Bramah Systems

dYdX Public Report

PROJECT: dYdX/solo

February 2019

Prepared For:

Brendan Chou | dYdX Trading, Inc.

brendan@dydx.exchange

Prepared By:

Jonathan Haas | Bramah Systems, LLC.

jonathan@bramah.systems

Table of Contents

Executive Summary **3**
- Scope of Engagement 3
- Timeline 3
- Engagement Goals 3
- Protocol Specification 3
- Overall Assessment 4
- Timeliness of Content 5

General Recommendations **6**
- Usage of experimental Solidity version 6
- nonReentrant functions not marked External 6
- Choose “Base Currency” for consistency 6
- Non-initialized Return Values 6
- Completion of TODO’s & Incomplete Functionality 7
- Usage of Block.timestamp 7
- External call failure could be more verbose 7
- Highly Privileged Owner Accounts 7
- NPM Module Usage 8

Toolset Warnings **9**
- Extensive Test Coverage 9
- Excess Gas Consumption & Costly Loops 9
- Parsing Functions & Malformed Input 10
- ERC-20 Race Condition 10

Directory Structure **11**

Appendix **14**
- Mythril Detection Capabilities 14
- Oyente Detection Capabilities 16

dYdX “Solo” Assessment

Executive Summary

Scope of Engagement

Bramah Systems, LLC was engaged in late January of 2019 to perform a comprehensive security review of the dYdX Trading, Inc. “Solo” repository protocol. A review was conducted over the period by a member of the Bramah Systems, LLC. executive staff. During this period, all Solidity smart contract code (*.sol) as of commit **77917ea86771ea3b2deec2f9c21ac0b587148b60** was included within scope, along with TypeScript files (*.ts) relevant to testing. Bramah Systems completed the assessment using manual, static and dynamic analysis techniques.

Timeline

Audit Commencement: February 11, 2019

Report Delivery: February 18, 2019

Engagement Goals

The primary scope of the engagement was to evaluate and establish the overall security of the Solo system, with a specific focus on trading actions. In specific, the engagement sought to answer the following questions:

- Is it possible for an attacker to steal or freeze tokens?
- Is there a way to interfere with the trading mechanisms?
- Are the arithmetic calculations trustworthy?

Protocol Specification

While minimal specification is proposed within the README, multiple checks and constants are presented through the Truffle project to ensure intended actions are successfully performed.

Overall Assessment

Bramah Systems was engaged to evaluate and identify significant security concerns in the codebase of the dYdX protocol architecture. During the course of our engagement, Bramah Systems noted numerous instances wherein the protocol deviated from established best practices and procedures of secure software development. **With limited exceptions (as described below), these instances were a result of structural limitations of Solidity and not due to inactions on behalf of the development team.**

Overall, the code reviewed is of excellent quality, written with clear awareness of current smart contract development best practices, common security pitfalls, and overall readability. Its interfaces are well designed and its use of patterns display strong code maturity.

In particular, Bramah Systems notes that the code is well commented, particularly in sections where understanding the developer's intent is essential. Additionally, the overall contract organization is consistent throughout (within contracts themselves and their overarching interactions with others).

While during the course of the review Bramah Systems discovered areas worthy of attention by the dYdX team, these issues have since been addressed and no significant security concerns remain. We applaud the dYdX team on their immense dedication in following security best practices throughout the course of development of their protocol.

Disclaimer

As of the date of publication, the information provided in this report reflects the presently held, commercially reasonable understanding of Bramah Systems, LLC.'s knowledge of security patterns as they relate to the dYdX Protocol, with the understanding that distributed ledger technologies ("DLT") remain under frequent and continual development, and resultantly carry with them unknown technical risks and flaws. The scope of the review provided herein is limited solely to items denoted within "Scope of Engagement" and contained within "Directory Structure". The report does NOT cover, review, or opine upon security considerations unique to the Solidity compiler, tools used in the development of the protocol, or distributed ledger technologies themselves, or to any other matters not specifically covered in this report.
The contents of this report must NOT be construed as investment advice or advice of any other kind. This report does NOT have any bearing upon the potential economics of the dYdX protocol or any other relevant product, service or asset of dYdX or otherwise. This report is not and should not be relied upon by dYdX or any reader of this report as any form of financial, tax, legal, regulatory, or other advice.

To the full extent permissible by applicable law, Bramah Systems, LLC. disclaims all warranties, express or implied. The information in this report is provided "as is" without warranty,

representation, or guarantee of any kind, including the accuracy of the information provided.

Bramah Systems, LLC. makes no warranties, representations, or guarantees about the dYdX Protocol. Use of this report and/or any of the information provided herein is at the users sole risk, and Bramah Systems, LLC. hereby disclaims, and each user of this report hereby waives, releases, and holds Bramah Systems, LLC. harmless from, any and all liability, damage, expense, or harm (actual, threatened, or claimed) from such use.

Timeliness of Content

All content within this report is presented only as of the date published or indicated, to the commercially reasonable knowledge of Bramah Systems, LLC. as of such date, and may be superseded by subsequent events or for other reasons. The content contained within this report is subject to change without notice. Bramah Systems, LLC. does not guarantee or warrant the accuracy or timeliness of any of the content contained within this report, whether accessed through digital means or otherwise.

Bramah Systems, LLC. is not responsible for setting individual browser cache settings nor can it ensure any parties beyond those individuals directly listed within this report are receiving the most recent content as reasonably understood by Bramah Systems, LLC. as of the date this report is provided to such individuals.

General Recommendations

Best Practices & Solidity Development Guidelines

Usage of experimental Solidity version

A majority of the contracts associated with the protocol make usage of an experimental Solidity version (**pragma experimental ABIEncoderV2**) which enables usage of the new ABI encoder. **ABIEncoderV2** allows for the usage of structs and arbitrarily nested arrays (such as **string[]** and **uint256[][]**) in function arguments and return values.

As no present non experimental version for these constructs exists, one must acknowledge the associated risk in utilizing non release-candidate (“RC) software. It is understood that software in the beta phase will generally have more bugs than completed software as well as speed/performance issues and may cause crashes or data loss.

File: Numerous

nonReentrant functions not marked External

The **nonReentrant** modifier prevents a contract from calling itself, directly or indirectly. If you mark a function `**nonReentrant**`, you should also mark it `**external**`. Calling one `**nonReentrant**` function from another is not supported. Instead, you can implement a `**private**` function doing the actual work, and an `**external**` wrapper marked as `**nonReentrant**`.

File: Numerous

Choose “Base Currency” for consistency

Multiple digital currencies (and their respective components) are referenced throughout the project. In order to maintain consistency, a core currency should be chosen (such as Ether or Dai). As various stable-coins the protocol seeks to accept may differ in their relative price to the dollar (USD), pegging these entities to their relative price in Ether will allow for the most comprehensive price tracking.

File: Numerous

Non-initialized Return Values

Within **protocol/lib/Storage.sol** in the **SetStatus** function, the function's signature only denotes

the type of the return value, but the function's body does not contain return statement. If the return value is not needed, the specification of the return type is not inherently necessary.

File: protocol/lib/Storage.sol
Lines: 531-540

Completion of TODO's & Incomplete Functionality

Throughout the project, there are multiple instances in which TODO is referenced. In each, establish whether or not the goal of the file has been established (e.g. in **contracts\protocol\State.sol** it appears the state implementation is relatively feature complete). In particular, pricing oracles appear to be heavily referenced throughout the code but their actual implementation does not appear to be present at the time of review.

File: Numerous

Usage of Block.timestamp

Miners can affect block.timestamp for their benefits. Thus, one should not rely on the exact value of block.timestamp. As a result of such, **block.timestamp** and **now** should traditionally only be used within inequalities (note: the protocol **does** follow this strategy).

Block numbers and average block time can be used to estimate time, but this is not future proof as block times may change (such as the changes expected during Casper).

File: protocol/lib/Time.sol
Lines: 46-46

External call failure could be more verbose

There is a non-zero possibility that for numerous reasons an external contract call could fail. Especially when sending Ether, it is critical to check for relevant return values and ensure error handling. Without verbose logging of the potential error (and only the arguments, as is currently present), it may pose a large amount of difficulty to debug where potential errors may stem from.

File: protocol/impl/OperationImpl.sol
Lines: 811-815

Highly Privileged Owner Accounts

Within the Admin.sol file, multiple onlyOwner gated actions exist, allowing configuration

changes of multiple market and risk functions. If the exchange/proxy owner is hacked, and their Ethereum private key is exposed, catastrophic damage could be caused to the protocol. It is suggested that a two week time-delay be added to new administrative actors in the event a new owner is added, allowing individuals to withdraw their funds from the protocol in case an untrusted address is authorized for administrative actions.

No changes are recommended to the contract system here, but we wish to underscore the importance of the surrounding systems. Ensuring that these keys are protected is of the utmost criticality.

File: protocol/Admin.sol

NPM Module Usage

Throughout the project, NPM modules are utilized in order to import various functionality (notably, OpenZeppelin contracts). While this practice enables relatively minimal modifications to be made in order to invoke certain functions securely (such as with SafeMath), these libraries must be continuously updated in order to ensure they are used securely.

Virtually every non-blockchain application has these issues because most development teams do not focus on ensuring their components/libraries are up to date. In the case of blockchain codebases, however, knowing all outside components utilized is critical.

It is suggested the following steps are followed (as noted by the OWASP project):

1. Identify all components and the versions you are using, including all dependencies. (NPM package lock can help determine these).
2. Monitor the security of these components in public databases, project mailing lists, and security mailing lists, and keep them up to date.
3. Establish security policies governing component use, such as requiring certain software development practices, passing security tests, and acceptable licenses.
4. Where appropriate, consider adding security wrappers around components to disable unused functionality and/ or secure weak or vulnerable aspects of the component.

File: Numerous

Toolset Warnings

In addition to our manual review, our process involves utilizing concolic analysis and dynamic testing in order to perform additional verification of the presence security vulnerabilities. An additional part of this review phase consists of reviewing any automated unit testing frameworks that exist.
The following sections detail warnings generated by the automated tools and confirmation of false positives where applicable, in addition to findings generated through manual inspection.

Extensive Test Coverage

The contract repository heavily benefits from extensive unit test coverage throughout. This testing provides a variety of unit tests which encompass the various operational stages of the contract. The dYdX protocol (and its relevant components and their respective subcomponents) possesses numerous tests validating functionality and ensuring that certain behaviors (those relating to erroneous or overflow-prone input) do not see successful execution.

In particular, specific focus within the testing suite was placed upon validating that various actions (such as setting, getting, approval, and various market actions) cannot occur after a state change or as the result of bad input (such as an invalid address).

The dYdX team constructed tests in both TypeScript and native Solidity, allowing for a fairly robust test-suite.

Excess Gas Consumption & Costly Loops

If the state variables **.balance** or **.length** are used several times, holding its value in a local variable is more gas efficient (as the variable does not need to be accessed every loop iteration).

Moreover, as Ethereum miners impose a limit on the total number of gas consumed in a block, if **array.length** is large enough, the function will exceed the block gas limit, and transactions calling it will never be confirmed. As a result, if an external entity is to influence **array.length**, this could pose issue. Where possible, avoiding loops with a large number of iterations (or an unknown number of iterations) is advised.

In particular, code concerning accounts on various markets within **OperationImpl.sol** appears to be impacted by these constraints. With an immensely high number of markets (yet still within the bounds of **uint256**), various requests began to fail.

File: external/proxies/PayableProxyForSoloMargin.sol
Lines: 92-115
File: protocol/lib/Storage.sol
Lines: 499-511

File: protocol/lib/Storage.sol
Lines: 298-324
File: protocol/Permission.sol
Lines: 56-61
File: protocol/impl/OperationImpl.sol
Lines: 206-240
File: protocol/impl/OperationImpl.sol
Lines: 251-279
File: protocol/impl/OperationImpl.sol
Lines: 96-106
File: protocol/impl/OperationImpl.sol
Lines: 125-166
File: protocol/impl/OperationImpl.sol
Lines: 173-178

Parsing Functions & Malformed Input

Parsing functions within Actions.sol do not appear to handle malformed input (no reverting statement within enforces such behavior). While this may be anticipated, as these contract functions are not intended to be called directly, they could benefit from the inclusion of logic to alert on missing arguments. Seen similarly within Token.sol, a number of instances exist in which success must be assumed due to the lack of verbose failure. **In general, these limitations exist as underlying structural issues with Solidity (and the Ethereum platform as a whole), as logging for each potential failure would require an exorbitant amount of gas, but not logging presents potential risk. Overall, the dYdX team made careful and concise points of inclusion for each potential area of alert.**

File: protoco/lib/Actions.sol

ERC-20 Race Condition

A known race condition exists within the present implementation of the ERC20 standard. Due to the nature of this vulnerability being an inherent flaw in the ERC20 standard, considerations must be made for any divergence (as modifications made while no longer be ERC20 compliant).

With this noted, the dYdX protocol makes appropriate mitigations and utilizes the suggested allowance approach in order to remove concern presented by this vulnerability.

Directory Structure

At time of review, the directory structure of the dYdX “Solo” contract repository was as follows:

```
Directory of \User\Audit\Documents\GitHub\solo\contracts

02/14/2019  03:39 PM    <DIR>          external
02/14/2019  03:39 PM             1,221 Migrations.sol
02/14/2019  03:39 PM    <DIR>          protocol
02/14/2019  03:39 PM    <DIR>          testing
               1 File(s)          1,221 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\external

02/14/2019  03:39 PM    <DIR>          helpers
02/14/2019  03:39 PM    <DIR>          proxies
02/14/2019  03:39 PM    <DIR>          traders
               0 File(s)              0 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\external\helpers

02/14/2019  03:39 PM             1,488 OnlySolo.sol
               1 File(s)          1,488 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\external\proxies

02/14/2019  03:39 PM             3,724 PayableProxyForSoloMargin.sol
               1 File(s)          3,724 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\external\traders

02/14/2019  03:39 PM             6,840 Expiry.sol
               1 File(s)          6,840 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\protocol

02/14/2019  03:39 PM             4,867 Admin.sol
02/14/2019  03:39 PM             8,014 Getters.sol
02/14/2019  03:39 PM    <DIR>          impl
02/14/2019  03:39 PM    <DIR>          interfaces
02/14/2019  03:39 PM    <DIR>          lib
```

```
02/14/2019  03:39 PM             1,423 Operation.sol
02/14/2019  03:39 PM             1,552 Permission.sol
02/14/2019  03:39 PM             1,354 SoloMargin.sol
02/14/2019  03:39 PM               839 State.sol
               6 File(s)         18,049 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\protocol\impl

02/14/2019  03:39 PM             8,835 AdminImpl.sol
02/14/2019  03:39 PM            25,825 OperationImpl.sol
               2 File(s)         34,660 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\protocol\interfaces

02/14/2019  03:39 PM             1,309 IAutoTrader.sol
02/14/2019  03:39 PM             1,016 ICallee.sol
02/14/2019  03:39 PM             2,139 IErc20.sol
02/14/2019  03:39 PM             2,925 IExchangeWrapper.sol
02/14/2019  03:39 PM             1,242 IInterestSetter.sol
02/14/2019  03:39 PM             1,425 IPriceOracle.sol
               6 File(s)         10,056 bytes

Directory of \User\Audit\Documents\GitHub\solo\contracts\protocol\lib

02/14/2019  03:39 PM             1,410 Account.sol
02/14/2019  03:39 PM             9,481 Actions.sol
02/14/2019  03:39 PM             1,312 Decimal.sol
02/14/2019  03:39 PM            11,820 Events.sol
02/14/2019  03:39 PM             3,785 Exchange.sol
02/14/2019  03:39 PM             4,712 Interest.sol
02/14/2019  03:39 PM             2,829 Math.sol
02/14/2019  03:39 PM             1,797 Monetary.sol
02/14/2019  03:39 PM             6,191 Require.sol
02/14/2019  03:39 PM            17,199 Storage.sol
02/14/2019  03:39 PM             1,419 Time.sol
02/14/2019  03:39 PM             3,914 Token.sol
02/14/2019  03:39 PM             5,578 Types.sol
              13 File(s)         71,447 bytes
```

```
Directory of \User\Audit\Documents\GitHub\solo\contracts\testing

02/14/2019  03:39 PM             1,011 ErroringOmiseToken.sol
02/14/2019  03:39 PM             1,042 ErroringToken.sol
02/14/2019  03:39 PM             3,146 OmiseToken.sol
02/14/2019  03:39 PM             7,546 TestAutoTrader.sol
02/14/2019  03:39 PM             3,251 TestCallee.sol
02/14/2019  03:39 PM             1,592 TestInterestSetter.sol
02/14/2019  03:39 PM             1,372 TestPriceOracle.sol
02/14/2019  03:39 PM             3,112 TestSoloMargin.sol
02/14/2019  03:39 PM             3,459 TestToken.sol
02/14/2019  03:39 PM             1,035 TokenA.sol
02/14/2019  03:39 PM             1,035 TokenB.sol
02/14/2019  03:39 PM             1,035 TokenC.sol
              12 File(s)         28,636 bytes

    Total Files Listed:
              43 File(s)        176,121 bytes
              29 Dir(s)  142,041,116,672 bytes free
```

Appendix

Mythril Detection Capabilities

Issue	Description	Mythril Detection Module(s)	References
Unprotected functions	Critical functions such as sends with non-zero value or suicide() calls are callable by anyone, or msg.sender is compared against an address in storage that can be written to. E.g. Parity wallet bugs.	Unchecked_suicide, Ether_send unchecked_retval	
Missing check on CALL return value		unchecked_retval	Handle errors in external calls
Re-entrancy	Contract state should never be relied on if untrusted contracts are called. State changes after external calls should be avoided.	external calls to untrusted contracts	Call external functions lastAvoid state changes after external calls
Multiple sends in a single transaction	External calls can fail accidentally or deliberately. Avoid combining multiple send() calls in a single transaction.		Favor pull over push for external calls

External call to untrusted contract		external calls to untrusted contracts	
Delegatecall or callcode to untrusted contract		delegatecall_forward	
Integer overflow/underflow		integer	Validate arithmetic
Timestamp dependence		Dependence on predictable variables	Miner time manipulation
Payable transaction does not revert in case of failure			
Use of tx.origin		tx_origin	Solidity documentation, Avoid using tx.origin
Type confusion			
Predictable RNG		Dependence on predictable variables	
Transaction order dependence		Transaction order dependence	Front Running
Information exposure			
Complex fallback function (uses more than 2,300 gas)	A too complex fallback function will cause send() and transfer() from other contracts to fail. To implement this we first need to fully implement gas simulation.		

Use require()instead of assert()	Use assert() only to check against states which should be completely unreachable.	Exceptions	Solidity docs
Use of depreciated functions	Use revert()instead of throw(), selfdestruct() instead of suicide(), keccak256() instead of sha3()		
Detect tautologies	Detect comparisons that always evaluate to 'true', see also #54		
Call depth attack	Deprecated		

Oyente Detection Capabilities

Issue	Description
Re-entrancy	Contract state should never be relied on if untrusted contracts are called. State changes after external calls should be avoided.
Timestamp Dependence	The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. Block numbers and average block time can be used to estimate time, but this is not future proof as block times may change (such as the changes expected during Casper).
Assertion Failure	An assertion is a boolean expression at a specific point in a program which will be true unless there is a bug in the program.

	Assertion failures as such denote critical instances in which assumptions made by the developer no longer hold to be true.
Callstack Depth Attack	Deprecated
Transaction Order Dependence (TOD)	Since a transaction is in the mempool for a short while, one can know what actions will occur, before it is included in a block. This can be troublesome for things like decentralized markets, where a transaction to buy some tokens can be seen, and a market order implemented before the other transaction gets included.
Parity Multisig Bug 2	Unchecked kill/selfdestruct functions, such as those within the Parity Multisig Bug 2 can lead to destruction of the contract, sending funds to the given address provided.