

Primitive Hyper

Security assessment by HashEye · prepared for Primitive

HASHEYE AUDITED

PROJECT	Primitive Hyper
CLIENT	Primitive
CATEGORY	Blockchain
PUBLISHED	March 1, 2023
REPORT ID	research-primitive-hyper-2023-03-01-nquv70

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hashey.io/audits/research-primitive-hyper-2023-03-01-nquv70.

PrimitiveHyper SecurityAssessment March31,2023 Preparedfor: AlexAngel Primitive
Preparedby: NatChin, RobertSchneider, TarunBansal, andKurtWillis

About hashey Founded in 2012 and headquartered in New York, hashey 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/hashey-io/publications>, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, hashey 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. hashey 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 hashey on Twitter and explore our public repositories at <https://github.com/hashey-io>. To engage us directly, visit our "Contact" page at <https://www.hashey.io/contact>, or email us at info@hashey.io. hashey, Inc. 228 Park Ave S #80688 New York, NY 10003 <https://www.hashey.io> info@hashey.io hashey 1 Primitive Security Assessment PUBLIC

Notices and Remarks Copyright and Distribution ©2023 by hashey, Inc.

All rights reserved. hashey hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by hashey to be public information; it is licensed to Primitive under the terms of the project statement of work and has been made public at Primitive's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of hashey.

This sole canonical source for hashey publications is the hashey Publications page.

Reports accessed through any source other than that page may have been modified and

should not be considered authentic. Test Coverage Disclaimer

All activities undertaken by hashey 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.

hashey 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. hashey 2 Primitive Security Assessment PUBLIC

Table of Contents

- About hashey 1
- Notices and Remarks 2
- Table of Contents 3
- Executive Summary 5
- Project Summary 7
- Summary of Recommendations 7
- Project Goals 9
- Project Targets 10
- Project Coverage 11
- Automated Testing 13
- Codebase Maturity Evaluation 24
- Summary of Findings 27
- Detailed Findings 29
 - 1. Lack of zero-value checks on functions 29
 - 2. Documentation discrepancy in computePriceWithChangeInTau 30
 - 3. Risk of token theft due to possible integer underflow in `lt32` 32
 - 4. Risk of token theft due to unchecked type conversion 34
 - 5. Users can swap without paying any fees 36
 - 6. Swap function returns incorrectly scaled output token amount 38
 - 7. Liquidity providers can withdraw total fees earned by a pool 40
 - 8. Asset token price deviates from the price curve of the pool 42
 - 9. New pair creation can overwrite existing pairs 43
 - 10. Error in `Invariant.getX` 45
 - 11. Pools with overflowing maturity dates can be created 46
 - 12. Minting funds to the Hypercontract arbitrarily increases the next caller's balance 48
 - 13. Pool strike price could be zero due to lack of lower bound check on `maxTick` 49
 - 14. Rounding error allows liquidity to be added without depositing tokens 51
 - 15. Attacker scans and `wichChangeParameters` call steal funds 53

16. Limited due to strike prices due to fixed tick spacing 57
17. Functions that round by adding 1 result in unexpected behavior 59
18. Solidity compiler optimizations can be problematic 61
19. getAmountOut returns incorrect value when called by controller 62 hashey 3 Primitive Security Assessment PUBLIC

20. Mismatched base unit comparison can inflate limit tolerance 64
21. Incorrect implementation of fee cases in getY function 66
22. Lack of proper bound handling for solstat functions 68
23. Attackers can steal funds by swapping in both directions 71 A. Vulnerability Categories 73
B. Code Maturity Categories 75 C. Rounding Recommendations 77 D. Risks with Arbitrary Tokens and Third-Party Controllers 79 E. Recommendations for Overflow and Underflow Analysis in Assembly Blocks 80
F. Staking Issues 82 G. Code Quality Recommendations 84 hashey 4 Primitive Security Assessment PUBLIC

Executive Summary Engagement Overview

Primitive engaged hashey to review the security of its Hyper smart contracts. From January 3 to January 31, 2023, a team of four consultants conducted a security review of the client-provided source code, with eight person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report. 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 conducted this audit with full knowledge of the 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 uncovered significant flaws that could impact system confidentiality, integrity, or availability. A summary of the findings and detail on notable findings are provided below. EXPOSURE ANALYSIS Severity Count High 9 Medium 2 Low 4 Informational 6 Undetermined 2 CATEGORY BREAKDOWN Category Count Configuration 1 Data Validation 11 Undefined Behavior 11 hashey 5 Primitive Security Assessment PUBLIC

Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below. • Missing data validation throughout the codebase Throughout the codebase, many functions fail to check that incoming user arguments are bound between two values, resulting in the ambiguity of expected input parameters. For example, missing checks on type conversions could allow an attacker to steal funds (TOB-HYPR-4), and pools could be defined with a zero strike price (TOB-HYPR-13). • Incorrect handling of arithmetic The system is also affected by the mishandling of arithmetic operations, which could allow an attacker to steal funds. These issues stem from underlying assumptions that cause integer overflow issues (TOB-HYPR-3) and rounding issues (TOB-HYPR-13, TOB-HYPR-14, TOB-HYPR-16, TOB-HYPR-17, TOB-HYPR-20, TOB-HYPR-22, and TOB-HYPR-23). We recommend that the Primitive team continue to extend the existing Echidna suite to test for these corner cases in unexpected behavior. hashey 6 Primitive Security Assessment PUBLIC

Project Summary Contact Information The following managers were associated with this project:

Dan Guido, Account Manager Mary O'Brien, Project Manager dan@hashey.io mary.obrien@hashey.io The following engineers were associated with this project: Nat Chin, Consultant Robert Schneider, Consultant natalie.chin@hashey.io robert.schneider@hashey.io Tarun Bansal, Consultant Kurt Willis, Consultant tarun.bansal@hashey.io kurt.willis@hashey.io Project Timeline

The significant events and milestones of the project are listed below. Date Event January 3, 2023 Pre-project kickoff call January 9, 2023 Status update meeting #1 January 17, 2023 Status update meeting #2 January 24, 2023 Status update meeting #3 February 2, 2023 Delivery of report draft February 2, 2023 Report readout meeting March 31, 2023 Delivery of final report hashey 7 Primitive Security Assessment PUBLIC

Summary of Recommendations

hashey recommends that Primitive address the findings detailed in this report and take the following additional steps prior to deployment: • Perform additional economic analysis on the Hypercurve to ensure that input and output bounds are always known and checked. Add these bounds into the Hyper smart contracts to ensure that all functions behave as expected. • Identify additional global system invariants and continue to extend the Echidna end-to-end test suite to ensure that it captures all functions. Run and maintain this extended fuzzing campaign on a server with a corpus. • Document all implicit and explicit uses of rounding in the system. Integrate this documentation into a flowchart to create a visual representation outlining all of the formulas' rounding directions to ensure that they always favor pools over users. • Follow all of the recommendations related to rounding outlined in appendix C. •

Perform another security review on a release candidate prior to production deployment after all of the issues in this report have been addressed and fixed. hashey
8PrimitiveSecurityAssessment PUBLIC

Project Goals The engagement was scoped to provide a security assessment of Primitive's Hyper smart contracts. Specifically, we sought to answer the following non-exhaustive list of questions: • Can an attacker steal funds? • Are there appropriate access control measures in place for users and admins? • Does the system's behavior match the specification? • Can an attacker freeze funds or deny service to the system? • Is it possible to perform system operations without paying the required fees? • Are the arithmetic libraries used correctly and do they correctly apply rounding directions? hashey
9PrimitiveSecurityAssessment PUBLIC

Project Targets The engagement involved a review and testing of the targets listed below. hyper
Repository <https://github.com/primitivefinance/hyper> Versions
0bcadb70827227dd77a99f58e57f9f8dfed3c79 (Findings 1-17) 577b5c861a9541a91ec39b3618278f93869d0d38
(Findings 19-22) 5a75fc5aad144a1066d9ed19320869c03b305c26 TypeSolidity PlatformEthereum solstat
Repository <https://github.com/primitivefinance/solstat> Version
6e9654163765aac867af1a56fc84462ffdad7d56 TypeSolidity PlatformEthereum hashey
10PrimitiveSecurityAssessment PUBLIC

Project Coverage This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches for covering all of the target components involved a combination of static analysis, manual review, and fuzz testing through Echidna: The Hyper contract: The Hyper contract is the main entry point for creating pools and using them to deposit and swap tokens. The contract has many features, including the following: • Token and pair creation: Users can define reusable "token pairs" structs and use them to create pools; the structs store the number of token decimals and the address of both tokens. We checked to ensure that pairs can be created and used when pools are recreated. • Pool creation: Pools are recreated with a curve to represent a pool with specific parameters. These pools can be set up as mutable or immutable, which determines whether a specified "pool controller" can later change the pool parameters. We reviewed the pool creation code and implemented a series of global system invariants to check the system's assumptions on created pools. • Allocating and unallocating: This functionality allows users to add and remove funds from a pool. We reviewed the arithmetic and rounding directions used in these operations. • Swapping: This functionality allows users to use pools to swap tokens against their pairs. We reviewed the arithmetic in the contract to ensure that the calculations round correctly and use the correct scale factors. We reviewed two different versions of the swap function; the versions of Hyper that we audited are listed in the "Project Targets" section. The HyperLib contract: This contract manages structs that are used by the Hyper contract, including the curve, pair, pools, and orders structs. They provide helper functions to compute amounts, validate parameters, and store state information. We focused on the arithmetic logic in these functions, ensuring that all operations round in the correct direction. 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: hashey
11PrimitiveSecurityAssessment PUBLIC

- **Swapping:** While we spent a significant amount of effort reviewing swaps, this code is incredibly complex. Therefore, we recommend continuing to develop additional fuzz tests against the swapping code to allow fuzzersto explore additional states. Our review of the swapping functionality is inconclusive, and more issues may still be present. •
- **Settlement:** The Hyper contracts implement settlement functionality that iterates over a cached list of tokens to carry out appropriatedebits and credits of reserves. This logic is meant to integrate with the jump processing functionality to provide users with a more gas-efficient way to handle tokens. Due to time limitations, we did not review this functionality. •
- **The Enigma contract and jump processing:** During our audit, we briefly reviewed the functionality of the Enigma contract. In the long term, this contract requires more in-depth analysis to ensure that unique identifiers are encoded and decoded correctly, and it requires thorough fuzz testing to catch unexpected behavior. •
- **Staking and unstaking:** The Primitive code base contains functionality to allow users to stake and un stake assets, providing different interest rates and rewards for putting funds into the system. This feature was deprioritized and removed from the scope of this review. The Primitive team later pushed a commit to remove this functionality from the Hyper contracts. •
- **Custom function dispatching:** The Hyper contracts implement function custom dispatching, which allows functions to be invoked through the fallback function. Due

totimelimitations, this functionality was prioritized and requires additional analysis. hashey
12PrimitiveSecurityAssessment PUBLIC

AutomatedTesting hasheyhasdevelopeduniquetoolsfortestingsmartcontracts.Inthisassessment,we
usedEchidna,asmartcontractfuzzerthatcanrapidlytestsecuritypropertiesviamalicious, coverage-
guidedtestcasegeneration, tocheckvariousssystemstates.

Automatedtestingtechniquesaugmentourmanualsecurityreviewbutdonotreplaceit.

Eachtechniquehaslimitations;forexample,Echidnamaynotrandomlygenerateanedge
casethatviolatesaproperty.Wefollowaconsistentprocesstomaximizetheefficacyof
testingsecurityproperties.WhenusingEchidna,wegenerate100,000,000testcasesper property.

TestHarnessConfiguration Weusedthefollowingtoolsintheautomatedtestingphaseofthisproject:

ToolDescription SlitherAstaticanalysisframeworkthatcanstaticallyverifyalgebraicrelationships
betweenSolidityvariables

EchidnaAsmartcontractfuzzerthatcanrapidlytestsecuritypropertiesviamalicious, coverage-
guidedtestcasegeneration TestResults

Ourautomatedtestingandverificationfocusedonthefollowingsystemproperties:

Globalsysteminvariants:UsingEchidna,wetestedthefollowingexpectedsystem properties.

IDPropertyToolResult 1The locked variableisalwayssetto 1 outsideofexecution.EchidnaPassed

2TheHyperaccount's settled variableissetto true

immediatelyafterdeployment.Thisvariableissavedinthis stateoutsideofexecution. EchidnaPassed hashey

13PrimitiveSecurityAssessment PUBLIC

3TheHyperaccount's prepared variableissetto false outsideofexecution. EchidnaPassed

4Thenumberofwarmtokensoutsideofexecutionisalways 0 .EchidnaPassed

5Thepriorityfeeofacontrolledpoolcanneverbesetto 0 .EchidnaPassed 6The Hyper

contract'stokenbalanceisalwaysgreaterthanor equaltothereservesofthetokensavedinthecontract.

EchidnaPassed 7Amutablepoolcanneverhaveanonzeropriorityfee.EchidnaPassed

8Apool'smaturityisneverlessthanthelasttimestamp.EchidnaPassed

9Apoolwithanonzerolastpriceneverhaszeroliquidity.EchidnaPassed 10Whencalledonapoolwithanonzero
deltaLiquidity value, the getLiquidityDeltas functionneverreturns zero deltaAsset or deltaQuote
amounts. EchidnaFAILED (TOB-HYP R-17)

11Apool'slastpriceisnevergreaterthanthestrikeprice.EchidnaFAILED (TOB-HYP R-14)

12Thestrikepriceofapoolcanneverbesetto 0 .EchidnaFAILED (TOB-HYP R-13)

Paircreation:UsingEchidna,wetestedthefollowingpropertiesoftheHypercontract's

statewhentokenpairsarecreated. IDPropertyToolResult

13Thecreationofatokenpairwiththecorrectpreconditions alwaysucceeds. EchidnaPassed 14The pairNonce

variableofthe Hyper contractincreasesEchidnaPassed hashey 14PrimitiveSecurityAssessment PUBLIC

whenapairiscreated. 15Thesameassetandquotetokenalwayseyieldthesamepair ID. EchidnaPassed

16Thecreationofatokenpairwithtwoidenticaltokensalways fails. EchidnaPassed

17Thecreationofatokenpairwithatokenathaslessthan sixdecimalsalwaysfails. EchidnaPassed

18Thecreationofatokenpairwithatokenathasmorethan 18decimalsalwaysfails. EchidnaPassed

Curvecreation:UsingEchidna,wetestedthefollowingpropertiesoftheHypercontract's

statewhencurvesarecreated. IDPropertyToolResult 19Acurve'sfeecanneverbesetto 0 .EchidnaPassed

20Acurve'spriorityfeecanneverexceedthefee.EchidnaPassed 21Acurve'sdurationcanneverbesetto 0

.EchidnaPassed 22Acurvecanneverhaveavolatilitygreaterthanthevalueof MIN_VOLATILITY . EchidnaPassed

23Acurve's createdAt timestampcanneverbesetto 0 .EchidnaPassed

Poolcreation:UsingEchidna,wetestedthefollowingpropertiesoftheHypercontract's

statewhenpoolsarecreated. IDPropertyToolResult 24Thecreationofanon-

controlledpoolwiththecorrectEchidnaFAILED hashey 15PrimitiveSecurityAssessment PUBLIC

preconditionsdoesnotrevert.(TOB-HYP R-13) 25Thecreationofanon-controlledpoolproperlysetsthepool

statetoimmutable. EchidnaPassed 26Thecreationofanon-controlledpoolsetsthe jit valueto

thedefaultvalueof JUST_IN_TIME_LIQUIDITY_POLICY . EchidnaPassed

27Thecreationofacontrolledpoolwiththecorrect preconditionsnneverreverts. EchidnaPassed

28Thecreationofacontrolledpoolsetsthepoolstateto mutable. EchidnaPassed

29Thecreationofapoolsetsthelasttimestampandcurve createdtothecurrenttimestamp. EchidnaPassed 30The

getVirtualReserves methodalwaysreturnsvalues lessthanthevaluesreturnedbyHyper'srespective

getReserve functionforeachtokenofthepool'spair. EchidnaPassed 31The getAmountsWad

methodalwaysreturnsan amountAssetWad valuethatislessthan1e18. EchidnaPassed 32The getAmountsWad

methodalwaysreturnsan amountQuoteWad valuethatislessthanthereturnvalueof pool.params.strike() .

EchidnaPassed 33Thecreationofacontrolledpoolwithazero-valuepriority feealwaysfails. EchidnaPassed

Updatingpoolparameters:UsingEchidna,wetestedthefollowingpropertiesofthe Hyper

contract'sstatewhenpoolparametersareupdated. IDPropertyToolResult

34Updatingpoolparametersdoesnotupdatethelatest timestamp. EchidnaPassed hashey

16PrimitiveSecurityAssessment PUBLIC

35Updatingpoolparametersmaintainsthesamecontroller address. EchidnaPassed
36Updatingpoolparametersmaintainsthesamecreationtime.EchidnaPassed 37The priorityFee , fee ,
volatility , duration , jit ,and maxTick valuesareupdatedaslongastheyarewithinthe correctbounds.
EchidnaPassed Depositing:UsingEchidna,wetestedthefollowingpropertiesoftheHypercontract'sstate
whenusersdeposittokens. IDPropertyToolResult
38Depositswiththecorrectpreconditionsalwaysucceed.EchidnaPassed 39Whenthe deposit
functioniscalled,thesender'sETH balancedecreasesbythevalueof msg.value . EchidnaPassed
40Depositsdonotalterthe Hyper contract'sETHbalance.EchidnaPassed 41Depositsincreasethe Hyper
contract'sbalanceofthetoken beingdepositedbytheamountbeingdeposited. EchidnaPassed
42Depositsincreasethegivenpool'sreservevalueforthe tokenbeingdepositedbytheamountbeingdeposited.
EchidnaPassed Fundinganddrawing:UsingEchidna,wetestedthefollowingpropertiesoftheHyper
contract'sstatewhenusersfundanddrawtokensfromapool. IDPropertyToolResult
44Fundingoperationswiththecorrectpreconditionsalways succeed. EchidnaPassed 45Callingthe fund
functionwithinsufficientcallerfundsalways fails. EchidnaPassed hasheye
17PrimitiveSecurityAssessment PUBLIC

46Callingthe fund functionwithinsufficientallowancealways fails. EchidnaPassed
47Aftertokensarefunded,thesender'stokenbalance decreasesbythefundedamount. EchidnaPassed
48Aftertokensarefunded,theHyper-registeredbalanceofthe user'stokenincreasesbythefundedamount.
EchidnaFAILED (TOB-HYP R-12) 49Aftertokensarefunded,thereservebalanceofthetoken
increasesbythefundedamount. EchidnaFAILED (TOB-HYP R-12)
50Aftertokensarefunded,thetokenbalanceofthe Hyper contractincreasesbythefundedamount. EchidnaPassed
51Callingthe fund functionwithzerofundsalwaysucceeds.EchidnaPassed 52Callingthe draw
functionwithsufficientbalancealways succeeds. EchidnaPassed 53Callingthe draw
functionwithinsufficientbalancealways fails. EchidnaPassed 54Aftertokensaredrawn,theHyper-
registeredbalanceofthe user'stokendecreasesbythedrawnamount. EchidnaPassed
55Aftertokensaredrawn,thesavedreservebalancedecreases bythedrawnamount. EchidnaPassed
56Aftertokensaredrawn,therecipient'stokenbalancealways increasesbythedrawnamount. EchidnaPassed
56Aftertokensaredrawn,the Hyper contract'stokenbalance decreasesbythedrawnamount. EchidnaPassed
57Callingthe draw functionwiththezeroaddressasthe recipientalwaysfails. EchidnaPassed hasheye
18PrimitiveSecurityAssessment PUBLIC

58Fundinganddrawingoperationsalwaysresultinthesame pre-andpost-state. EchidnaPassed
Allocatingandremovingfunds:UsingEchidna,wetestedthefollowingpropertiesofthe Hyper
contract'sstatewhenfundsareallocatedandremoved. IDPropertyToolResult 59Callingthe allocate
functionwiththecorrectpreconditions alwaysucceeds. EchidnaPassed
60Afterfundsareallocated,theliquidityforthespecified poolId increasesbythevalueof deltaLiquidity .
EchidnaPassed 61Afterfundsareallocated,theliquidityforthespecified poolId neverdecreases.
EchidnaPassed 62Afterfundsareallocated,the Hyper contract'sreservetoken
balanceforthetokenincreasesbythevaluesof deltaAsset and deltaQuote ifthecallerdoesnothave
enoughtokensintheirbalance. EchidnaPassed 63Afterfundsareallocated,the Hyper contract'stokenbalance
increasesbythevaluesof deltaAsset and deltaQuote . EchidnaPassed
64Afterfundsareallocated,thecaller'sfreeliquidityposition alwaysincreasesbythevalueof
deltaLiquidity . EchidnaPassed 65Afterfundsareallocated,thephysicalbalancefortheasset
tokenincreases. EchidnaPassed 66Afterfundsareallocated,thephysicalbalanceforthequote
tokenincreases. EchidnaPassed 67Theallocationoffundstoanonexistentpoolalwaysreverts.EchidnaPassed
68Theallocationofzerodeltaliquidityalwaysreverts.EchidnaPassed hasheye
19PrimitiveSecurityAssessment PUBLIC

69Afterfundsareallocated,ifthefeegrowthassetpoolhas changed,thefeegrowthpositionisnonzero.
EchidnaPassed 70Withthecorrectpreconditions,callstothe unallocate functionneverrevert.
EchidnaPassed 71Afterfundsareunallocated,thepoolliquiditydecreasesby theunallocatedamount.
EchidnaPassed 72Afterfundsareunallocated,thecaller'spositionliquidity
decreasesbytheunallocatedamount. EchidnaPassed 73Thetrackedassetreservebalancedoesnotchangeafter
fundsareunallocated. EchidnaPassed 74Thetrackedquotereservebalancedoesnotchangeafter
fundsareunallocated. EchidnaPassed 75Thephysicalassetbalanceofthe Hyper contractdoesnot
changeafterfundsareunallocated. EchidnaPassed 76Whencalledbyacallerwithoutaposition,the unallocate
functionreverts. EchidnaWIP 77Theunallocationoffundstoanonexistentpoolreverts.EchidnaPassed hasheye
20PrimitiveSecurityAssessment PUBLIC

Swapping:UsingEchidna,wetestedthefollowingpropertiesofthe Hyper contract'sstate
whenusersswaptokens.AlthoughEchidnapassedonthe100,000fuzzingruns,we
recommendcontinuingtorunthetestsuitewithafocusontargetingspecificsafeinput
rangestoallowEchidnaoeeasilyexploreallareas.Thepropertieslistedbelowthatrequire
additionaladjustmentsaregiventhestatus"WIP". IDPropertyToolResult
78Whencalledwiththecorrectpreconditions,the swap functiondoesnotrevert. EchidnaWIP

79Swapsafterthegivenpoolreachesmaturityalwaysrevert.EchidnaWIP
80Swapsonanonexistentpoolalwaysrevert.EchidnaWIP 81Swapswithazeroamountalwaysrevert.EchidnaWIP
82Swapswithazeroamountalwaysrevert.EchidnaWIP
83Swappinganassettokenintoapoolalwaysdecreasesthe priceofthetoken.* EchidnaWIP
84Swappingaquotetokenintoapoolalwaysincreasesthe priceofthetoken.* EchidnaWIP
85Swappinganassettokenintoapoolincreasesthetoken reserves.* EchidnaWIP
86Pricesofthegiventokensalwayschangeduring swap operations.* EchidnaWIP
87Liquiditydoesnotchangeduring swap operations.*EchidnaWIP 88Feegrowthcheckpointsincreaseduring
swap operations.*EchidnaWIP 89Aswapinonedirectionfollowedbyaswapintheopposite
directionresultsinthesamestatepre-andpost-swap. EchidnaFAILED (TOB-HYP R-23) hasheyeye
21PrimitiveSecurityAssessment PUBLIC

90Aswapofaquotetokenwithmorethan14decimalplaces fortheassettokenreverts. EchidnaPassed
*InvarianttestsrewrittenjointlywithPrimitivethroughouttheaudit.
Mathfunctioninvariants:UsingEchidna,wetestedthefollowingpropertiesofthe Hyper contract'sstate.
IDPropertyToolResult 91The expWad function'soutputdomainis $[0, \infty)$.EchidnaPassed 92The expWad
functionisstrictlymonotonicallyincreasing.EchidnaPassed 93The expWad function'smaximumerroris 0
when x is approximately 0 . Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \exp(x) - 1 - x | < \epsilon$
EchidnaPassed 94The sqrt functionisstrictlymonotonicallyincreasing.EchidnaPassed 95The sqrt
functionroundsdown.EchidnaPassed 96The sqrt function'smaximumerroris 0 . Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \sqrt{x} - \lfloor \sqrt{x} \rfloor | < \epsilon$
EchidnaPassed 97The pdf function'soutputdomainis $[0, 1/\sqrt{2\pi})$.
Note:Theoutputdomainis $[0, 1/\sqrt{2\pi}]$;however,the upperboundshouldnotbeincludedwhenrounding
correctly.Theresultisinconsistentlyroundedupwhen x is 0 . EchidnaFAILED (TOB-HYP R-22) 98The pdf
functionismonotonicallydecreasing. Itsmaximumerroronvalueswithinitinputdomainis approximately
 $0.4e18$. Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \exp(-x^2) - \lfloor \exp(-x^2) \rfloor | < \epsilon$ EchidnaFAILED (TOB-HYP R-22) hasheyeye
22PrimitiveSecurityAssessment PUBLIC

99The pdf function'smaximumerroris 0 when x is approximately 0 . Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \exp(-x^2) - 1 + 2x^2 | < \epsilon$
EchidnaPassed 100The erfc function'soutputdomainis $[0, 2]$.EchidnaFAILED (TOB-HYP R-
22) 101The erfc functionismonotonicallydecreasing. Itsmaximumerroronvalueswithinitinputdomainis
approximately $1.36e57$. Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \operatorname{erfc}(x) - \lfloor \operatorname{erfc}(x) \rfloor | < \epsilon$ EchidnaFAILED (TOB-HYP
R-22) 102The erfc function'smaximumerrorisapproximately $3e12$ when x isapproximately 0 .
Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \operatorname{erfc}(x) - 1 + 2x^2 | < \epsilon$ EchidnaFAILED (TOB-HYP R-22) 103The
ierfc functionrevertshengivenvaluesoutsideofits inputdomain. Note:Itreturnsthehard-codedvalue
 $+/-1e20$ onvalues outsideofitsdomain,butitshouldreturn $+/-\infty$ or,better yet,revert. EchidnaFAILED
(TOB-HYP R-22) 104The ierfc functionismonotonicallydecreasing. Itsmaximumerrorisapproximately
 $4.9e12$ onvalues withinitinputdomain. Thisismeasuredas $\forall \epsilon > 0 \exists \delta > 0 \forall x \in [0, \delta] \implies | \operatorname{ierfc}(x) - \lfloor \operatorname{ierfc}(x) \rfloor | < \epsilon$
EchidnaFAILED (TOB-HYP R-22) 105The ierfc function'smaximumerroris 0 when x is approximately 1 .
EchidnaPassed hasheyeye 23PrimitiveSecurityAssessment PUBLIC

CodebaseMaturityEvaluation hasheyeyesatraffic-
lightprotocoltoprovideeachclientwithaclearunderstandingof
theareasinwhichitscodebaseismature,immature,orunderdeveloped.Deficiencies
identifiedhereoftenstemfromrootcauseswithinthesoftwaredevelopmentlifecyclethat
shouldbeaddressedthroughstandardizationmeasures(e.g.,theuseofcommonlibraries,
functions,orframeworks)ortrainingandawarenessprograms. CategorySummaryResult
ArithmeticTheHypercodebaseusesasignificantamountof
complexarithmetic.Wefoundissuesrelatedtoprecision loss,theriskthatfundscouldbetrapped,andincorrect
roundingdirections,whichattackerscouldexploitfor profit(TOB-HYPR-3,TOB-HYPR-4,TOB-HYPR-5, TOB-HYPR-
6,TOB-HYPR-7,TOB-HYPR-14,TOB-HYPR-16, TOB-HYPR-19,TOB-HYPR-20,TOB-HYPR-22,and TOB-HYPR-
23).Theseareasofthecodebaserequire additionaldocumentationspecifyingthearithmetic
operations'expectedbehaviorandadditional invariant-basedtestingtoensurethattheybehaveas expected.
Weak AuditingThefunctionsintheHypercontractsemitinsufficient
eventstothehelpPrimitivedetectunexpectedbehavior.A respectiveeventisproperlyemittedforeach state-
changingfunction.However,therearesomecases inwhichusersreceiveobscureunderflow/overflowerror
messagesduetotheuseofassemblyoperationsto checkagainstvalidinput(appendixG).Werecommend
writingeacheventsothatitexplainsandshowsthe updatedstate. Moderate Authentication/ AccessControls
Mostoftheexpectedaccesscontrolsacrossthesystem aredocumented.However,documentationonrolesand
privilegesneedstobeaddedtothecodebase.Moreover, theaccesscontroltestsneedtobemoreconclusiveto
testbothhappyandunhappypaths. Moderate hasheyeye 24PrimitiveSecurityAssessment PUBLIC

Complexity Management Somefunctionsinthe Hyper contractarewellisolated
andeasytotest.However,certainfunctions,suchas allocate and unallocate ,usenestedhelperfunctions
toupdatethestateofpools,reserves,andpositions, whichmakethemslightlyhardertotrack.Certainareas
inthecodebaseuseassemblyoverhigh-levelSolidity code,whichwouldbenefitfromadditional
documentation.Moreover,the Enigma contractand customfunctiondispatchingfeaturerequirefurther

analysis. Moderate Decentralization Pool controllers have significant control over the system, as they have the ability to update the parameters of a curve. As a result, users have to check that their pools are set up with the values that they intended; and for mutable pools, users have to check that the updated parameters still reflect the option they want to purchase. The core contracts of the system are not upgradeable. Moderate Documentation The Hyper contracts contain a significant amount of documentation, which provides a one-to-one cross-reference between the terms and definitions in the whitepaper and those in the code. The terms related to the system are explained in the whitepaper through equation derivations, interactive curves through Desmos, and detailed explanations of system components. In many cases, the documentation provided us with sufficient context to understand the complex behavior of the system. However, the contracts lack per-function documentation for complicated logics such as the swap function. Additionally, we found minor issues related to incorrect and/or outdated NatSpec comments (TOB-HYPR-2, TOB-HYPR-10); those areas of the code base should be closely reviewed to ensure they behave as expected. Primitive developed thorough documentation of the code base's global system invariants that specify the expected state of pools and pairs and per-function preconditions and postconditions that allow us to easily reason about whether the invariants hold. Although many of these invariants are implemented, we recommend that Primitive continue to develop additional Moderate has eye 25 Primitive Security Assessment PUBLIC

documentation on invariants, specifically those related to the adjustment of pool parameters and the tracking of ticks. Transaction Ordering Risks We found one issue that would allow an attacker to profit immediately after a controller has changed parameters of a mutable pool (TOB-HYPR-15). However, additional investigation is required to determine whether transaction ordering poses another risk to the other sections of the Hyper code base, particularly to the custom function dispatching feature. Further Investigation Required Low-Level Manipulation Assembly versions of functions are used in many areas of the system; these versions do not include checks that high-level Solidity versions would have provided (TOB-HYPR-22). Many of these assembly blocks are missing inline comments describing their operations and are missing high-level reference implementations that could be used in differential fuzzing. Weak Testing and Verification Unit tests are used for the majority of the code base; however, the tests are missing unhappy path testing and corner cases in input bounds. Although basic fuzz tests are also present, we recommend that Primitive continue to extend the current end-to-end fuzz tests to catch undesired behavior that could occur due to pool sets of multiple token decimals and due to complex operations. Weak has eye 26 Primitive Security Assessment PUBLIC

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

ID	Title	Type	Severity
1	Lack of zero-value checks on functions	Data Validation	Informational
2	Documentation discrepancy in computePriceWithChangeInTau	Undefined Behavior	Informational
3	Risk of token theft due to possible integer underflow in slt	Data Validation	High
4	Risk of token theft due to unchecked type conversion	Data Validation	High
5	Users can swap without paying any fees	Data Validation	Medium
6	Swap function returns incorrectly scaled output token amount	Data Validation	High
7	Liquidity providers can withdraw total fees earned by a pool	Undefined Behavior	High
8	Asset token price deviates from the price curve of the pool	Undefined Behavior	Undetermined
9	New pair creation can overwrite existing pairs	Undefined Behavior	High
10	Error in Invariant.getX	Undefined Behavior	Informational
11	Pools with overflowing maturity dates can be created	Data Validation	Low

has eye 27 Primitive Security Assessment PUBLIC

12	Minting funds to the Hyper contract arbitrarily increases the next caller's balance	Configuration	Informational
13	Pool strike price could be zero due to lack of lower bound check on maxTick	Data Validation	High
14	Rounding error allows liquidity to be added without depositing tokens	Data Validation	High
15	Attacker can steal funds by changing parameters calls to steal funds	Undefined Behavior	High
16	Limited precision in strike prices due to fixed tick spacing	Data Validation	Low
17	Function that rounds by adding 1 results in unexpected behavior	Undefined Behavior	Informational
18	Solidity compiler optimizations can be problematic	Undefined Behavior	Informational
19	getAmountOut returns incorrect value when called by controller	Data Validation	Low
20	Mismatched base unit comparison can inflate limit tolerance	Data Validation	Medium
21	Incorrect implementation of edge cases in getY function	Undefined Behavior	Low
22	Lack of proper bound handling for solstat functions	Undefined Behavior	Undetermined
23	Attacker can steal funds by swapping in both directions	Undefined Behavior	High

has eye 28 Primitive Security Assessment PUBLIC

Detailed Findings 1. Lack of zero-value checks on functions Severity: Informational Difficulty: High Type: Data Validation Finding ID: TOB-HYPR-1 Target: contracts/Hyper.sol Description Certain setter functions fail to validate incoming arguments; therefore, callers of these functions can accidentally set important state variables to the zero address. For example, the constructor function in the Hyper contract, which sets the WETH contract, lacks zero-value checks. constructor(address weth) { WETH=weth; __account__.settled=true; } Figure 1.1: The constructor function in Hyper.sol If the WETH address is set to the zero address, the admin must redeploy the Hyper

contractstoresettheaddress'svalue. ExploitScenario Alice deploys a new version of the Hyper contract but mistakenly enters the zero address for the WETH address. She must re-deploy the system to reset the value of WETH. Recommendations Short term, add zero-value checks to all function arguments to ensure that users cannot accidentally set incorrect values, misconfiguring the system. Long term, use the Slither static analyzer to catch common issues such as this one. Consider integrating a Slither scan into the project's continuous integration pipeline, pre-commit hooks, or build scripts. hashey 29 Primitive Security Assessment PUBLIC

2. Documentation discrepancy in computePriceWithChangeInTau

Severity: Informational Difficulty: Undetermined Type: Undefined Behavior Finding ID: TOB-HYPR-2 Target: contracts/Libraries/Price.sol Description The formula provided in the @custom field of the NatSpec comment for the computePriceWithChangeInTau function does not match the formula implemented in the function body. The discrepancy between the documentation and implementation can cause end users to misunderstand what the function actually does. /**
*@dev Computes change in price given a change in time in seconds. *@param stkWAD *@param volPercentage
*@param prcWAD *@param tauSeconds *@param epsilonSeconds *@custom:math $P(\tau - \epsilon) = (P(\tau)^{\sqrt{1 - \epsilon/\tau}} / K^2) e^{((1/2)(\tau^2)(\sqrt{\tau}\sqrt{\tau - \epsilon} - (\tau - \epsilon)))}$ */ Figure 2.1: The NatSpec comment for the computePriceWithChangeInTau function in Price.sol The function body implements the following formula: $(\tau - \epsilon) = ((\tau/\sigma)^{\sqrt{1 - \epsilon/\tau}}) * \sigma * \sigma^{((1/2)(\sigma^2)(\sqrt{\tau}\sqrt{\tau - \epsilon} - (\tau - \epsilon)))}$
The documented and implemented formulas will result in different values for $P(\tau - \epsilon)$. Exploit Scenario Alice calculates the result of her investment using formulas from the codebase's NatSpec comments. The result of her calculations convinces her to submit transactions. When Alice completes her series of transactions, the end result differs from her expectations. Recommendations Short term, correct the formula error in the NatSpec comment for computePriceWithChangeInTau so that it matches the implementation. hashey 30 Primitive Security Assessment PUBLIC

Long term, thoroughly proofread the NatSpec comments throughout the codebase, especially where they describe important formulas and operations. hashey 31 Primitive Security Assessment PUBLIC

3. Risk of token theft due to possible integer underflow in slt

Severity: High Difficulty: Low Type: Data Validation Finding ID: TOB-HYPR-3 Target: contracts/Assembly.sol Description An attacker can steal funds from a pool using a function that fails to revert on unexpected input. The addSignedDelta function is used by the allocate, unallocate, and unstake functions to alter the liquidity of a pool. These functions lack checks to ensure that the input values are within permissible limits; instead, they rely on the addSignedDelta function to revert on unexpected inputs. The addSignedDelta function checks for the sign of the delta value; it subtracts its two's complement from the input value if delta is a negative integer. After the subtraction operation, the code checks for underflow using the slt (signed-less-than) function from the EVM dialect of the YULL language. The slt function assumes that both arguments are in a two's complement representation of the integer values and checks their signs using the most significant bit. If an underflow occurs in the previous subtraction operation, slt's output value will be interpreted as a negative integer with its most significant bit set to 1. Because the slt function will find that the output value is less than the input value based on their signs, it will return 1. However, the addSignedDelta function expects the result of slt to be 0 when an underflow occurs; therefore, it fails to capture the underflow condition correctly. function addSignedDelta(uint128 input, int128 delta) pure returns (uint128 output) { bytes memory revertData = abi.encodeWithSelector(InvalidLiquidity.selector); assembly { switch slt(delta, 0) // delta < 0 ? 1 : 0 // negative delta case 1 { output := sub(input, add(not(delta), 1)) switch slt(output, input) // output < input ? 1 : 0 case 0 { // not less than revert(add(32, revertData), mload(revertData)) // 0x1fff9681 } } } hashey 32 Primitive Security Assessment PUBLIC

```
// position delta case 0 { output := add(input, delta) switch slt(output, input) // (output < input ? 1 : 0) == 0 ? 1 : 0 case 1 { // less than revert(add(32, revertData), mload(revertData)) // 0x1fff9681 } } }
```

Figure 3.1: The vulnerable addSignedDelta function in Assembly.sol Exploit Scenario Alice allocates 100 USD into a pool. Eve, an attacker, calls the unallocate function with the 100 USD as the deltaLiquidity argument. The addSignedDelta function fails to revert on the integer underflow that results, and Eve is able to withdraw Alice's assets. Recommendations Short term, take one of the following actions: • Correct the implementation of the addSignedDelta function to account for underflows. • Use high-level Solidity code instead of assembly code to avoid further issues; assembly code does not support sub256-bit types. Regardless of which action is taken, add test cases to verify the correctness of the new implementation; add both unit test cases and fuzz test cases to capture all of the edge cases.

Longterm, carefully review the code base to find assembly code and verify the correctness of these assembly code blocks by adding test cases. Do not rely on certain behavior of assembly code while working with sub256-bit types because this behavior is not defined and can change at any time. Note: Do not consider using the lt function in place of the slt function because it is not sufficient to capture all of the overflow and underflow conditions. hashey 33 Primitive Security Assessment PUBLIC

4. Risk of token theft due to unchecked type conversion Severity: High Difficulty: Low Type: Data Validation Finding ID: TOB-HYPR-4 Target: contracts/Assembly.sol Description An attacker can steal funds from a pool using a function that fails to revert on unexpected input. The addSignedDelta function is used by the allocate, unallocate, and unstake functions to alter the liquidity of a pool. These functions lack checks to ensure that the input values are within permissible limits; instead, they rely on the addSignedDelta function to revert on unexpected inputs. When the value of delta is a positive integer, the result of the add function (from the EVM dialect of the YULL language) will overflow; however, the code cannot capture this overflow. This is because the arguments of the function are 128-bit types, but the assembly code does not have types and operates on 256-bit values. The add function returns a 256-bit value as its result. The addition of two 128-bit integers can never overflow a 256-bit integer. For this reason, the result of the add function will never wrap around the maximum value of a 256-bit integer, and the slt function will never find the output value to be less than the input value, which means it will never return 1 to indicate that an overflow has occurred. As a result, the code fails to capture the overflow condition correctly.

```
function addSignedDelta(uint128 input, int128 delta) pure returns (uint128 output) {
    bytes memory revertData = abi.encodeWithSelector(InvalidLiquidity.selector);
    assembly {
        switch slt(delta, 0) // delta < 0 ? 1 : 0 // negative delta
        case 1 { output := sub(input, add(not(delta), 1))
        switch slt(output, input) // output < input ? 1 : 0
        case 0 { // not less than
            revert(add(32, revertData), mload(revertData)) // 0x1fff9681 } } // position delta
        case 0 {
            output := add(input, delta)
            hashey 34 Primitive Security Assessment PUBLIC
        }
    }
}
```

switch slt(output, input) // (output < input ? 1 : 0) == 0 ? 1 : 0 case 1 { // less than
revert(add(32, revertData), mload(revertData)) // 0x1fff9681 } } } } Figure 4.1: The vulnerable addSignedDelta function in Assembly.sol

There are multiple ways in which an attacker could use this issue to withdraw more liquidity than they have deposited in a pool. Exploit Scenario 1 Alice and Bob allocate 500 USDC each into a USDC-ETH pool. The total allocated liquidity is 1,000 USDC. Eve, an attacker, unallocates 1,000 USDC from the entire pool, withdrawing everyone's assets. Exploit Scenario 2 Alice allocates 100 USDC into a pool. Eve calls the unstake function to increase the value of her own liquidity without depositing any assets and then calls unallocate to withdraw the funds from the pool. Recommendations Short term, take one of the following actions: • Correct the implementation of the addSignedDelta function to account for overflows. • Use high-level Solidity code instead of assembly code to avoid further issues; assembly code does not support sub256-bit types. Regardless of which action is taken, add test cases to verify the correctness of the new implementation; add both unit test cases and fuzz test cases to capture all of the edge cases. Long term, carefully review the code base to find assembly code and verify the correctness of these assembly code blocks by adding test cases. Do not rely on certain behavior of assembly code while working with sub256-bit types because this behavior is not defined and can change at any time. hashey 35 Primitive Security Assessment PUBLIC

5. Users can swap without paying any fees Severity: Medium Difficulty: Low Type: Data Validation Finding ID: TOB-HYPR-5 Target: contracts/Hyper.sol Description While the swap function is being processed, the input and output values of the swap are calculated with and without the fee amount assessed from the total. However, if the user's requested amount exceeds the maxInput amount, there is no addition of the swap fee to deltaInput, which is dismissed, allowing the user to swap without paying the fee. The swap function calls the _swapExactIn function, which contains logic to save intermediate values while processing token swaps. The deltaInput variable is initially used to derive the nextIndependent value without the fee amount assessed. That fee amount should be added back to deltaInput afterward so that it can be added to the total amount withdrawn from the user later in the process. The fee amount is added back to deltaInput when _swap remainder is less than or equal to maxInput, but not if _swap remainder is greater than maxInput. If the fee is not added, then when deltaInput is added to _swap.input, the fee will not be represented in that amount, which means that it will not be withdrawn from the user.

```
if(_swap.remainder > maxInput){
    deltaInput = maxInput - _swap.feeAmount;
    nextIndependent = liveIndependent + deltaInput.divWadDown(_swap.liquidity);
    _swap.remainder -= (deltaInput + _swap.feeAmount);
} else {
    deltaInput = _swap.remainder - _swap.feeAmount;
    nextIndependent = liveIndependent + deltaInput.divWadDown(_swap.liquidity);
    deltaInput = _swap.remainder; // Swap input amount including the fee payment.
}
```

```

_swap.remainder=0; //Cleartheremainderintozero,astheorderisgreaterthanfilled. }
//Computetheoutputoftheswapbycomputingthedifferencebetweenthe dependentreserves.
if(_state.sell)nextDependent=rmm.getYWithX(nextIndependent);
elsenextDependent=rmm.getXWithY(nextIndependent); _swap.input+=deltaInput; _swap.output+=
(liveDependent-nextDependent); Figure5.1:The _swapExactIn functionin Hyper.sol hasheye
36PrimitiveSecurityAssessment PUBLIC

```

ExploitScenario Eveexecutesaswapinwhichtheremainderisgreaterthanthemaximuminput.Duetothe calculationsinthe swap function,Evecanswapwithoutpayingthefeeamount. Recommendations Shortterm,fixthefunctiontofactorin feeAmounts whenthevalueof remainder is greaterthanthevalueof maxInput . Longterm,thoroughlyanalyzethesystemtoidentifyinvariantsrelatedtoproperfee assessment.FuzzthoseinvariantsusingEchidnaensurethatthefunctionsreturnthe expectedvaluesandthattheyareaccurate. hasheye 37PrimitiveSecurityAssessment PUBLIC

6.Swapfunctionreturnsincorrectlyscaledoutputtokenamount Severity:HighDifficulty:Low Type:DataValidationFindingID:TOB-HYPR-6 Target: contracts/Hyper.sol Description The swap function's output valueisgivenperunitofliquidityinthegivenpool,butitisnot scaledbythepool'stotalliquidity.Asaresult,userswillnotreceivethenumberoftokens thattheyexpectswaps: functionswap(uint64poolId, boolsellAsset, uintamount, uintlimit)externallockinteractionsreturns(uintoutput,uintremainder){ Figure6.1:Thefunctionsignatureofthe swap functionin Hyper.sol The output tokenvalueiscalculatedusingthedifferencebetweenthe liveDependent and nextDependent variables,bothofwhicharecalculatedusingthereserveamountof theinputtoken.However,theoutputvalueisnotmultipliedbythetotalliquidityvalueof thepool,sotheoutputamountisscaledincorrectly: _swap.output+=(liveDependent-nextDependent); Figure6.2:The output calculationinthe _swapExactIn functionin Hyper.sol Thiscausethenumberofoutputtokenstobeeithertoofewortoomany,dependingon thecurrentamountofliquidityinthepool. Primitivealsodiscoveredthisissueduringthecodereview. ExploitScenario AliceswapsWETHforUSDCusingHyper.Thepoolhaslessthan1wadofliquidity.The tokenoutputvaluethatisreturnedtoAliceislessthanwhatitshouldbe. Recommendations Shortterm,revise the swap functionsothatitmultipliesthe output tokenbythetotal liquiditypresentinthepoolinwhichtheswaptakesplace. hasheye 38PrimitiveSecurityAssessment PUBLIC

Longterm, identifyadditionalssysteminvariantsandfuzzthemusingEchidnaensure thatthefunctionsreturntheexpectedvaluesandthattheyareaccurate. hasheye 39PrimitiveSecurityAssessment PUBLIC

7.Liquidityproviderscanwithdrawtotalfeesearedbyapool Severity:HighDifficulty:Low Type:UndefinedBehaviorFindingID:TOB-HYPR-7 Target: contracts/Hyper.sol Description The syncPositionFees functionisimplementedincorrectly.Asaresult,liquidity providerscanwithdrawthetotalfeesearedbyapoolanddrainassetsfromthecontract reserves. Poolsearnfeesfromswaps,whichshouldbedistributedamongtheliquidityproviders proportionaltotheliquiditytheyprovidedduringtheswaps.However,the Hyper contract insteadistributesthetotalfeesearedbythepooltoeveryliquidityprovider,resultingin thedistributionofmoretokensinfeesthanearnedbythepool. Asshowninfigure7.1,the syncPositionFees functionisusedtocomputethefeeeared byaliquidityprovider.Thisfunctionmultipliesthefeeearedperwadofliquiditybythe liquidity value,providedasanargumenttothefunction. functionsyncPositionFees(HyperPositionstorageself, uintliquidity, uintfeeGrowthAsset, uintfeeGrowthQuote)returns(uintfeeAssetEarned,uintfeeQuoteEarned){ uintcheckpointAsset=Assembly.computeCheckpointDistance(feeGrowthAsset, self.feeGrowthAssetLast); uintcheckpointQuote=Assembly.computeCheckpointDistance(feeGrowthQuote, self.feeGrowthQuoteLast); feeAssetEarned=FixedPointMathLib.mulWadDown(checkpointAsset,liquidity); feeQuoteEarned=FixedPointMathLib.mulWadDown(checkpointQuote,liquidity); self.feeGrowthAssetLast=feeGrowthAsset; self.feeGrowthQuoteLast=feeGrowthQuote; self.tokensOwedAsset+=SafeCastLib.safeCastTo128(feeAssetEarned); self.tokensOwedQuote+=SafeCastLib.safeCastTo128(feeQuoteEarned); } Figure7.1:The syncPositionFees functionin HyperLib.sol hasheye 40PrimitiveSecurityAssessment PUBLIC

The syncPositionFees functionisusedinthe _changeLiquidity and claim functions definedinthe Hyper contract.Inbothlocations,whenthe syncPositionFees functionis called,thevalueofthepool'stotalliquidityisprovidedasthefirstargument,whichisthen multipliedbythefeeearnedperwadofliquidity.Thevalueresultingfromthe multiplicationisthenaddedtothefeeearnedbytheliquidityprovider,whichmeansthe totalfeeearedbythepoolisaddedtothefeeearnedbytheliquidityprovider. Therearemultiplewaysinwhichanattackercouldusethisissuetowithdrawmorethan whattheyhaveearnedinfees. ExploitScenario1 Eveprovidesminimalliquiditytoapool.Evewaitsforsometimeforsomeswapsto happen.Shecallsthe claim functiontowithdrawthetotalfeeearedbythepoolduring

theperiodofAliceandotherusershaveprovidedliquidity. ExploitScenario2
Eveprovidesminimalliquiditytoapoolusing10accounts.Evemakesomelargeswapsto
accruefeesinthepool.Shethencalls the claim functionfromall10accountstowithdraw
10timesthetotalfeeshepaidfortheswaps.Sherepeatsthesestepstodrainthecontract reserves.
Recommendations Shortterm, revisetherelevantcodesothatthevalueofthesumofaliquidityprovider's
liquidity(freeLiquidity summedwith stakedLiquidity)ispassedasanargumentto the syncPositionFees
functioninsteadoftheentirepool'sliquidity. Longterm,takethefollowingactions: •
Inallfunctions,documenttheirarguments,theirmeanings,andtheirusage. •
Addunittestcasesthatcheckallhappyandunhappypaths. •
Identifyadditional systeminvariantsandimplementEchidnatocapturebugsrelated tothem. hashey
41PrimitiveSecurityAssessment PUBLIC

8.Assettokenpricedeviatesfromthepricecurveofthepool Severity:UndeterminedDifficulty:Undetermined
Type:UndefinedBehaviorFindingID:TOB-HYPR-8 Target: contracts/Hyper.sol Description
Theassettokenpricedeviatesfromthepricecurveofthepoolwith eachswapoperation
becauseofthepriceadjustmentperformedinthe swap function. Oneachswap,the _swapExactIn
functioncomputesnewvirtualreservesofapoolbased
ontheuser'sinputandthencomputesthenewpriceoftheassettokenbasedonthe pool's
newvirtualreserves.Asshowninfigure8.1,afactorof10 11 weiisaddedtothecalculated price. _swap.price=
(nextPrice*10_000_001)/10_000_000; Figure8.1:Thepriceadjustmentstatementinthe _swapExactIn
functionin Hyper.sol Thisadjustedpriceisthenusedtocalculatetheoutputamountinthenextswapoperation.
Thepriceoftheassettokenincreasesbythefactorof10 11 weiwith eachswapoperation,
whichcausethethepriceoftheassettokentodeviatefromthepricecurveofthepool.
Thisadjustedpriceisalsousedtocalculatethepool'snewvirtualreservesonsubsequent
swaps,whichmeansthatthevirtualreservesofthepoolwillbedifferentfromthoseused
tocalculatethisprice.Thiscreatesadifferencebetweenthecontract'sbalanceofpool
tokensandthepool'svirtualreserves;thisdifferenceincreaseswith everyswapoperation,
causingassetstobecomestuckinthecontract. Recommendations
Shortterm,investigatethemaximumamountofdeviationthatcanoccurbetweenthe
assettokenpriceandthepricecurveofthepooltoensurethattheerrorfitswithinsafe bounds.
Longterm,executethorougheconomicanalysisontheimplicationsofanyupdateto
systemvariables.Thisshouldinclude maximumerrorcalculationsforallvariables. hashey
42PrimitiveSecurityAssessment PUBLIC

9.Newpaircreationcanoverwriteexistingpairs Severity:HighDifficulty:High
Type:UndefinedBehaviorFindingID:TOB-HYPR-9 Target: contracts/Hyper.sol Description
Anoverflowofthemaximumvalueof uint24 couldoccurinthe _createPair() function,
whichcanbeusedtooverwriteexistingpairs. Asshowninfigure9.1,the _createPair()
functionusesanuncheckedblocktocompute andcastthevalueof getPairNonce toassignthevalueof pairId
.This pairId isusedas akeyinthe pairs mappingtostoreinformationrelatedtoaspecificpair. unchecked{
pairId=uint24(++getPairNonce); } Figure9.1:The pairId assignmentinthe _createPair() functionin
Hyper.sol AccordingtotheSoliditydocumentation,values thatundergoexplicit typeconversionare
truncatedifthenewtypecannot holdallofthebitsrequiredtorepresentthenewvalue. Here,thetypeof
getPairNonce isconvertedfrom uint256 to uint24 ,soifthenewvalue overflownthelimitofthe
value,thehigher-orderbitsof getPairNonce willbe
truncatedtoanunexpectedvalue.Thismeansthatexistingpoolswillbeoverwrittenby
everynewpaircreationoperation,resultingin aninconsistentcontractstatewiththe followingissues: •
Twosetsofassetswillstorethesame pairId inthe getPairId mapping. • Thevalueof HyperPair
willbeoverwritteninthe pairs mapping. •
Allofthepreviouspoolsc Createdforthepairwillstillholdthepreviousvalueof HyperPair .
Thisoverflowlimitmaynotseemfeasibletoreachbecausethelimitofthe uint24
typeis16,777,215.Itwouldtakealotof timeandcostalotofgastocreate so manypairs.However,becauseofthelow-
costnatureandhighertransactionthroughputof
theL2networks,amalicioususercouldexploitthisissuetoconductaDoSattackonthe
protocolwithinsignificantfinancialloss. hashey 43PrimitiveSecurityAssessment PUBLIC

Wealsofoundthatapoolexistencecheckisimplementedin _createPool function,but
aninlinecommentindicatesPrimitive'splanstoremovethischeck.Werecommend
keepingthischecktopreventsimilar poolId overflowandoverwritingissues. ExploitScenario
EvedeploysnumerousfakeERC-20tokencontracts.ShethenusesthesesERC-20tokensto
create16,777,215boguspairsinthe Hyper contract.Now, everynewpairoverwrites
existingpairs,makingthisinstanceoftheprotocolunusable. Recommendations
Shortterm,makethefollowingchanges: 1.Addacheckofthe pairId in _createPair()
toensurethatithasnotalready beenusedtopreventexistingpairsfrombeingoverwritten.
2.Increasetheupperboundofthevalueof pairId bychangingitstype.
Longterm,carefullyreviewthecodebaseforexplicit typeconversions. Documentscenarios

in which these explicit type conversions result in an overflow or underflow of the result, and use Echidna to test for these scenarios throughout the codebase. hashey 44 Primitive Security Assessment PUBLIC

10. Error in Invariant.getX Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-HYPR-10 Target: Invariant.sol Description The getX function in the Invariant contract implements a formula that does not align with the formula specified in the whitepaper. $1 * @dev \text{Computes } x \text{ in } x = 1 - \Phi(\Phi^{-1}((y+k)/K) + \sigma\sqrt{\tau})$. Figure 10.1: The NatSpec comment for the getX() function in Invariant.sol The body of the function matches the formula described in the NatSpec comment of the function. However, the formula itself is derived incorrectly. The actual quantity to be used inside the parentheses should be $(y-k)$ instead of $(y+k)$. The formula is derived by rearranging the terms in the following: $y = K\Phi(\Phi^{-1}(1-x) - \sigma\sqrt{\tau}) + k$ By subtracting k from both sides of the equation, it becomes clear that the formula should read $(y-k)$. The actual impact of this error in the current implementation is low because the function is only ever called with the invariant $k=0$. Exploit Scenario In a future release of the protocol, Primitives decide to use the function with the parameter $k \neq 0$. This breaks the desired invariant after swaps occur. Recommendations Short term, correct the getX function's code and update the formula in the function's NatSpec comment. Long term, keep track of the derivations of formulas used throughout the codebase, and add fuzz tests that verify the properties of and assumptions about the functions that implement them. hashey 45 Primitive Security Assessment PUBLIC

11. Pools with overflowing maturity dates can be created Severity: Low Difficulty: High Type: Data Validation Finding ID: TOB-HYPR-11 Target: Hyper.sol Description Users could create pools with maturity date timestamps (which are of type uint32) that are too close to when uint32 timestamp overflows; pools with overflowing timestamps would become unusable.

```
326 function maturity(HyperCurv memory self) view returns (uint32 endTimestamp) {
327 return (Assembly.convertDaysToSeconds(self.duration) + self.createdAt).safeCastTo32();
328 }
```

 Figure 11.1: The maturity() function in HyperLib.sol Maturity dates are recurrently limited to five years in the future, and the date when uint32 timestamps will overflow is September 25, 2104. The maturity date is not validated on pool creation, so pools that will be unusable when the year 2104 approaches could be created. The maturity date is also not validated in the checkParameters() function of the HyperLib contract, which could allow an attacker to set the timestamp parameter of the pool to an overflowing timestamp. Exploit Scenario In a future version of the protocol, Primitives remove the five-year limit on pool maturity dates. Alice, the controller of a pool, decides to set the maturity date past the year 2104 and is able to trap all of the funds of the pool's liquidity providers. Recommendations Short term, add a check to the checkParameters() function in HyperLib to ensure that pools' maturity dates will not overflow. Long term, thoroughly document all of the assumptions that are made on the codebase's variables. Where types are limited in size, use modular arithmetic or a larger data type to handle any issues that could occur. hashey 46 Primitive Security Assessment PUBLIC

12. Minting funds to the Hyper contract arbitrarily increases the next caller's balance Severity: Informational Difficulty: High Type: Configuration Finding ID: TOB-HYPR-12 Target: Invariant.sol Description When a user mints funds to the Hyper contract, the contract relies on a calculation of the difference between its physical balance and its virtual balance of the given token. However, this calculation increases the Hyper contract's reserves and then the next caller's balance. To add tokens into the system, users call the fund function; this function uses the _settlement() function, which calls the settle() function. This function uses the getNetBalance() function, which calculates the difference between the return value of the token.balanceOf function and the Hyper contract's reserves of the token.

```
function getNetBalance(AccountSystem storage self, address token, address account) view returns (int256 net) {
uint256 internalBalance = self.reserves[token];
uint256 physicalBalance = __balanceOf__(token, account);
net = int256(physicalBalance) - int256(internalBalance);
}
```

 Figure 12.1: The getNetBalance() function in Hyper.sol Exploit Scenario Eve, an attacker, creates a DRP token. With her minting rights, she mints 1 million DRP to the Hyper contract. As a result, when Alice funds her account, the Hyper contract's reserve balance and Alice's tracked token balance increase by 1 million DRP, along with the tokens she intended to fund. Recommendations Short term, document the fact that the Hyper reserves and the respective user's balance for the airdropped token will be attributed to the next user. Long term, clearly identify the expected and unexpected flows in the contract to ensure that users are aware of expected behavior. hashey 47 Primitive Security Assessment PUBLIC

13. Pool strike price could be zero due to lack of lower bound check on maxTick Severity: High Difficulty: Low Type: Data Validation Finding ID: TOB-HYPR-13 Target: Hyper.sol Description The maxTick

variable is used to approximate the strike price of a pool. However, the code does not validate the lower bound of maxTick, which means that the strike price of a pool can be 0, causing the pool's asset to be mispriced. The maxTick variable is provided by pool creators and is validated on pool creation by the checkParameters function, which checks that the volatility, maxTick, duration, jit, and priorityFee values are within safe bounds. However, for the maxTick parameter, the validation function checks only its upper bound: `/**@dev Invalid parameters should revert.*/ function checkParameters(HyperCurve memory self) view returns (bool, bytes memory) { if (!Assembly.isBetween(self.volatility, MIN_VOLATILITY, MAX_VOLATILITY)) return (false, abi.encodeWithSelector(InvalidVolatility.selector, self.volatility)); if (!Assembly.isBetween(self.duration, MIN_DURATION, MAX_DURATION)) return (false, abi.encodeWithSelector(InvalidDuration.selector, self.duration)); if (self.maxTick >= MAX_TICK) return (false, abi.encodeWithSelector(InvalidTick.selector, self.maxTick)); // todo: fix, mintick check? if (self.jit > JUST_IN_TIME_MAX) return (false, abi.encodeWithSelector(InvalidJit.selector, self.jit)); if (!Assembly.isBetween(self.fee, MIN_FEE, MAX_FEE)) return (false, abi.encodeWithSelector(InvalidFee.selector, self.fee)); // @priorityfee = no controller, impossible to set to zero unless default from non-controlled pools. if (!Assembly.isBetween(self.priorityFee, 0, self.fee)) return (false, abi.encodeWithSelector(InvalidFee.selector, self.priorityFee)); return (true, ""); } Figure 13.1: The checkParameters() function in HyperLib.sol. The maxTick value is used to calculate prices, including the strike price of an asset: hasheye 48 PrimitiveSecurityAssessment PUBLIC`

```
function strike(HyperCurve memory self) view returns (uint) {
    return Price.computePriceWithTick(self.maxTick); } Figure 13.2: The strike() function in HyperLib.sol
However, because the lower bound of maxTick is not checked, the computePriceWithTick function could return a 0
value for the price, which would cause the system to use the incorrect value for strike prices. /**
 * @dev Computes a price value from a tick key. * @custom: math price = e^(ln(1.0001) * tick) *
 * @param tick Key of a slot in a price/liquidity grid.
 * @return price WAD value on a key (tick) value pair of a price grid. */
function computePriceWithTick(int24 tick) internal pure returns (uint256 price) {
    int256 tickWad = int256(tick) * int256(FixedPointMathLib.WAD);
    price = uint256(FixedPointMathLib.powWad(TICK_BASE, tickWad)); } Figure 13.3: The computePriceWithTick()
function in HyperLib.sol. Exploit Scenario: Alice creates a pool with a maxTick value of -887272. Upon calculating the strike price at maturity, the tickWad and price values are calculated as follows: 000
256 00000000 = -887272 * 1018 = -7.2019 000000 = 1_0001014 ^ (-7.2019) = 0 Recommendations
Short term, bound maxTick to a lower bound that will not allow strike prices to converge to 0, and have strike prices round up to ensure that they can never be 0.
Long term, clearly document the expected and unexpected flows in the contract to ensure that users are aware of expected behavior. hasheye 49 PrimitiveSecurityAssessment PUBLIC
```

14. Rounding error allows liquidity to be added without depositing tokens. Severity: High Difficulty: Low

Type: Data Validation Finding ID: TOB-HYPR-14 Target: HyperLib.sol Description: In pools that have an asset token of six decimals, small allocations of those tokens will be scaled by the number of decimals and then rounded down. A deltaAsset value of 0 can be returned by the getLiquidityDeltas function, even if a non-zero deltaLiquidity argument is provided, allowing an attacker to add liquidity without transferring any tokens.

```
function getLiquidityDeltas(HyperPool memory self, int128 deltaLiquidity) view returns (uint128 deltaAsset, uint128 deltaQuote) {
    if (deltaLiquidity == 0) return (deltaAsset, deltaQuote);
    (uint amountAsset, uint amountQuote) = self.getAmounts(); uint delta; if (deltaLiquidity > 0) {
        delta = uint128(deltaLiquidity); deltaAsset = amountAsset.mulWadUp(delta).safeCastTo128();
        deltaQuote = amountQuote.mulWadUp(delta).safeCastTo128(); } else { delta = uint128(-deltaLiquidity);
        deltaAsset = amountAsset.mulWadDown(delta).safeCastTo128();
        deltaQuote = amountQuote.mulWadDown(delta).safeCastTo128(); } }
/**@dev Decimal amounts per WAD of liquidity, rounded down...*/
function getAmounts(HyperPool memory self) view returns (uint amountAssetDec, uint amountQuoteDec) {
    (uint amountAssetWad, uint amountQuoteWad) = self.getAmountsWad();
    amountAssetDec = amountAssetWad.scaleFromWadDown(self.pair.decimalsAsset);
    amountQuoteDec = amountQuoteWad.scaleFromWadDown(self.pair.decimalsQuote); } Figure 14.1: The getLiquidityDeltas and getAmounts functions in HyperLib.sol. In the calculation of amountAssetDec in the getAmounts function, the amount of the asset token in wad units (1e18) is scaled to a value representative of that token's decimals and then rounded down. If amountAssetWad is a small value, amountAssetDec is hasheye 50 PrimitiveSecurityAssessment PUBLIC
```

rounded down to 0 and returned, tricking the system into thinking zero asset tokens are required to fulfill liquidity allocation. Exploit Scenario

Eve, an attacker, finds or creates a pool where the asset token has six decimals. She calls the `allocate` function and adds the smallest possible unit (1) as the amount to that pool. The `getAmounts` function scales the value to six decimals, rounds down, and returns 0 for `amountAssetDec`, which is then multiplied by `deltaLiquidity`; as a result, 0 is returned for the required `deltaAsset`. The parameters of the pool are changed and the `_increaseReserves` function is called with the correct `deltaQuote` value but 0 for the `deltaAsset` value. Recommendations Short term, make one of the following changes:

- Have amounts of token allocations rounded up to the nearest decimal unit depending on the token's assigned decimal value (e.g., for a token with six decimals, amounts should round up to the next token decimal point, which would be $1e12 = 1e(18-6)$).
- Add a zero-value check on the return values of `getAmounts`. Be careful to consider the downstream implications of any short-term fixes implemented for this issue, as the `getAmounts` function is used in critical system operations.

Long term, continue to add unit tests that consider the expected outcomes of a wide array of inputs and scenarios. Document all of the assumptions within the system and implement fuzz testing for them with Echidna in order to catch edge cases like this that might break assumptions that are not readily apparent. hashey 51 Primitive Security Assessment PUBLIC

15. Attackers can sandwich changeParameters call to steal funds Severity: High Difficulty: Low
 Type: Undefined Behavior Finding ID: TOB-HYPR-15 Target: HyperLib.sol Description The changeParameters function does not adjust the reserves according to the new parameters, which results in a discrepancy between the virtual reserves of a pool and the reserves of the tokens in the Hyper contract.

An attacker can create a new controlled pool with the token they want to steal and a fake token. They can then use a sequence of operations—`allocate`, `changeParameters`, and `unallocate`—to steal tokens from the shared reserves of the Hyper contract. In the worst-case scenario, an attacker can drain all of the funds from the Hyper contract. The Hyper contract does not store the reserves of tokens in a pool. The virtual reserves of the pool are recomputed with the last price of the given asset token and the curve parameters. Below, we discuss the impact that a change in parameters could have on various operations:

Allocating and Unallocating A user can add liquidity to or remove liquidity from a pool using the `allocate` and `unallocate` functions. Both of these functions use the `getLiquidityDeltas` function to compute the amount of the asset token and the amount of the quote token required to change the liquidity by the desired amount. The `getLiquidityDeltas` function calls the `getAmountsWad` function to compute the amount of reserves required for adding one unit of liquidity to the pool. The `getAmountsWad` function uses the last price of the asset token (`self.lastPrice`), `strikePrice`, `timeToMaturity`, and implied volatility to compute the amount of reserves per liquidity. `function getAmountsWad(HyperPool memory self) view returns (uint amountAssetWad, uint amountQuoteWad) { Price.RMM memory rmm = self.getRMM(); amountAssetWad = rmm.getXWithPrice(self.lastPrice); amountQuoteWad = rmm.getYWithX(amountAssetWad); }`

Figure 15.1: The `getAmountsWad` function in HyperLib.sol hashey 52 Primitive Security Assessment PUBLIC

When a controller calls the `changeParameters` function, the function updates the pool parameter values stored in the Hyper contract to the provided values. These new values are then used in the next execution of the `getAmountsWad` function along with the value of `pool.lastPrice`, which was computed with the previous pool parameters. If `getAmountsWad` uses the previous `pool.lastPrice` value with new pool parameters, it will return new values for the reserves of the pool; however, the Hyper contract's token balances will match those computed with the previous pool parameters. When a user calls the `unallocate` function after a pool's parameters have been updated, the new computed reserve amounts are used to transfer tokens to the user. Because the Hyper contract uses shared reserves of tokens for all of the pools, a user can still withdraw the tokens, allowing them to steal tokens from other pools.

Swapping The `swap` function computes the price of the asset token using the `_computeSyncedPrice` function. This function uses the pool parameters to compute the price of the asset token. It uses `pool.lastPrice` as the current price of the asset token, as shown in figure 15.2. It then calls the `computePriceChangeWithTime` function with the pool parameters and the time elapsed since the last swap operation. `function _computeSyncedPrice(uint64 poolId) internal view returns (uint256 price, int24 tick, uint updatedTau) { HyperPool memory pool = pools[poolId]; if (!pool.exists()) revert NonExistentPool(poolId); (price, tick, updatedTau) = (pool.lastPrice, pool.lastTick, pool.tau(_blockTimestamp())); uint passed = getTimePassed(poolId); if (passed > 0) { uint256 lastTau = pool.lastTau(); // pool.params.maturity() - pool.lastTimestamp. (price, tick) = pool.computePriceChangeWithTime(lastTau, passed); } }`

Figure 15.2: The `_computeSyncedPrice()` function in Hyper.sol The `computePriceChangeWithTime` function computes the strike price of the pool and then calls the `Price.computeWithChangeInTau` function to get the current price of the asset token. `function computePriceChangeWithTime(`

HyperPoolmemoryself, uinttimeRemaining, uintepsilon)purereturns(uintprice,int24tick){ hashey
53PrimitiveSecurityAssessment PUBLIC

```
uintmaxPrice=Price.computePriceWithTick(self.params.maxTick);  
price=Price.computePriceWithChangeInTau(maxPrice,self.params.volatility,  
self.lastPrice,timeRemaining,epsilon); tick=Price.computeTickWithPrice(price); } Figure15.3:The  
computePriceChangeWithTime() functionin HyperLib.sol The computePriceWithChangeInTau()  
usesaformulathatisderivedunderthe  
assumptionthattheimpliedvolatilityandstrikepriceofthepoolremainconstantduring  
theepsilonperiod.Thisepsilonperiodisthetimeelapsedsince-thelastswapoperation.  
Theproblem ariseswhenthecontrollerofthepoolchangesthepool'sparameters.The formulaisedinthe  
computePriceWithChangeInTau functionthenbecomesinvalidifthe epsilonperiodisgreaterthanzero.  
Ifauserswapstokensafterthecontrollerhasupdatedthecurveparameters,thenthe  
wrongpricecomputedbytheswapfunctionwillresultinunexpectedbehavior.Thisissue  
canbeusedbythecontrollerofthepooltoswapatadiscountedrate.  
Therearemultiplewaysinwhichanattackercouldusethisissueto steal fundsfromthe Hyper contract.  
ExploitScenario1 EvecreatesanewcontrolledpoolwithWETHasanassettokenandafaketokenasaquote  
token.Eveallocates1e18wadofliquidityinthepoolbydepositingXamountofWETH  
andYamountofthefaketoken.Evethendoublesthestrikepriceofthepoolbycalling changeParameters()  
.Thischangeinthestrikepricechangesthevirtualreservesofthe  
pool;specifically,itincreasesthenumberofassettokensanddecreasesthenumberof  
quotetokensrequiredtoallocate1e18wadofliquidity.Evethenunallocates1e18wad  
liquidityandwithdrawsX1amountofWETHandY1amountofthefaketoken.Thevalueof  
X1ishigherthanXbecauseofthechangeinthestrikeprice.ThisallowsEvetowithdraw moreWETHthanshedeposited.  
ExploitScenario2 Evecreatesacontrolledpoolfortwopopulartokens.Otherusersaddliquiditytothepool.  
Evethenchangesthepool'sparameterstochangethestrikepricetoavalueinherfavor  
andexecutesalargeswaptobenefitfromtheliquidityaddedtothepool.Theotherusers  
seeEve'sactionsasarbitrageandlosethevalueoftheirprovidedliquidity.Evehas  
effectivelyswappedtokensatadiscountedrateandstolenfundsfromthepool'sliquidity providers.  
Recommendations Shortterm,modifythe changeParameters functionsothatitcomputesnewtoken  
reserveamountsforpoolsbasedonupdatedpoolparametersandtransferstokensto  
fromthecontrollertoalignthe Hyper contractreserveswiththenewpoolreserves. hashey  
54PrimitiveSecurityAssessment PUBLIC
```

Longterm, carefullyreviewthecodebasetofindinstancesinwhichtheassumptionsused
informulasbecomeinvalidbecauseofuseractions;resolveissuesarisingfromtheuseof
invalidformulas.Addfuzzteststocapturesuchinstancesinthecodebase. hashey
55PrimitiveSecurityAssessment PUBLIC

```
16.Limitedprecisioninstrikepricesduetofixedtickspacing Severity:LowDifficulty:Low  
Type:DataValidationFindingID:T0B-HYPR-16 Target: Price.sol Description  
Thestrikepriceisallowedtotakeonvaluesonlyfromapredefinedsetofvalues.Thisisa  
fixedpricinggridwithlimitedprecision,whichmeansthatthestrikepricecandeviatefrom adesiredprice.  
ThestrikepriceinHyperissuppliedthroughan int24 tickvaluethatmapstoapriceinthe  
pricinggrid,whichwascomputedfromthetickbasevalueusingtheexponentialfunction ( price=TICK_BASE^tick  
).Asaresult,thepricevaluesarespacedoutexponentially fromeachother. /**  
*@devComputesapricevaluefromatickkey. * *@custom:mathprice=e^(ln(1.0001)*tick) *  
*@paramtickKeyofaslotinaprice/liquiditygrid.  
*@returnpriceWADValueonakey(tick)valuepaairofapricegrid. */  
functioncomputePriceWithTick(int24tick)internalpurereturns(uint256price){  
int256tickWad=int256(tick)*int256(FixedPointMathLib.WAD);  
price=uint256(FixedPointMathLib.powWad(TICK_BASE,tickWad)); } Figure16.1:The computePriceWithTick()  
functionin Price.sol Theriskofpricedeviationisevidentwhenlookingattheresultingvaluesfromonetickto  
another.Givenapriceintherangeof30,000perquotetoken(e.g.,BTC/USD),theprice  
differencefromoneticktoanotherisapproximately3USD.Thedifferencebetweenthe  
ticksgrowsexponentiallybyonepartperthousand.  
Furtherimprecisionscouldcompoundwhencomputingthenexttickfromthepriceaftera  
swap.However,thisisnotanexploitableissue,asthenexttickisnotactuallyusedto  
derivethenextpriceintheprotocol. hashey 56PrimitiveSecurityAssessment PUBLIC
```

ExploitScenario AliceopensapoolwithaUSD/BTCpairandwantstosetapriceof30,003USD/BTC.Dueto
thelimitedprecisionintheticks,thestrikepriceendsupbeingsetto30,000USD/BTC. Recommendations
Shortterm,documentwhetherthisisdesiredbehavioranddescribethelimitationsand
roundingissuessthatcouldresultfromit.

Longterm, considerwhetherafixedpricegridisnecessary;ifitisnot, considerusingthe
decimalsrepresentationforstoringprices. hashey 57PrimitiveSecurityAssessment PUBLIC

17.Functionsthatroundbyadding1resultinunexpectedbehavior

Severity:InformationalDifficulty:Undetermined Type:UndefinedBehaviorFindingID:TOB-HYPR-17 Target: Assembly.sol Description Unitconversionfunctionsinthe Assembly contractadd 1 totheresultsoftheirdivision operationstoroundthemup,whichresultsinunexpectedroundingeffects. The scaleFromWadUp() functionisusedtoconverttheinputforswapoperationsfroma wadunittoatokendecimalunit.Thefunctionalwaysadds 1 totheresultofthedivision operation,eveniftheinputamountwouldbeawholenumberintokendecimals.As a result,theuserwilltransfermoretokensthanexpected.

```
functionscaleFromWadUp(uintamountWad,uintdecimals)purereturns(uintoutputDec) {
    uintfactor=computeScalar(decimals); assembly{ outputDec:=add(div(amountWad,factor),1) } }
Figure17.1:The scaleFromWadUp() functionin Assembly.sol The scaleFromWadUpSigned() alsoadds 1
totheresultofthedivisionoperationto roundupthereturnvalue.
```

```
functionscaleFromWadUpSigned(intamountWad,uintdecimals)purereturns(int outputDec){
    uintfactor=computeScalar(decimals); assembly{ outputDec:=add(sdiv(amountWad,factor),1) } }
```

Figure17.2:The scaleFromWadUpSigned() functionin Assembly.sol Recommendations

Shortterm,usetheformula $(a-1)/b+1$ inthe scaleFromWadUp() and scaleFromWadUpSigned() functionstocomputetheroundedupresultofthedivision operation a/b . hasheyeye

58PrimitiveSecurityAssessment PUBLIC

Longterm,reviewtheentirecodebaseforfunctionsthatroundtoensurethattheydonot add 1 unconditionallytoresults.Addfuzzingtestcasestofindedgecaseshatcouldcause unexpectedroundingissues.

References • NumberLogic hasheyeye 59PrimitiveSecurityAssessment PUBLIC

18.Soliditycompileroptimizationscanbeproblematic Severity:InformationalDifficulty:Low

Type:UndefinedBehaviorFindingID:TOB-HYPR-18 Target: solstat/foundry.toml Description

TheHypercontractshaveenabledoptionalcompileroptimizationsinSolidity.

Therehavebeenseveraloptimizationbugswithsecurityimplications.Moreover, optimizationsareactivelybeingdeveloped.Soliditycompileroptimizationsaredisabledby default,anditisunclearhowmanycontractsinthewildactuallyusethe. Therefore,itis unclearhowwelltheyarebeingtestedandexercised. High-

severitysecurityissuesduetooptimizationbugshaveoccurredinthepast.A high-severitybuginthe emscripten -generated solc-js compilerusedbyTruffleand

Remixpersisteduntillate2018.ThefixforthisbugwasnotreportedintheSolidity CHANGELOG .Anotherhigh- severityoptimizationbugresultinginincorrectbitshiftresults

waspatchedinSolidity 0.5.6.Morerecently,anotherbugduetotheincorrectcachingof keccak256 wasreported. AcompilerauditofSolidityfromNovember2018concludedthattheoptionaloptimizations maynotbesafe.

Itislikelythattherearelatentbugsrelatedtooptimizationandthatnewbugswillbe introducedduetofutureoptimizations. ExploitScenario

AlatentorfuturebuginSoliditycompileroptimizations—orintheEmscriptentranspilation to solc-js — opensupasecurityvulnerabilityintheSolstatcontracts. Recommendations

Shortterm,measurethegassavingsfromoptimizationsandcarefullyweighthemagainst thepossibilityofanoptimization-relatedbug.

Longterm,monitorthedevelopmentandadoptionofSoliditycompileroptimizationsto assesstheirmaturity. hasheyeye 60PrimitiveSecurityAssessment PUBLIC

19.getAmountOutreturnsincorrectvaluewhencalledbycontroller Severity:LowDifficulty:Low

Type:DataValidationFindingID:TOB-HYPR-19 Target: HyperLib.sol Description

Whenacontrollercontractcallsthe getAmountOut function,the feeAmount shouldbe calculatedusingthe priorityFee value;however,thefunctionincorrectlyusesthe fee

value.Thiscausethefunctiontoreturnanincorrect output value. data.feeAmount= ((data.remainer>maxInput?maxInput:data.remainer)* self.params.fee)/10_000; Figure19.1:The getAmountOut functionin HyperLib.sol

Whenacontrollercontractswapstokens,thefeeassessedinthetransactioniscalculated usingthe priorityFee parameter. _state.fee=msg.sender==pool.controller?pool.params.priorityFee: uint(pool.params.fee);

Figure19.2:The _swap functionin Hyper.sol Thepurposeofthe getAmountOut

functionistoreturntheexpectedtokenamountthat theuserwouldreceiveafterexecutingaswap.Usingthewrong feeAmount willskewthe output calculation,causingadiscrepancybetweenwhattheuserexpectstoreceiveand whattheyactuallyreceiveafterexecutingtheswap. ExploitScenario

Alicewantstoswaptwotokensfromapoolthathasaccontrollercontractset.Shequeries thecontrollerwiththeproposedswapparameters,andthecontrollercontractcallsthe getAmountOut functionwiththesesameparameters.The getAmountOut functionreturns

anoutputamountoffivetokens.Alicesendsaswaptransactiontothecontroller,which callsthe swap functiononthe Hyper contract.OnlyfourtokensarereturnedtoAlice insteadoftheexpectedfive.

Recommendations Shortterm,refactorthe getAmountOut functiontoaccuratelymirrorcalculationsmadein the swap functionandtouse priorityFee whenacontrollercalls getAmountOut . hasheyeye

61PrimitiveSecurityAssessment PUBLIC

Longterm, expand the current unit tests suite to ensure that data returned by view functions is accurate and up to date. hashey 62 Primitive Security Assessment PUBLIC

20. Mismatched base unit comparison can inflate limit tolerance Severity: Medium Difficulty: Low
Type: Data Validation Finding ID: TOB-HYPR-20 Target: Hyper.sol Description
When a user swaps tokens, the code enforces the user's limitPrice value, denominated in the token's decimal units, by comparing it to the value of the pool's lastPrice value, denominated in wad units. The discrepancy between these units could prevent a user's intended price limit from being enforced, resulting in a swap at a market rate that the user did not intend to swap at. When a user calls the swap function, the limit argument is used in the _swapExactIn function as the limitPrice value (figure 20.1); the limitPrice value determines whether the user's intended limit price has been exceeded. The user's assumption is that this value is denominated in the quote token's decimal units. The limitPrice value is compared to the nextPrice value, taken from the pool's lastPrice value, and if the user's price limit has been met or exceeded, the swap reverts. `uint nextPrice = pools[args.poolId].lastPrice; if(!sellAsset && nextPrice > limitPrice) revert SwapLimitReached(); if(sellAsset && limitPrice > nextPrice) revert SwapLimitReached();` Figure 20.1: The _swapExactIn function in Hyper.sol The lastPrice variable is denominated in wad units, so it has 18 decimal places of precision. If the quote token used to denominate limitPrice has any fewer than 18 decimals, then it will always be smaller than intended compared to nextPrice. As a result, a swap transaction submitted by a user who has met or exceeded their price limit will not revert as expected.
Exploit Scenario Alice swaps tokens in a pool in which both tokens have six decimals. She sets a price limit of seven A tokens for one B token. The trade causes the pricing formula to swing to nine A tokens for one B token. The _swapExactIn function checks whether $7e6$ is greater than $9e18$, which it is not. No SwapLimitReached error is thrown as a result, and the swap is allowed to complete despite the surpassed price limit. hashey 63 Primitive Security Assessment PUBLIC

Recommendations Short term, modify the associated code so that either the limitPrice input value is scaled to wad units or the lastPrice value is scaled to however many decimal places the quote token has. Clearly document for users the denomination they should use for the price limit.
Long term, expand the current unit tests suite to consider token pools with all ranges of decimals; for each scenario, ensure that the swap function will revert when a price limit is reached. hashey 64 Primitive Security Assessment PUBLIC

21. Incorrect implementation of edge cases in getY function Severity: Low Difficulty: Low
Type: Undefined Behavior Finding ID: TOB-HYPR-21 Target: solstat/src/Invariant.sol Description The getY function in the Invariant contract deviates from the intended behavior when the value of R_x is equivalent to WAD and 0. `if(R_x == WAD) return uint256(int256(stk) + inv); // For `ppf(0)` case, because $1 - R_x = 0$, and $y = K * 1 + k$ simplifies to $y = K + k$ if(R_x == 0) return uint256(inv); // For `ppf(1)` case, because $1 - 0 = 1$, and $y = K * 0 + k$ simplifies to $y = k$.` Figure 21.1: The two incorrect if statements in the getY function in Invariant.sol The first if statement seems to be checking for the ppf(1) case, evinced by the comparison of $R_x == WAD$, with WAD representing one unit. The function should return the invariant in this case, but it returns the stk value summed with the invariant. $y = K * (0^{1-1} - \sigma/\tau) + k$
 $y = K * (0^{1-0} - \sigma/\tau) + k$ $y = K * (\text{negative infinite} - \sigma/\tau) + k$ $y = K * (\text{negative infinite}) + k$ $y = K * 0 + k$ $y = k$ Figure 21.2: The getY derivation from the white paper when ppf approaches negative infinity Similarly, when R_x is 0, the return value should be stk summed with the invariant, but it is simply the invariant. $y = K * (0^{1-1} - \sigma/\tau) + k$
 $y = K * (0^{1-1}) - \sigma/\tau + k$ $y = K * (\text{positive infinite} - \sigma/\tau) + k$ $y = K * (\text{positive infinite}) + k$ $y = K * 1 + k$ $y = K + k$ Figure 21.3: The getY derivation from the white paper when approaching positive infinity
Therefore, the return values for the two branches are incorrect. hashey 65 Primitive Security Assessment PUBLIC

Exploit Scenario Alice attempts to execute a swap, for which the R_x value is equivalent to 0. The function returns the invariant rather than stk summed with the invariant. This results in further miscalculations in the swaps.
Recommendations Short term, switch the return value statements in the two affected branches of the getY function: if R_x is WAD, the function should return the invariant, and if R_x is 0, the function should return stk summed with the invariant. Long term, thoroughly document all of the expected edge cases of inputs and check that these edge cases are handled. hashey 66 Primitive Security Assessment PUBLIC

22. Lack of proper bound handling for solstat functions Severity: Undetermined Difficulty: Low
Type: Undefined Behavior Finding ID: TOB-HYPR-22 Target: solstat/Gaussian.sol Description
The use of unchecked assembly across the system combined with a lack of data validation means that obscure bugs that are difficult to track down may be prevalent in the system. One example of unchecked assembly that could result in bugs is the getY function. Due to assembly rounding issues in the code base, the inverse of the function's two variables does not hold. This function uses targeted functions in the Gaussian code. We wrote a fuzz test and ran it on the getY function to ensure that the return values are monotonically

decreasing, the fuzzing campaign found cases in which the `getY` function returns a lower value than it should: `Logs: BoundResult99999998999999997 BoundResult10000000003 BoundResult10000000000 BoundResult10000000000 BoundResult86400 Error:a ≤ b not satisfied [uint] Valuea:10000000000 Valueb:99 Figure22.1:Resultsoffuzztestingthe getY function The solstat Gaussian contract contains the cumulative distribution function (cdf), which relies on the erfc function. The erfc function, however, has multiple issues, indicated by its many breaking invariants; for example, it is not monotonic, it returns hard-coded values outside of its input domain, it has inconsistent rounding directions, and it is missing overflow protection (further described below). This means that the cdf function's assumption that it always returns a maximum error of $1.2e-7$ may not hold under all conditions: /** @notice Approximation of the Cumulative Distribution Function. * @dev Equal to $D(x) = 0.5[1 + \text{erf}((x - \mu) / \sigma\sqrt{2})]$. * Only computes cdf of a distribution with $\mu = 0$ and $\sigma = 1$. hashey 67 PrimitiveSecurityAssessment PUBLIC * @custom: errorMaximumErrorof1.2e-7. * @custom: source https://mathworld.wolfram.com/NormalDistribution.html. */ function cdf(int256 x) internal pure returns (int256 z) { int256 negated; assembly { let res := sdiv(mul(x, ONE), SQRT2) negated := add(not(res), 1) } int256 erfc = erfc(negated); assembly { z := sdiv(mul(ONE, _erfc), TWO) } } Figure22.2: The cdf() function in Gaussian.sol The erfc function (and related functions) are used throughout the Hyper contract to compute the contract's reserves, prices, and the invariants. This function contains a few issues, described below in the figure. function erfc(int256 input) internal pure returns (int256 output) { uint256 z = abs(input); int256 t; int256 step; int256 k; assembly { let quo := sdiv(mul(z, ONE), TWO) let den := add(ONE, quo) t := sdiv(SCALAR_SQRD, den) // [...] } } Figure22.3: The erfc() function in Gaussian.sol Lack of Overflow Checks The erfc function does not contain overflow checks. In the above assembly block, the first multiplication operation does not check for overflow. Operations performed in an assembly block use unchecked arithmetic by default. If z, the absolute value of the input, is larger than $\text{type}(\text{int256}).\text{max} / 1e18$ (rounded up), the multiplication operation will result in an overflow. Use of sdiv instead of div Additionally, the erfc function uses the sdiv function instead of the div function on the result of the multiplication operation. The div function should be used instead because z and the product are positive. Even if the result of the previous multiplication operation has a hashey 68 PrimitiveSecurityAssessment PUBLIC`

does not overflow, if the result is larger than `type(int256).max`, then it will be incorrectly interpreted as a negative number due to the use of `sdiv`. Because of these issues, the output values could lie well beyond the intended output domain of the function, `[0, 2]`. For example, `erfc(x) ≈ 1e57` is a possible output value. Other functions—and those that rely on `erfc`, such as `pdf`, `ierfc`, `cdf`, `ppf`, `getX`, and `getY`—are similarly affected and could produce unexpected results. Some of these issues are further outlined in appendix E. Due to the high complexity and use of the function throughout this codebase, the exact implications of an incorrect bound on the function are unclear. We specify further areas that require investigation in appendix C; however, primitives should conduct additional analysis on the precision loss and specificity of `solstat` functions. `ExploitScenario` An attacker sees that under a certain swap configuration, the output amount in `Hyper`'s `swap` function will result in a significant advantage for the attacker. `Recommendations` Short term, rewrite all of the affected code in high-level Solidity with native overflow protection enabled. Long term, set up sufficient invariant tests using `Echidna` that can detect these issues in the code. For all functions, perform thorough analysis on the valid input range, document all assumptions, and ensure that all functions revert if the assumptions on the inputs do not hold. `hashey 69 PrimitiveSecurityAssessment PUBLIC`

23. Attackers can steal funds by swapping in both directions `Severity: High Difficulty: Low Type: Undefined Behavior Finding ID: TOB-HYPR-23 Target: contracts/Hyper.sol Description` Unexpected behavior in the `swap` function allows users to profit when executing a swap in one direction and then executing the same swap in the other direction. In a fuzz test, we identified a case in which an attacker is able to create a pool with a certain configuration that allows them to swap in 1 wei of the asset token and then to swap back out a high number of asset tokens. This would allow the attacker to drain the pool. This issue was found toward the end of the audit, so we were unable to locate the root cause. This is the path taken: `Swapping: 1 asset → 1 quote → 10000000001 asset CreatingPool: controller0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496 priorityFee1 fee1 volatility100 duration272 jit0 stk3 price3 Allocating liquidity: 171859515069719386357 Selling, then buying asset Swapping: 1 asset → 1 quote → 10000000001 asset SWAPdir0(selling): asset → quote SWAPdir1(buying): quote → asset balasset10000000000 balquote0 Figure23.1: The swap output depicting unexpected behavior Recommendations Short term, analyze the swap function to identify the root cause of the vulnerability. This`

issue still persists after fixing overflow issues and unit conversions by replacing assembly code with high-level code. hashey 70 Primitive Security Assessment PUBLIC

Long term, add fuzz test cases to find edge cases causing issues with unexpected rounding behavior. hashey 71 Primitive Security Assessment PUBLIC

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	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	As 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

hashey 72 Primitive Security Assessment PUBLIC

Severity Levels

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.

hashey 73 Primitive Security Assessment PUBLIC

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 code-based documentation
Transaction Reordering Risks	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	Are 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.

hashey 74 Primitive Security Assessment PUBLIC

hashey 75 Primitive Security Assessment PUBLIC

C. Rounding Recommendations Primitive uses fixed-point arithmetic. The current strategy has resulted in incorrect rounding, allowing an attacker to benefit from dust and misprices and to steal assets (TOB-HYPR-3, TOB-HYPR-4, TOB-HYPR-5, TOB-HYPR-6, TOB-HYPR-7, TOB-HYPR-14, TOB-HYPR-16, TOB-HYPR-19, TOB-HYPR-20, TOB-HYPR-22, and TOB-HYPR-23). These issues point to a greater need to test the system in more depth. We recommend ensuring that rounding directions always benefit the pool. Determining Rounding Directions To determine how to apply rounding (whether up or down), consider the result of the expected output. For example, the formula for a swap of token x for token y calculates how much of token x must be sent to the contract to receive y. $y' = \frac{y}{\psi} (\psi - 1 (1 - \sigma)) + \frac{y}{\psi}$ In order to benefit the pool, y' must tend toward a lower value to minimize the amount paid out. As a result, the following should hold:

- must round $\frac{y}{\psi} (\psi - 1 (1 - \sigma))$
- must round $\frac{y}{\psi} \circ \sigma \tau$
- must round $\frac{y}{\psi}$

Therefore, the mathematics in the formula should perform this check: $y' = \frac{y}{\psi} (\psi - 1 (1 - \sigma \tau)) + \frac{y}{\psi}$ Similar rounding techniques can be applied in all of the system's formulas to ensure that rounding always occurs in the direction that benefits Primitive. Rounding Results

- When funds leave the pool, these values should round down to favor the protocol over the user. Rounding these values up allows an attacker to profit from the rounding direction by receiving more than intended on pool interactions.
- When funds enter the pool, these values should always round up to maximize the number of tokens a pool receives. Rounding these values down can result in

near-zero values, which allow an attacker to profit from the rounding direction by receiving heavily discounted funds. Rounding down to zero can allow an attacker to steal funds.

- When fees are recalculated, the amount attributed to the fee bucket should always round up to maximize the amount the protocol receives. Rounding down means residual dust may be sent to users instead of the protocol.

Recommendations for Further Investigation

- Analyze all instances in which invariantson variable approximations are used, because after rounding and scaling, the original state of the unscaled variable may not be correctly bounded.
- Implement thorough happy and unhappy path testing throughout the codebase.

D. Risks with Arbitrary Tokens and Third-Party Controllers Primitive aims to allow third-party users to create their own token pairs and pools. These user-created pairs and pools could introduce problems that could allow an attacker to steal funds. We recommend that users review the tokens and vet third-party controllers to ensure that pools do not behave unexpectedly.

Ensure that users follow these guidelines when creating token pairs and pools:

- Pools should never be upgradeable. Upgradeable pools have inherent risks that may not be apparent with different versions.
- Tokens should not have a self-destruct capability. Destructible tokens have inherent risks, including malicious upgrades through create2.
- Users should not be able to change token decimals. Adjusting a token's decimals to either less than six or greater than 18 will break the token's composability with the arithmetic in the pool. An example is shown in figure D.1.

```
pair_decimals_never_exceed_bounds(uint256):failed! Call sequence:
create_pair_with_safe_preconditions(1,0) setDecimals(0) pair_decimals_never_exceed_bounds(0)
```

Figure D.1: An Echidna failure on a pair whose token decimals changed after creation

- Tokens should not be interest-bearing or re-adjusting. The formulaic derivation for the AMM relies on a risk-free rate of return for the asset token. Any form of a wrapper token that pays fees to liquidity providers poses risks to the codebase.

E. Recommendations for Overflow and Underflow Analysis in Assembly Blocks In this appendix, we provide recommendations for improving the assembly blocks in the Primitive Hypercodebase. Operations in assembly blocks can be problematic if the code does not check for overflows or underflows, and if certain assumptions about the operations' inputs are not documented or checked.

```
function pdf(int256 x) internal pure returns (int256 z) { int256 e; assembly {
e := sdiv(mul(add(not(x), 1), x), TWO) // (-x*x)/2. } e = FixedPointMathLib.expWad(e); assembly {
z := sdiv(mul(e, ONE), SQRT_2PI) } }
```

Figure E.1: The pdf() function in Gaussian.sol

For example, the negation operation in the code above (using mul, not, and add), does not check whether x equals type(int256).min, a possible input to the function. Calling the function with type(int256).min would cause an overflow in the addition operation, preventing the result from being negated. Additionally, the multiplication operation (-x*x) could underflow if the result is less than type(int256).min. In the next assembly block in figure E.1, sdiv is used, where div would be appropriate. This solmate function expWad's maximum output is such that it could be multiplied by one wad(1e18) without overflowing in int256. Using sdiv instead of div implies that the result is interpreted as a signed integer. However, expWad always outputs positive numbers; therefore, div

should be used instead. Because of the way solidity restricts `expWad` output, overflow is not an issue in this case. Nonetheless, multiplying `e` by anything larger than `1e18` could result in an overflow, causing `sdiv` to misinterpret an unsigned value as a signed value. These assumptions must be carefully checked and documented when using inline assembly. Overflow and underflow checks can be omitted via unchecked blocks where appropriate analysis is performed and heavy optimization is required. An example of such analysis is `hashey` 79 Primitive Security Assessment PUBLIC

shown in figure E.2. However, we strongly recommend that overflow checks always be included because it can become hard to keep track of the assumptions when the code evolves. `function pdf_checks_overflow(int256 x) internal pure returns (int256 z) { uint256 absX = abs(x); // Reverts for `x = type(int256).min`. uint256 xSquared = absX * absX; // Overflow check in uint256 is required. unchecked { // We can safely cast the result of the division to int256, since // dividing `xSquared` by `2e18` ensures that the result is less than `type(int256).max`. // The result is positive, which means that a check for `type(int256).min` // can be omitted when negating the result. x = -int256(xSquared / 2e18); } int256 e = FixedPointMathLib.expWad(k); unchecked { // The output of `expWad` is such that it can be safely // multiplied by `1e18` without causing an overflow in int256. z = e * ONE / Sqrt_2PI; } }` Figure E.2: An example `pdf_checks_overflow` function with unchecked blocks that contains enough analysis to justify omitting overflow checks

Although overflow checks and appropriate analysis in assembly blocks can improve otherwise unchecked code, a better practice is to use high-level Solidity versions instead, as exemplified in figure E.3. Using high-level Solidity would improve the given function's protection against overflow and underflow (when native overflow and underflow protection is enabled under `pragma ^0.8.0`) and improve the auditability of the code. We recommend that Primitive consider using the higher-level implementations of functions to allow for the use of native in-built protection.

`function erfc_checks_overflow(int256 input) internal pure returns (int256 output) { uint256 z = abs(input); // Reverts for `x = type(int256).min`. // We can safely cast the result of the division to int256, // because it is positive and less than `type(int256).max`. int256 t = int256(1e36 / (1e18 + z / 2)); int256 step = ERFC_J; step = ERFC_I + (t * step / 1e18); step = ERFC_H + (t * step / 1e18); step = ERFC_G + (t * step / 1e18); step = ERFC_F + (t * step / 1e18); step = ERFC_E + (t * step / 1e18); step = ERFC_D + (t * step / 1e18); step = ERFC_C + (t * step / 1e18); hashey 80 Primitive Security Assessment PUBLIC`

`step = ERFC_B + (t * step / 1e18); step = -ERFC_A + (t * step / 1e18); // We can safely cast the result of the division to int256, // because it is positive and less than `type(int256).max`, // if the multiplication does not revert due to overflow. int256 k = step - int256(z * z / 1e18); int256 expWad = FixedPointMathLib.expWad(k); int256 r = t * expWad / 1e18; output = input < 0 ? TWO - r : r; }` Figure E.3: An example `erfc_checks_overflow` function in high-level Solidity `hashey` 81 Primitive Security Assessment PUBLIC

F. Staking Issues Although the staking-related code was considered out of scope for this audit, we gave these contracts a best-effort review and identified the following issues. Because they were not thoroughly investigated, we recommend that Primitive check the following areas: • When a swap occurs, the priority fee amount is computed for the total pool liquidity and transferred from the controller. This means that an attacker may be able to profit from executing a swap against the total pool liquidity. • The `unstakeTimestamp` value is not updated after pool parameters are updated. This may result in undesired behavior. • In the `claim` and `_changeStake` functions, every liquidity provider gets a fee payment for the total staked liquidity. It is unclear whether this behavior is intended; if it is, it should be thoroughly and clearly documented. • Users cannot withdraw liquidity after unstaking due to the JIT restriction. `hashey` 82 Primitive Security Assessment PUBLIC

G. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future. • Use consistent naming conventions throughout the codebase. Choose conventions for naming variables and use those conventions consistently throughout the codebase. Figure 6.1 shows an example of variables that use leading underscores and one that does not. `uint256 privateLocked = 1; Payment[] privatePayments; SwapState privateState;` Figure 6.1: private variable declarations in `Hyper.sol` • Beware of potential overflows in assembly blocks. Information contained in an

arbitrarily large bytes array could overflow when loaded into memory.

```
function toBytes32(bytes memory raw) pure returns (bytes32 data) { assembly { data := mload(add(raw, 32))
let shift := mul(sub(32, mload(raw)), 8) data := shr(shift, data) } }
Figure 6.2: The toBytes32 function in Assembly.sol
```

- Ensure that variables' higher-order bits are cleared before they are accessed in assembly blocks if their types are less than 256 bits. Accessing variables of types that are less than 256 bits in assembly blocks does not guarantee that higher-order bits will be zeroed out. See the Solidity documentation for more details on this issue.

```
function addSignedDelta(uint128 input, int128 delta) pure returns (uint128 output) {
bytes memory revertData = abi.encodeWithSelector(InvalidLiquidity.selector); assembly {
switchslt(delta, 0) // delta < 0 ? 1:0 // negative delta case 1 { output := sub(input, add(not(delta), 1))
switchslt(output, input) // output < input ? 1:0 case 0 { // not less than
revert(add(32, revertData), mload(revertData)) // 0x1fff9681 } hashey 83 PrimitiveSecurityAssessment
PUBLIC
```

```
} // position delta case 0 { output := add(input, delta) switchslt(output, input) // (output < input ? 1:0) == 0 ?
1:0 case 1 { // less than revert(add(32, revertData), mload(revertData)) // 0x1fff9681 } } }
```

Figure 6.3: The addSignedDelta function in Assembly.sol

- Avoid casting down inputs. Accepting inputs of one type only to cast them to a different type in the code can be confusing to end users and can make it difficult to reason about how the system will behave.

```
function allocate(uint64 poolId, uint amount) external lock interactions returns (uint deltaAsset, uint deltaQuote) {
bool useMax = amount == type(uint).max;
(deltaAsset, deltaQuote) = _allocate(useMax, poolId, (useMax ? 1: amount).safeCastTo128()); }
Figure 6.4: The allocate function in Hyper.sol
```

- Standardize the use of uint and uint256 throughout the code. uint is an alias of uint256, and the two can be used interchangeably without altering the underlying type. However, it is best practice to commit to using one or the other throughout a codebase.
- Follow consistent conventions for outputs returned in tuples. Doing so can improve the codebase's readability. For example, the code in figure 6.5 returns the quote amount first followed by the asset amount; however, the code in figure 6.6 outputs them in reverse.

```
function computeReserves(RMM memory r, uint price) internal pure returns (uint R_y, uint R_x) {
Figure 6.5: The computeReserves function from Price.sol returns the quote amount and then the asset amount.
function getLiquidityDeltas(HyperPool memory self, int128 deltaLiquidity) hashey 84 PrimitiveSecurityAssessment
PUBLIC
```

```
) view returns (uint128 deltaAsset, uint128 deltaQuote) {
Figure 6.6: The getLiquidityDeltas function from HyperLib.sol returns the asset amount and then the quote amount.
```

- Revise certain error messages that lack detail. In some parts of the codebase, calculations and rounding could trigger underflow and divide-by-zero reverts that do not return helpful error messages. End users could be confused about why exactly their apparently valid transactions have not succeeded. An example of this issue can be found in calculations made by the computePriceWithChangeInTau function of the Price library.

```
hashey 85 PrimitiveSecurityAssessment PUBLIC
```