

Panoptic

Security assessment by HashEye · prepared for Panoptic

HASHEYE AUDITED

PROJECT	Panoptic
CLIENT	Panoptic
CATEGORY	Blockchain
PUBLISHED	May 1, 2024
REPORT ID	research-panoptic-2024-05-01-1x7z3o

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-panoptic-2024-05-01-1x7z3o.

PanopticLiquidationEngine InvariantDevelopment June4,2024 Preparedfor: GuillaumeLambert Panoptic
Preparedby:GuillermoLarregayandNatChin

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. 497 Carroll St., Space 71, Seventh Floor Brooklyn, NY 11215 <https://www.hashey.io> info@hashey.io hashey 1PanopticLiquidationEngineInvariantDevelopment PUBLIC

Notices and Remarks Copyright and Distribution ©2024 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 Panoptic under the terms of the project statement of work and has been made public at Panoptic'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 is the canonical source for hashey publications; 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

2PanopticLiquidationEngineInvariantDevelopment PUBLIC

Table of Contents About hashey 1 Notices and Remarks 2 Table of Contents 3 Project Summary 4 Executive Summary 5 Project Goals 6 Project Targets 7 Project Coverage 8 Automated Testing 10 Summary of Invariants 10 Harness Setup 11 System-Wide Invariants 11 Collateral Invariants 12 Burning Invariants 12 Minting Invariants 12 Liquidation Invariants 13 B. Documentation Improvement Recommendations 14 C. Testing Improvement Recommendations 15 D. Fuzzing Harness Design 20 Free versus Guided Options Creation 21 Long and Short Options Minting 21 Sequence Length 21 On-Chain versus Off-Chain Fuzzing 21 Single versus Multiple Entry Points 21 EOA versus Contract Actors 22 hashey
3PanopticLiquidationEngineInvariantDevelopment PUBLIC

Project Summary Contact Information The following project manager was associated with this project: Mary O'Brien, Project Manager mary.obrien@hashey.io

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain josselin.feist@hashey.io

The following consultants were associated with this project:

GuillermoLarregay, ConsultantNatChin, Consultant guillermo.larregay@hashey.eionat.chin@hashey.io
ProjectTimeline Thesignificanteventsandmilestonesoftheprojectarelistedbelow. DateEvent
March28,2024Pre-projectkickoffcall April8,2024Statusupdatemeeting#1
April16,2024Statusupdatemeeting#2 April22,2024Statusupdatemeeting#3
April29,2024Statusupdatemeeting#4 May10,2024Deliveryofinitialreportdraft
May10,2024Finalreadoutmeeting June4,2024Deliveryofcomprehensivereport hashey
4PanopticLiquidationEngineInvariantDevelopment PUBLIC

ExecutiveSummary EngagementOverview

Panopticengagedhasheyeforaninvariantdevelopmentandtestingexercisefor
Panopticliquidationengine.ThePanopticprotocolenables theminting, trading, and market-
makingofperpetualputandcalloptions, leveragingUniswap'sliquiditytoenhance
theuserexperienceforoptionstraders.

AteamoftwoconsultantsconductedtheexercisefromApril1toMay10,2024,foratotal offiveengineer-
weeksoffort.Ourtestingeffortsfocusedoncreatingatestharnessthat
couldsetuptheprotocol,andtestbasicfunctionssuchasmintingandburningoptionsand
strategies, simulatingpricefluctuations, andtriggeringliquidationsforundercollateralized
positions.Withfullaccesstosourcecodeanddocumentation,weidentifiedandwrote
invariantsofthesystemandranthemwithEchidna.

Thedeliverablefromthisinvariantdevelopmentandtestingexerciseincludesastateful
fuzztestingsuitetotesttheinvariantswedeveloped, coveringtheuserflowsmentioned
intheProjectCoverageofthisreport, andthisreport, whichincludesasummaryof
theinvariantswewrote, thesecurityfindings thatresultedfromourtesting,
recommendationsforwritingfutureinvariantsandexpandingthefuzztestingsuite, and otherinsights.

Recommendations Basedonthebaselinecodebaseandfindingsidentifiedduringthetesting
exercise, hasheyerecommends thatPanoptictakethefollowingsteps:

- Improveontheprovidedfuzzingharness.Theprovidedharnessisnotacomplete
testforthewholesystem; rather, itismeanttobeusedasareferenceforadding
moresystemandfunctionalinvariantsinthefuture, increasingthetestcoveragefor
theremainingfunctionsinthecodebase.

- Improvethe technicaldocumentation.UsersanddevelopersinthePanoptic
ecosystemwillrelyonupdateddocumentationtointeractwiththeprotocoland
makeeducateddecisionsabouttheirassetsondevelopment.Havinganupdated
documentationmakesiteasierforthemtocatchupwiththeprotocol.Seeappendix Bformoreinformation.

- Improvethetestsuite.Currently,thetestsuiteisdifficulttoreadandunderstand.

We providesomerecommendationstoimprovetests inappendixC. hashey

5PanopticLiquidationEngineInvariantDevelopment PUBLIC

ProjectGoals TheengagementwasscopedtowriteandtestinvariantsofthePanopticoptionsminting,
burningandliquidationmechanisms.Specifically, weusedthefollowingnon-exhaustivelist
ofquestionstoguideourdevelopment:

- Canactorsmintoptionsthatcannotbeburnedorliquidated?

- Isitpossibletoincorrectlyliquidateacollateralizedposition?

- Doactorsgetcompensatedcorrectlywhenburningorliquidatingpositions?

- Aretheactors' assetscorrectlytrackedtoavoidlosses?

- Arethesystemlimitationsstrong, andcantheyhandlemaliciousactors?

hashey
6PanopticLiquidationEngineInvariantDevelopment PUBLIC

ProjectTargets Theengagementinvolvedtestingofthefollowingtarget. panoptic-v1-core-private
Repository<https://github.com/panoptic-labs/panoptic-v1-core-private> Initial f59ed0f TypeSolidity
PlatformEthereum hashey 7PanopticLiquidationEngineInvariantDevelopment PUBLIC

ProjectCoverage Thissectionprovidesanoverviewoftheanalysiscoverageoftheproject, asdeterminedby
ourhigh-levelengagementgoals.Ourapproachesincludedthefollowing:

- UniswapV3pools:ThefirstapproachallowedustodeployPanopticprotocolinthe
testharnessusingamainnetfork.Echidnasupportsforkingnatively, soitserverdas

aquickwayofhavingaworkingsystem.Later, afterwediscoveredthatfuzzing on-

chaincontractsisslowandputsahaavyloadontheRPCkey, wecreatedalocal

deploymentoftheUniswapPoolandRoutercontracts, withalloftheinitializations

required.Wedidnottestthepoolcontractitself, butitisarequirementfor deployingPanoptic.

- PanopticPool

:Thecorelogiccontractthathandlestheminting, burning,

exercising, andliquidationofoptions.Publiclycallablefunctionsweretestedwith

severalactorsthatssimulatedifferentusersofthesystem, checkingtherequired

preconditionsandpostconditionsfortheflowsundertest.Invariantswerewritten

forminting, burning, andliquidatingpositions.

- CollateralTracker

:Contractsthatandletheuserscollateralforinteractingwith

theprotocol.Actorscandepositandwithdrawcollateralgivencertainpreconditions

that are checked in the harness. The user flows that were covered in the tests are:

- Minting of single-leg options. Multiple actors can mint single-leg options with fuzzed parameters that specify which asset to use to mint the option, whether it is a call or a put option, and whether it is a long or a short. The strike price, width and position size are also fuzzed inside a range.
- Minting of strategies. An additional minting function was created to mint strategies that require specific conditions or risk partnering between legs. We have implemented the Strangle and Straddle strategies, but the function can be extended to include other strategies.
- Burning of options. Actors can burn their own positions, either as a specific one or all of them in a single call.
- Liquidation of options. Actors can try to liquidate other actors' accounts, if they are undercollateralized. We tested two approaches for this: one involved a manipulation of the user collateral, and the other performing swaps in the underlying pool to move the price to a certain value. Since the first approach required interacting with the protocol in ways that are not realistic, we decided to leave the pool price manipulation as the default strategy for liquidations.

hasheyeye
8PanopticLiquidationEngineInvariantDevelopment PUBLIC

It is assumed that users can have access to a virtually unlimited supply of both tokens in the pool.

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:

- Operation with multiple-leg positions. Even though a function to mint multi-leg positions was implemented and provided in the code, it was not sufficiently tested. It is advised for the team to perform the required tests to ensure that the minted options are correct, and within the system limitations.
- Insufficient test executions for liquidations. For each test run, Echidna generates random sequences of calls. In order for the Panoptics system to be in a state where users are refunded and positions are minted and later liquidated, certain calls should happen in the correct sequence. Given that the sequence length used was of 500 calls per test run, it is difficult to hit more than two consecutive liquidations in the same sequence. This can be compensated by increasing the number of test runs, but this solution will not cover cases where an issue can occur after a certain number of positions have been liquidated.
- Insufficient test executions for minting. In a similar position as the previous point, the number of positions minted per actor in a sequence can be low. This is due to the relatively high number of factors present in the system, and the prerequisites that must be met to mint a new option. This affects certain system assumptions that will probably never be met in the test runs.
- Exercise of options was not covered. Our focus was on the minting, burning, and liquidation flows, so there is room for expanding the test suite to include invariants for the exercising or force-exercising of options.

hasheyeye
9PanopticLiquidationEngineInvariantDevelopment PUBLIC

Automated Testing hasheyeye uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in-house, to perform automated testing of source code and compiled software.

Test Harness Configuration We used the following tools in the automated testing phase of this project:

- Tool Description Policy
- Echidna A smart contract fuzzer that can rapidly test security properties via malicious, coverage-guided test case generation
- Local: Short runs for testing.
- Cloud: continuous running.

Summary of Invariants The table below summarizes the number and type of invariants we ran for each component.

Component	Invariant Type	Total Number
Harness Setup	System Invariants	7
System	System Invariants	4
Collaterals	Functional Invariants	4
Burning	Functional Invariants	4
Minting	Functional Invariants	4
Liquidation	Functional Invariants	6
Total	Invariants	29

hasheyeye
10PanopticLiquidationEngineInvariantDevelopment PUBLIC

Harness Setup To deploy Panoptic, we implemented a setup harness that performs several checks to ensure that the protocol is correctly deployed.

IDPropertyResult PAN-DEP-01Deployment of a working UniSwap V3 pool and router. Passed PAN-DEP-02Deployment of a Panoptic Pool should point to a local pool with the same parameters as a live pool. Passed PAN-DEP-03Depositing tokens to the Collateral Trackers should succeed. Passed PAN-DEP-04Minting arbitrary-sized options should succeed. Passed PAN-DEP-05Premium should be accumulating via arbitrary swaps in the pool. Passed PAN-DEP-06Liquidating user's existing options via collateral drop must succeed. This is currently not used in the other liquidation tests. Passed PAN-DEP-07Liquidating user's existing options via pool price variation must succeed. Passed System-Wide Invariants This group tests the global system state consistency over user transactions.

IDPropertyResult PAN-SYS-01The maximum collateral withdrawal amount of users with open positions is zero. Passed PAN-SYS-02Actors cannot withdraw collateral when having open positions. Passed PAN-SYS-

03Actorscan'thaveanopenpositionwithnocollateral.Passed hashey
11PanopticLiquidationEngineInvariantDevelopment PUBLIC

PAN-SYS-04Actor'sowedpremiaisnotlessthantheavailablepremia.Passed CollateralInvariants
Thisgroupofinvariantsisrelatedtothewaythatactorsinteractwiththecollateral trackers.
IDPropertyResult PAN-COL-01ThePanopticpoolbalancemustincreasebythedeveloped
amountwhenadepositismade. Passed PAN-COL-02Theuserbalancemustdecreasebythedevelopedamount
whenadepositismade. Passed PAN-COL-03ThePanopticpoolbalancemustdecreasebythe
withdrawnamountwhenawithdrawalismade. Passed PAN-COL-
04Theuserbalancemustincreasebythewithdrawnamount whenawithdrawalismade. Passed BurningInvariants
Thisgroupofinvariantsisrelatedtotheburningprocessofoptions. IDPropertyResult PAN-BUR-
01Zerosizedpositionscannotbeburned.Passed PAN-BUR-
02Forashortposition,thecurrentliquiditymustbegreater thantheliquidityinthechunkoftheposition.
Passed PAN-BUR-03Afterburningalloptions,thenumberofpositionsofthe actormustbezero. Passed PAN-BUR-
04Burningasingleoptiondecreasestheamountofopen positionsfortheactor. Passed MintingInvariants
Thisgroupofinvariantsisrelatedtothemintingprocessofoptions. hashey
12PanopticLiquidationEngineInvariantDevelopment PUBLIC

IDPropertyResult PAN-BUR-01Forlongpositions,theeffectiveliquidityfactormustbe
lowerthanorequaltotheliquiditylimit. Passed PAN-BUR-
02Thepositionbalanceforthemintedpositionmustequal thepositionsize. Passed PAN-BUR-
03Userscannot havemorethan32simultaneouspositions opened. Passed PAN-BUR-
04Thepositioncountermustincreaseafterasuccessful mint. Passed LiquidationInvariants
Thisgroupofinvariantsisrelatedtothemintingprocessofoptions. IDPropertyResult PAN-LIQ-
01Theactormustbeundercollateralizedfortheaccountto beliquidated. Passed PAN-LIQ-
02Liquidationclosesalloftheactor'spositions.Passed PAN-LIQ-
03Theliquidatedactorisdebitedthecorrectbonusvalue.Passed.Needs further testing. PAN-LIQ-
04Theliquidatorreceivesthecorrectbonusvalue.Passed.Needs further testing. PAN-LIQ-
05Thehaircutpremiaamountiscorrect.Passed.Needs further testing. PAN-LIQ-
06Whenthereisprotocolloss,thepremiaiscorrectly haircut. Passed.Needs further testing. hashey
13PanopticLiquidationEngineInvariantDevelopment PUBLIC

B. DocumentationImprovementRecommendations

ThisappendixprovidesanoverviewofthecurrentdocumentationforthePanoptic
protocolandsuggestsimprovementsstoenhanceitsusability, readability, and
comprehension. Comprehensiveandengaginglearningmaterialwillhelpusersinteract
withthesysteminaninformedwayandwillhelpfutureintegrators, reviewers, and
auditorsquicklyunderstandthesystem'sbusinesslogic.
Currently,thereisadocumentationsection,anewsblog,anda"deepdive"sectionof
articlesonPanoptic'swebsite,explainingtheprotocol'sfeatureswithexamples, images, andstep-by-
stepguides. Additionally,thecodeiswell-documentedintermsofNatSpecand general-
purposecomments.Despitethepresenceofdocumentation,thereisroomfor
improvementtocatertoabroaderaudienceandtomaketheinformationmoreaccessible. SummaryofProposedChanges
ToenhancethePanopticprotocoldocumentation,weproposethefollowing improvements: •
Updatethedocumentationinthewebsitetomatchthecurrentversionofthe
development. Duringtheengagement,wefoundthatseveraltechnical
documentationpageswereunderconstruction(example). Additionally, otherpages
referencedremovedprotocolfeatures, out-of-datefunctions(example), contracts
(example), orflowsthatwerereplacedorupgraded(example). •
Adddocumentationtotests. Mosttestcaseswereinsufficientlydocumentedor notdocumentedatall. hashey
14PanopticLiquidationEngineInvariantDevelopment PUBLIC

C. TestingImprovementRecommendations

Thisappendixaimstoprovidegeneralrecommendationsonimprovingprocessesand
enhancingthequalityofthePanoptictestsuite. IdentifiedTestingDeficiencies
Duringthereview,weidentifiedseveraldeficienciesinthetestsuitethatcouldmake
further testinganddevelopmentmoredifficultandtherebyreducethelikelihoodthatthe
testsuitewillfindsecurityissues: •
Usageofglobalvariables. Testsmakeheavyusageofglobalvariablestopersist
values, whichcontaminatescontractvariablenamespace. Nostandard
conventionisusedforvariablenames,andsometimesvariablesarereusedfor
differentpurposesacrosstests. Thismakesreading, understanding, anddebugging
testcasesdifficult. Usageofstructuresandlocalvariablesisrecommendedto increaselegibilityofthecode. •
Codeduplication. Codeduplicationintestcasescanleadtosituationswhereone
ofthetests'codeismodifiedandtheothersarenot, leadingtodifferenttesting
conditions. Manytestsrequirecommonpreconditions(forexample, mintingan

option before burning or liquidating) that could be extracted into independent functions for ease of reading and code simplification. •

Mix between unit tests and fuzzing tests. For clarity, these different types of tests should be kept separate. Additionally, since Panoptic uses network forking for some of the tests, it can also lead to a significant execution speed improvement when only one kind of test is to be executed. •

Complex and long functions. Some tests are complex, spanning multiple lines of code. This is often an indication that parts of the code can be extracted to independent functions to make the tests more readable, and as a consequence, increase the code reutilization. Furthermore, complex test cases usually test several failure or success cases in the same function. These should be rewritten as independent, separate tests. •

Stack depth limitation. Expanding on the previous point, long and complex test functions usually hit the stack depth limit. To overcome this limitation, code blocks and global variables are used in the tests. As a consequence, tests are more difficult to read and the test contract state is contaminated with new variables that are only used in certain functions. •

Lack of stopPrank calls. The test files contain frequent uses of `vm.startPrank()` without the corresponding `vm.stopPrank()`. Keeping a prank active for longer than necessary or between unit test cases can be error-prone, make it more difficult

to reason about the test cases, and hinder debugging and further test case development. Making all unit tests self-contained with explicitly defined callers would be beneficial.

To address these deficiencies and improve the Panoptic's test coverage and processes, we recommend that the Panoptic team define a clear testing strategy and create guidelines on how testing is performed in the codebase. Our general guidelines for improving test suite quality are as follows:

1. Define a clear test directory structure. A clear directory structure helps organize the work of multiple developers, makes it easier to identify which components and behaviors are being tested, and gives insight into the overall test coverage.
2. Write a plain-language design specification of the system, its components, and its functions. Defining a specification can allow the team to more easily detect bugs and inconsistencies in the system, reduce the likelihood that future code changes will introduce bugs, improve the maintainability of the system, and allow the team to create a robust and holistic testing strategy.
3. Use the function specification to guide the creation of unit tests. Creating a specification of all preconditions, postconditions, failure cases, entry points, and execution paths for a function will make it easier to maintain high test coverage and identify edge cases.
4. Use the interaction specification to guide the creation of integration tests. An interaction specification will make it easier to identify the interactions that need to be tested and the external failure cases that need to be validated or guarded against, and it will help identify issues related to access controls and external calls.
5. Implement fuzz testing by first defining a set of system- and function-level invariants and then testing them with Echidna, Foundry, and/or Medusa. Fuzz testing is a powerful technique for exposing security vulnerabilities and finding edge cases that are unlikely to be found through unit testing or manual review. Fuzz testing can be done on a single function by passing in randomized arguments, and on an entire system or on specific components by generating a sequence of random calls to various functions inside the system or component. Both testing approaches should be applied using one or multiple smart contract fuzzers.
6. Use mutation testing to identify gaps in the test coverage and more easily identify bugs in the code. Mutation testing can help identify coverage gaps in unit tests and help discover security vulnerabilities. Taking a two-pronged approach using `Necessist` to mutate tests and a universal mutator or `lither-mutate` to mutate source code can prove valuable in creating a robust test suite.

Directory Structure Creating a specific directory structure for the system's tests will make it easier to develop and maintain the test suite and find coverage gaps. This section contains brief guidelines on defining a directory structure. •

- Create individual directories for each test type (e.g., `unit/`, `integration/`, `fork/`, `fuzz/`) and for the utility contracts. The individual directories can be further divided into directories based on components or behaviors being tested. •
- Create a single base contract that inherits from the shared utility contracts and is inherited by individual test contracts. This will help reduce code duplication across the test suite. •
- Create a clear naming convention for test files and test functions. This will make it easier to filter tests and understand the properties or contracts that are being tested.

Unit Testing We provide the following general recommendations based on our findings: •

- Define a specification for each function and use it to guide the development of the unit tests. See guideline number 3 in the [Identified Testing Deficiencies](#) section above for more information.

- Improve the unit tests' coverage so that they test all functions and contracts in the codebase. Use coverage reports and mutation testing to guide the creation of additional unit tests. •

Use positive unit tests to test that functions and components behave as expected. Ideally, each unit test should test a single property, with additional unit tests for edge cases. The unit tests should test that all expected side effects are correct. •

Improve the use of negative unit tests by not defining test cases that pass on any failure within a test body; instead, each negative unit test should test for a specific failure case.

Integration and Fork Testing

Integration tests build on unit tests by testing how individual components integrate with each other or with third-party contracts. It can often be useful to run integration testing on a fork of the network to make the testing environment as close to production as possible and to minimize the use of mock contracts whose implementation can differ from third-party contracts. We provide the following general recommendations on performing integration and fork testing:

hashey 17 Panoptic Liquidation Engine Invariant Development PUBLIC

- Use the interaction specification to develop integration tests. Ensure that the integration tests aid in verifying the interaction specification. •

Identify valuable input data for the integration tests that can maximize code coverage and test potential edge cases. •

Use negative integration tests, similar to negative unit tests, to test common failure cases. •

Use fork testing to build on top of the integration testing suite. Fork testing will aid in testing third-party contract integrations and in testing the proper configuration of the system once it is deployed. •

Enrich the forked integration tests suite with fuzzed values and call sequences (refer to the Fuzz Testing recommendations below). This will aid in increasing code coverage, validating system-level invariants, and identifying edge cases. Fuzz Testing

Fuzz testing, also known as fuzzing, is an automated testing technique that involves testing program behavior with a large number of inputs and call sequences to discover bugs and vulnerabilities. It can help identify arithmetic errors such as precision loss, logical errors such as insufficient access controls, and other unexpected edge cases that may be difficult to discover through unit testing or manual review. We provide the following general recommendations on performing fuzz testing:

- Defines system- and function-level invariants. Invariants are properties that should always hold within a system, component, or function. Defining invariants is a prerequisite for developing effective fuzz tests that can detect unexpected behavior. Developing a robust system specification will directly aid in the identification of system- and function-level invariants. •
- Improve the fuzz testing coverage. When using Echidna or Medusa, regularly review the coverage files generated at the end of a run to determine whether the property tests' assertions are reached and what parts of the codebase are explored by the fuzzer. To improve the fuzzer's exploration and increase the chances that it finds an unexpected edge case, avoid overconstraining the function arguments. •

Integrate fuzz testing into the CI/CD workflow. Continuous fuzz testing can help quickly identify any code changes that will result in a violation of a system property, and it forces developers to update the fuzz tests suite in parallel with the code. Running fuzz campaigns stochastically may cause a divergence between the operations in the code and the fuzz tests.

hashey 18 Panoptic Liquidation Engine Invariant Development PUBLIC

- Add comprehensive logging mechanisms to all fuzz tests to aid in debugging. Logging during smart contract fuzzing is crucial for understanding the state of the system when a system property is broken. Without logging, it is difficult to identify the arithmetic or operation that caused the failure. •

Enrich each fuzz test with comments explaining the preconditions and postconditions of the test. Strong fuzz testing requires well-defined preconditions (for guiding the fuzzer) and postconditions (for properly testing the invariant[s] in question). Comments explaining the bounds on certain values and the importance of the system properties being tested will aid in tests suite maintenance and debugging efforts. Mutation Testing

At a high level, mutation tests make several changes to each line of a target file and rerun the test suite for each change. Changes that result in test failures indicate adequate test coverage, while changes that do not result in test failures indicate gaps in the test coverage. Although mutation testing is a slow process, it allows auditors to focus their review on areas of the codebase that are most likely to contain latent bugs, and it allows developers to identify and add missing tests.

We recommend using three mutation tools, both of which can help detect redundant code, insufficient test coverage, incorrectly defined tests or conditions, and bugs in the underlying source code being tested: •

- Necessitate perform mutation of the testing suite by iteratively removing lines in the test cases. •

universalmutator performs mutations of the underlying source code. • slither-mutate also mutates the source code, leveraging Slither's static analysis capabilities to increase the mutators' efficiency. hashey

19PanopticLiquidationEngineInvariantDevelopment PUBLIC

D. Fuzzing Harness Design

The following describes the design of the fuzzing harness and details some of the tradeoffs and choices made through the engagement. Given that there is no canonical way of performing invariant tests, understanding the decisions made will help to maintain and improve the invariants written through the engagement.

As shown in figure D.1, the core components of the fuzzing harness are: • The FuzzDeployments contract deploys and initializes all of the different components (e.g., the Panoptics system), sets initial balances, and provides wrappers to the Panoptic and Uniswap pools' functions. •

The Actors, a total of five specific EOA addresses used by Echidna to interact with the Panoptics system through minting, burning, and liquidation of options. They can be seen as normal users of the system that interact with it in the same way a non-privileged user does. •

The pool manipulator, a separate EOA with infinite funds that is set up to interact with a contract to perform controlled swaps on the Uniswap pool. This allows the fuzzer to set the pool price to a specific value by impersonating this EOA. hashey

20PanopticLiquidationEngineInvariantDevelopment PUBLIC

Figure D.1: Harness design Free versus Guided Options Creation

To create an option, the user first needs to have asset tokens, and lock them as collateral by calling the deposit function. Given that collateral cannot be removed once an option has been created, the initial deposit amount will heavily impact the likelihood that the position will be liquidated through the fuzzing sequence. While we did explore the creation of guided options (i.e., options created such that they are close from being liquidated), we decided to keep the options non-guided. This was done to keep the tests as general as possible, to avoid missing possible issues because the parameters were limited in value.

Long and Short Options Minting

Minting long options removes liquidity from the protocol. Minting short options has the opposite effect: it provides liquidity to the system. To be able to mint long options associated with a short option, we tested two strategies. The first one was to mint a short position before minting a long, with the same parameters and a position size bigger than the long option. The second one was to keep track of the minted short positions, and create longs with those parameters. Sequence Length

This sequence length of a fuzzer run is the number of transactions the fuzzer will perform until it resets the EVM state to its initial point. A large sequence length allows exploration of a more complex state; however, there is also a risk that the state ends in a locked state (for example, all the tokens are transferred to a locked place). Based on our empirical evaluation, we decided to use a length of 500. This affects the tests, as mentioned in the Coverage Limitations section. On-Chain versus Off-Chain Fuzzing Echidna supports both on-chain and off-chain fuzzing. While on-chain fuzzing allows more realistic scenarios, as it can use real pool information from Uniswap, it significantly slowed the fuzzing process, and it could prevent the exploration of black swan events. As a result, we decided to build an off-chain fuzzing harness. The downside of this approach is that all liquidity, activity, and fees of the pool will have to be simulated in the tests. Single versus Multiple Entry Points By default, Echidna calls only one contract. However, the fuzzer has the possibility to call any deployed contracts if the allContracts option is set. Using allContracts has the advantage of generating more complex interactions, in particular if the exploration requires to call multiple contracts in different steps. However, it comes at a cost for the exploration: the number of potential calls can quickly rise, and include unproductive calls, such as calling privileged functions from a normal account. In addition, randomly finding the right inputs for complex parameters (e.g., tokenId) could be challenging for the fuzzer. We decided to use a single entry point (FuzzDeployments), favoring a targeted approach

21PanopticLiquidationEngineInvariantDevelopment PUBLIC

focused on the options minting and their liquidations. This allows the harness to do the following: •

- Easily keep track of the options minted, and their parameters
 - Focus the exploration toward the targeted actions
- EOAs versus Contract Actors When testing a system with multiple actors, two approaches can be taken: •
- Leveraging the prank cheat code in the main harness contract, specifying the exact address that will make the call
 - Creating one individual contract wrapper per actor
- This requires setting allContracts to true
- Using prank has the advantage of lowering the complexity of the harness, while the individual wrapper enables better isolation of the operations and avoids cheat code-related

bugs. However, there are some risks associated with the use of cheat codes that have to be considered; for example, you can inadvertently start transactions from contract accounts, making the test situation impossible to replicate in a live network. In the current harness, we chose to leverage cheat codes for ease of use. If the harness grows to a point where multiple actors have different actions, moving toward individual contract wrappers could be beneficial. `hashey 22PanopticLiquidationEngineInvariantDevelopment PUBLIC`