitability. They are available at https://osec.io/pocs/pyth-oracle and a full list can be found in Appendix C. In this audit, we wrote proofs of concept by modifying the program's existing test framework. They are stored as patch files, meant to be applied at the target commit.

Proofs of concept also function as an effective regression test. We recommend integrating them as part of a comprehensive test suite.

To run a proof of concept, use the provided script:

```
./run.sh <directory name>
```

For example, to run OS-PYO-ADV-00:

```
./run.sh os-pyo-adv-00
```

05 | Vulnerabilities

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

Rating criterion can be found in Appendix E.

ID	Severity	Status	Description
OS-PYO-ADV-00	**High**	**Resolved**	Loss of precision in PD storage leads to incorrect time-weighted metrics
OS-PYO-ADV-01	**Medium**	**Resolved**	Rent exemption checks are incorrectly performed on the minimum buffer size

OS-PYO-ADV-00 [High] [Resolved]: Incorrect time-weighted metrics

Description

In certain situations, the listed time-weighted average price (TWAP) and confidence vary wildly from the theoretical values. The issue stems from the `pd_store` function, which attempts to coerce a floating point's exponent to be between -16 and 15. When the exponent is too small, the value is repeatedly divided by 10, resulting in loss of precision.

pd.h:L47-L50

```
while ( e < -( 1 << ( EXP_BITS - 1 ) ) ) {
  v /= 10;
  ++e;
}
```

This is only a problem if `pd_store` is called with extremely small quantities. However, that may be the case for the TWAP formula's denominator, which is effectively a sum of inverse confidences. This is further exacerbated by the `upd_twap` function scaling quantities by the underlying `pc_price_t` exponent.

upd_aggregate.h:L138-L139

```
pd_new_scale( px, ptr->agg_.price_, ptr->expo_ );
pd_new_scale( conf, ( int64_t )( ptr->agg_.conf_ ), ptr->expo_ );
```

Thus for large `ptr->agg_.conf_` or `ptr->expo_`, the resulting `conf` can easily trigger loss of precision.

Proof of Concept

Suppose a price struct is initialized with exponent 16. Consider the following updates to the time-weighted average price, i.e. calls to the `upd_twap` function.

- Update with aggregate price 1000 and confidence 1. The numerator is computed to be 100000000e-5 and the denominator is computed to be 10e-17. They are stored as 100000000e-5 and 1e-16 respectively.
- Update with aggregate price 1000 and confidence 1. The numerator is computed to be 199988293e-5, and the denominator is computed to be 199988293e-24. They are stored as 199988293e-5 and 1e-16 respectively. Notice the denominator's loss of precision.

- Update with aggregate price 1000 and confidence 1. The numerator is computed to be 29996488e-4 and the denominator is computed to be 199988293e-24. The time-weighted average price is computed to be 1499.

To conclude, the time-weighted average price is listed as 1499 even though all aggregate prices were 1000.

Remediation

Notice that scaling price and confidence by the `pc_price_t` exponent is unnecessary. To maintain a healthy level of precision, the `pd_t` quantities should be initialized with zero as the exponent.

```
pd_new_scale( px, ptr->agg_.price_, 0 );
pd_new_scale( conf, ( int64_t )( ptr->agg_.conf_ ), 0 );
```

Note that the time-weighted metrics will now be recovered by scaling back to zero, instead of `qs->expo_` as the exponent.

upd_aggregate.h:L125-L126

```
pd_adjust( val, qs->expo_, qs->fact_ );
ptr->val_   = val->v_;
```

Even when the exponent is zero, however, a confidence of 1e16 will still result in incorrect time-weighted metrics. A simple fix would be to uniformly scale `cwgt` by a constant, e.g. 10e8.

upd_aggregate.h:L89-L94

```
pd_new( one, 100000000L, -8 );
if ( conf->v_ ) {
  pd_div( cwgt, one, conf );
} else {
  pd_set( cwgt, one );
}
```

Patch

Pyth Data Association acknowledges the finding and developed a patch for this issue: #179.

OS-PYO-ADV-01 [Low] [Resolved]: Incorrect rent exemption checks

Description

The program requires that all writable accounts be rent-exempt; this is enforced within the `valid_signable_account` and `valid_writable` functions.

oracle.c:L38-L57

```
static bool valid_signable_account( SolParameters *prm,
                                    SolAccountInfo *ka,
                                    uint64_t dlen )
{
  return ka->is_signer &&
         ka->is_writable &&
         SolPubkey_same( ka->owner, prm->program_id ) &&
         ka->data_len >= dlen &&
         is_rent_exempt( *ka->lamports, dlen );
}

static bool valid_writable_account( SolParameters *prm,
                                    SolAccountInfo *ka,
                                    uint64_t dlen )
{
  return ka->is_writable &&
         SolPubkey_same( ka->owner, prm->program_id ) &&
         ka->data_len >= dlen &&
         is_rent_exempt( *ka->lamports, dlen );
}
```

However, the `is_rent_exempt` calculation is performed with the minimum required length `dlen`, rather than the account's actual length `ka->data_len`. This allows an attacker to create a larger-than-required account which is not rent-exempt.

Proof of Concept

Consider the following scenario:

- An attacker attempts to initialize a product account with a 1024-char buffer and 4454400 lamports. Note that `PC_PROD_ACC_SIZE` is 512.
- The program does not reject the instruction, since the lamport balance is sufficient for a 512-char buffer.

To conclude, the attacker was able to use a larger-than-required account to bypass the rent exemption check.

Remediation

The rent exemption checks in valid_signable_account and valid_writable account should be replaced with the following.

```
is_rent_exempt( *ka->lamports, ka->data_len );
```

Patch

Pyth Data Association acknowledges the finding and developed a patch for this issue: #168

06 | General Findings

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

ID	Description
OS-PYO-SUG-00	Unused quote-set data
OS-PYO-SUG-01	Test instructions remain in production
OS-PYO-SUG-02	Test instructions do not check exponent
OS-PYO-SUG-03	Potential out-of-bounds read in PD arithmetic
OS-PYO-SUG-04	Potential integer overflows in PD arithmetic

OS-PYO-SUG-00 [Resolved]: Unused quote-set data

Description

The quote-set data's `decay_` lookup table is unused in the codebase, besides being initialized in the `qset_new` function.

upd_aggregate.h:L14-L23

```
typedef struct pc_qset
{
  pd_t      iprice_[PC_COMP_SIZE];
  pd_t      uprice_[PC_COMP_SIZE];
  pd_t      lprice_[PC_COMP_SIZE];
  pd_t      weight_[PC_COMP_SIZE];
  int64_t   decay_[1+PC_MAX_SEND_LATENCY];
  int64_t   fact_[PC_FACTOR_SIZE];
  int32_t   expo_;
} pc_qset_t;
```

Remediation

The `decay_` field should be removed entirely.

Patch

Pyth Data Association acknowledges the finding, but doesn't believe it has security implications. However, they may deploy a fix to address it.

OS-PYO-SUG-01 [Resolved]: Test instructions remain in production

Description

The `init_test` and `upd_test` instructions are intended to be used in unit tests. However, they are also compiled in production, which is unnecessary.

Remediation

The `init_test` and `upd_test` functions, as well as their switch cases in the `dispatch` function, should be removed when compiling the on-chain program. This can be done with preprocessor directives.

Patch

Pyth Data Association acknowledges the finding and developed a patch for this issue: #182.

OS-PYO-SUG-02 [Resolved]: Test instructions do not check exponent

Description

The `init_price` instruction performs bounds checks on the exponent `cptr->expo_`.

oracle.c:L270-L275

```
cmd_init_price_t *cptr = (cmd_init_price_t*)prm->data;
if ( prm->data_len != sizeof( cmd_init_price_t ) ||
     cptr->expo_ > PC_MAX_NUM_DECIMALS ||
     cptr->expo_ < -PC_MAX_NUM_DECIMALS ) {
  return ERROR_INVALID_ARGUMENT;
}
```

However, there is no such check for the exponent `cmd->expo_` in the `upd_test` instruction.

oracle.c:L478-L486

```
cmd_upd_test_t *cmd = (cmd_upd_test_t*)prm->data;
pc_price_t *px = (pc_price_t*)ka[1].data;
if ( prm->data_len != sizeof( cmd_upd_test_t ) ||
     px->magic_ != PC_MAGIC ||
     px->ver_ != cmd->ver_ ||
     px->type_ != PC_ACCTYPE_TEST ||
     cmd->num_ > PC_COMP_SIZE ) {
  return ERROR_INVALID_ARGUMENT;
}
```

Remediation

The `upd_test` instruction should perform bounds checks on the provided exponent.

Patch

Pyth Data Association acknowledges the finding and developed a patch for this issue: #182.

OS-PYO-SUG-03 [Resolved]: Potential out-of-bounds read in PD arithmetic

Description

The `pd_adjust` function uses a lookup table to multiply the value `v` by a power of ten. However, if the original exponent `n->e_` and target exponent `e` differ too much, the index `d` will exceed the table's length. This constitutes an out-of-bounds read.

pd.h:L68-L79

```
static inline void pd_adjust( pd_t *n, int e, const int64_t *p )
{
  int64_t v = n->v_;
  int d = n->e_ - e;
  if ( d > 0 ) {
    v *= p[ d ];
  }
  else if ( d < 0 ) {
    v /= p[ -d ];
  }
  pd_new( n, v, e );
}
```

Remediation

The `pd_adjust` function should perform bounds checks on `d` and return a bool to signify whether the adjustment succeeded, similar to the `pd_store` function.

pd.h:L35-L60

```
static inline bool pd_store( int64_t *r, pd_t const *n )
{
  ...
  while ( e > ( 1 << ( EXP_BITS - 1 ) ) - 1 ) {
    v *= 10;
    if ( v < -( 1L << 58 ) || v > ( 1L << 58 ) - 1 ) {
      return false;
    }
    --e;
  }
  *r = ( v << EXP_BITS ) | ( e & EXP_MASK );
  return true;
}
```

Patch

Pyth Data Association acknowledges the finding, but doesn't believe it has security implications. However, they may deploy a fix to address it.

OS-PYO-SUG-04 [Resolved]: Potential integer overflows in PD arithmetic

Description

Virtually all PD arithmetic functions do not account for integer overflow. In particular, we were able to hit an overflow in `pd_adjust` by leveraging OS-PYO-ADV-00.

The `pd_mul` function can overflow while multiplying the value fields `n1->v_` and `n2->v_`. The same is true while adding the exponent fields `n1->e_` and `n2->e_`, although this is effectively impossible to trigger.

oracle.c:L81-L86

```
static inline void pd_mul( pd_t *r, const pd_t *n1, const pd_t *n2 )
{
  r->v_ = n1->v_ * n2->v_;
  r->e_ = n1->e_ + n2->e_;
  pd_scale( r );
}
```

The `pd_adjust` function can overflow while multiplying the value field `v` and the table element `p[d]`.

pd.h:L68-L79

```
static inline void pd_adjust( pd_t *n, int e, const int64_t *p )
{
  int64_t v = n->v_;
  int d = n->e_ - e;
  if ( d > 0 ) {
    v *= p[ d ];
  }
  else if ( d < 0 ) {
    v /= p[ -d ];
  }
  pd_new( n, v, e );
}
```

Finally, both the `pd_add` and `pd_sub` functions can overflow for similar reasons.

Remediation

PD arithmetic functions should perform bounds checks on input arguments, and return a bool to signify invalid values.

Patch

Pyth Data Association acknowledges the finding, but doesn't believe it has security implications. However, they may deploy a fix to address it.

07 | Appendix

Appendix A: Program Files

Listed below are the files in scope for this audit and their truncated sha256 hashes.

```
oracle.c                                3a267a63dc795bbf687e9b8d544af8b5
oracle.h                                4640929eef92aee32b2523cec2a8f84f
pd.h                                    062d7ec298a0a16f2610db15acfa3330
test_oracle.c                           2628f6e50a4342e15bdf1bd7be54f026
upd_aggregate.h                         ac9b7eab461b154508a491ca7a22ffd2
model
  clean                                 065d7eb9c162884a6fc90aede647ba1d
  price_model.c                         f6d7444a1c1a67ffe6f83b1a63cd99fc
  price_model.h                         c803f1d625623b34982f5b6457fdbdf4
  run_tests                             6f7dd4e7ac1c4af096fc049b2acd05c9
  test_price_model.c                    547eebd7b6e625b22dec0a13e5203513
sort
  clean                                 065d7eb9c162884a6fc90aede647ba1d
  run_tests                             25230ed03cb8c6be7a53dc8118e0f9fb
  sort_stable_base_gen.c                97df993dc7a374200bbb077ba730cfb8
  test_sort_stable.c                    e551aed1de3ee263f0549318a61ac33f
  tmpl
    sort_stable.c                       b2e2a51cdde4e55dfd6c0f99d2bbcff7
    sort_stable_base.c                  8174f67112100b6451333ad0f493160a
util
  align.h                               a9cf8c3e189762ce5ef5835c17be7ddb
  avg.h                                 b93e0d4bf199109e8076ff81d2dec8ca
  clean                                 065d7eb9c162884a6fc90aede647ba1d
  compat_stdint.h                       1b5134a2e9a960d054b3268f2a246747
  hash.h                                be81d745243fe7b31b23b8b5dd7256e8
  prng.h                                9cc93381ec43bd87a3edd5ca636abf2b
  run_tests                             592f86835a9a0c0e4ca20dd7747d7059
  sar.h                                 9eb74064fd87ecdcfb9388aa71283af4
  test_align.c                          8a642c4c7542f73f109123bd26a5163f
  test_avg.c                            30bbef1a8f3423fa035063770ac3ca41
  test_hash.c                           b514fe6e774b02d2a8fc3aa617ab89d6
  test_prng.c                           7dd1f97e63678008f3b6e6edded586dd
  test_prng_battery.c                   875a280eed3c49b5a1dba077a2cd68e2
  test_round.c                          9196c2400673e9f697a16c7b26e819ee
  test_sar.c                            b8421f4eebdc961f726ea0907a14cd68
  util.h                                55a8e4a0a4bb8a69928521f8775f37da
```

Appendix B: Implementation Security Checklist

Unsafe arithmetic

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

Account security

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

Input validation

Timestamps	Timestamp inputs should be properly validated against the current clock time. Timestamps which are meant to be in the future should be explicitly validated so.
Numbers	Sane limits should be put on numerical input data to mitigate the risk of unexpected over and underflows. Input data should be constrained to the smallest size type possible, and upcasted for unchecked arithmetic.
Strings	Strings should have reasonable size restrictions to prevent denial of service conditions.

Internal State	If there is internal state, ensure that there is explicit validation on the input account's state before engaging in any state transitions. For example, only open accounts should be eligible for closing.

Miscellaneous

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

Appendix C: Proofs of Concept

Listed below are the provided proof of concept files and their truncated sha256 hashes.

```
Dockerfile                                11a2e4daf5d16f3c4f945b51af31c89b
README.md                                 42ac276ee79ba2e01bfc4b526b76612d
run.sh                                    2ef3900684944342e7687c54779274ea
config
  run.sh                                  4e14c4214f777fd1b6764735c720b0c4
pocs
  os-pyo-adv-00
    patch                                 e9c3da23dc3a45866a0ab3c95b0147d1
  os-pyo-adv-01
    patch                                 4e1c5b37bf78e211dcf1a8e6ad45150d
```

Appendix D: Vulnerability Rating Scale

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

Critical	Vulnerabilities which **immediately** lead to loss of user funds with minimal preconditions. • Misconfigured authority/token account validation • Rounding errors on token transfers
High	Vulnerabilities which **could** lead to loss of user funds but are potentially difficult to exploit. • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities which could lead to denial of service scenarios or degraded usability. • Computation limit exhaustion due to malicious input • Forced exceptions preventing normal use
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as *general findings*. • Explicit assertion of critical internal invariants • Improved input validation • Uncaught Rust errors (vector out of bounds indexing)