Curvance

Invariant Development

March 13, 2024

Prepared for:
Chris
Curvance

Prepared by: **Nat Chin and Priyanka Bose**

About Trail of Bits

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

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

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

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

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

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

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

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

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

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

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Nat Chin, Consultant
natalie.chin@trailofbits.com

Priyanka Bose, Consultant
priyanka.bose@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below. The meetings listed include only the official status meetings held weekly on Fridays; however, the engagement involved daily syncs with the client, which are not listed below.

Date	Event
December 8, 2023	Pre-project kickoff call
December 18, 2023	Status update meeting #1
December 22, 2023	Status update meeting #2
December 25, 2023	Start of one-week break during the engagement
January 5, 2024	Status update meeting #3
January 12, 2024	Status update meeting #4
January 19, 2024	Status update meeting #5
January 26, 2024	Status update meeting #6
January 29, 2024	Start of one-week break during the engagement
February 9, 2024	Status update meeting #7

February 16, 2024	Status update meeting #8
February 26, 2024	Status update meeting #9
March 4, 2024	Delivery of initial report draft
March 4, 2024	Final readout meeting
March 13, 2024	Delivery of final invariant development report

Executive Summary

Engagement Overview

Curvance engaged Trail of Bits for an invariant development and testing exercise for the Curvance codebase.

One consultant conducted the exercise from December 11, 2023, to March 1, 2024, for a total of nine engineer-weeks of effort; another consultant shadowed the engagement from January 8 to January 19, 2024, spending two additional engineer-weeks of effort writing invariants. Our testing efforts focused on the vested CVE (veCVE) and market manager components. With full access to source code, documentation, and tests, we identified and wrote invariants of the system and ran them simultaneously with Medusa and Echidna. The deliverable from this invariant development and testing exercise includes a stateful fuzz testing suite to test the invariants we developed, covering the veCVE and market manager components; the Echidna and Medusa corpus that was run for 11 calendar weeks; and this report, which includes a summary of the invariants we wrote, the security findings that resulted from our testing, recommendations for writing future invariants and expanding the fuzz testing suite, and other insights.

Observations and Impact

During this engagement, we found high-impact findings affecting the `VeCVE` contract, including a high-severity issue in the `processExpiredLock` function; the function does not delete old locks when it adds new ones, allowing users to have more locks than they should and attackers to increase their user points and inflate their veCVE voting power (TOB-CURV-4). Other issues involve the `combineAllLocks` function; specifically, it does not account for expired locks (TOB-CURV-6), and it can still run when the system is shut down (TOB-CURV-5). Based on these issues, Curvance rewrote the function and we rebased the fuzz testing suite based on the changes.

We also found rounding issues in the `MarketManager` contract, specifically related to one-off errors in calculations related to the values of `totalBorrows` and `accountDebt` (e.g., TOB-CURV-7). We found functions that, in certain cases, panic instead of erroring out gracefully (e.g., TOB-CURV-3). Toward the end of the engagement, we also found issues related to the lack of data validation on important addresses, such as the DAO address, that would break assumptions of balances being transferred (e.g., TOB-CURV-11).

Recommendations

Based on the codebase maturity evaluation and findings identified during the testing exercise, Trail of Bits recommends that Curvance take the following steps prior to launching the system:

- **Continue to extend the fuzz testing suite, following the guidance outlined in appendix C.** This appendix highlights areas of the fuzz testing suite that lack coverage in the two tested components.
- **Extend the stateful fuzz testing suite to other components of the Curvance architecture.**
- **Perform additional mathematical analysis on all rounding directions to ensure that the code always rounds in favor of the protocol.** The most recently found findings involve underflow and rounding issues in both tested components, which should be heavily scrutinized for correctness.

Finding Severities

The following table provides the number of findings by severity.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	2
Low	2
Informational	0
Undetermined	8

Project Goals

The engagement was scoped to write and test invariants of the Curvance contracts. Specifically, we used the following non-exhaustive list of questions to guide our development:

- What are the expected states of the system in which functions can be called?
- When should the execution of functions fail and with what error messages?
- What boundary conditions on inputs are necessary for a function to be called successfully each time?
- What are the safe bounds on outputs that are necessary for a function?
- Do the functions safely account for the necessary range of inputs?

Project Targets

The engagement involved testing of the following target.

curvance-contracts

Repository	https://github.com/curvance/curvance-contracts
Type	Solidity
Platform	Ethereum

The following table summarizes our rebasing timeline and associated commits:

Commit	Date of Rebase	Branch/PR
d04463d	December 11, 2023	fuzzing/PR #107
9830ebb	December 16, 2023	fuzzing/PR #107
1c1ab73	December 19, 2023	fuzzing/PR #107
13d350	December 20, 2023	fuzzing/PR #107
94e582f	December 21, 2023	fuzzing-lendtroller
8269546	January 22, 2024	fuzz-vecve/PR #141
a96dc9a	January 22, 2024	fuzz-market-manager/PR #147
8c9a2dd	February 20, 2024	fuzz-liquidations/PR #156
e61711d	February 26, 2024	fuzz-liquidations/PR #156

Project Coverage

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

- **VeCVE:** This contract implements vested rewards through CVE tokens, allowing users to lock funds using either a noncontinuous lock (ending after 52 weeks) or a continuous lock (without an end time). During the engagement, we developed a fuzz testing suite that tests the publicly callable functions in the `VeCVE` contract, checking specified preconditions and postconditions of functions and system invariants.

- **Market manager contracts:** These contracts implement the markets of the system, allowing users to post collateral, liquidate user accounts, borrow tokens, and repay existing debts. During the engagement, we developed a testing suite that tests the publicly callable functions in the `MarketManager` (previously called `Lendtroller`) contract, checking specified preconditions and postconditions of functions and system invariants.

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 fuzz testing suite and indicates system elements that warrant additional effort:

System-Wide Limitations

- **Range of oracle prices:** The current test suite uses a default price of 1e8 for each asset to match the unit tests. The range of oracle prices should be expanded to allow the fuzzer to explore large price deviations.

- **Number of token interactions:** The range of debt and collateral tokens the fuzzer can transact with is limited. The fuzzer should be expanded with additional assets valued in USD and ETH and additional decimal checks.

- **Input ranges:** The fuzzer does not currently test the full range of inputs (e.g., for `uint256`, the full range of `0` through `type(uint256).max`).

VeCVE Contract Limitations

- **Comprehensiveness of system invariants:** Certain invariants of the `VeCVE` contract still need to be written and tested (e.g., the value of a user's `chainPoints` is equal to the result of this calculation: (noncontinuously locked CVE) + locked CVE / the `continuousPoint` multiplier.

- **Coverage of delegation functions:** Coverage of functions such as `createLockFor` and `increaseAmountAndExtendLockFor` is missing. They should be covered.

- **Preconditions and postconditions for certain functions:** Some functions such as `earlyExpire` were introduced after we started our fuzzing campaign, so they were not scoped for preconditions and postconditions and are missing coverage.
- **Reward data:** Currently, the fuzz testing suite tests against default reward data that has no empty bytes. Given changes implemented in the reward claiming logic on the VeCVE side, the fuzz testing suite is missing coverage of these functions.
- **Range of `uint` values for inputs:** The current fuzz testing suite does not test the full range of `uint` values for inputs. For example, the `createLock` function is currently bound at `uint64`. The upper bounds of inputs should be extended in the testing suite.

Market Manager Component Limitations

- **Partner gauges:** The system does not test the `GaugePool` contract and its interactions with partner gauges.
- **Removal of collateral with a shortfall:** Coverage is currently missing on the effects of removing collateral with a shortfall greater than zero, which may need additional tweaking with respect to the system state.
- **`MarketManager` state-checking functions:** The `canLiquidate` and `canLiquidateWithExecution` functions are missing coverage.

Automated Testing

Trail of Bits 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: 40,000 runs (approximately 4 minutes) Cloud: 10,000,000 runs (approximately 20 hours) Extended cloud: 100,000,000,000 runs
Medusa	A cross-platform `go-ethereum`-based fuzzer providing parallelized fuzz testing of smart contracts, heavily inspired by Echidna	Local: 2 minutes Cloud: 24 hours Extended cloud: 102 hours*

*Medusa was initially limited by an out-of-memory bug that caused a panic, and crashed at roughly the 2 hour and 30 minute mark. This prevented the fuzzer from running in extended execution. As of February 9, 2024, the out-of-memory bug has been fixed, and we have been running Medusa for 102 hours on extended runs.

Summary of Invariants

The table below summarizes the number and type of invariants we ran for each component. We ran the fuzzer both locally and on the cloud.

Component	Invariant Type	Total Number
Deployment	System invariants	16
`VeCVE`	Functional invariants	60
`VeCVE`	System invariants	7
`MarketManager`	Functional invariants	42
`MarketManager`	Functional invariants; role-based access control checks	15
`MarketManager`	State-checking functions	25
`MarketManager`	System invariants	6
`MarketManager`	Arithmetic functional invariants	13
`DToken`	Functional invariants	29
`DToken`	System invariants	3
Total Invariants		**216**

Summary of Failed Invariants

Throughout the engagement, several invariants failed. The following failed invariants led us to discover vulnerabilities:

- Calls to `combineAllLocks` that combine all continuous locks into a single continuous lock terminal result in identical user points before and after the operation. (VECVE-4, TOB-CURV-1)
- Calls to `combineAllLocks` that combine some noncontinuous locks into a single continuous lock terminal result in increased user points post-operation. (VECVE-10, TOB-CURV-2)
- Calls to `repay` with an `amount` value that is too high error out gracefully. (DTOK-11, TOB-CURV-3)
- Calls to `processExpiredLocks` with the relock option do not change the number of locks a user has. (VECVE-55, TOB-CURV-4)
- Calls to `combineAllLocks` are not possible when the system is shut down. (VECVE-56, TOB-CURV-5)
- Calls to `combineAllLocks` that combine some continuous locks into a noncontinuous lock terminal result in a user veCVE balance equal to the user points. (VECVE-18, TOB-CURV-6/TOB-CURV-4)
- Calls to liquidate an entire account with the correct preconditions succeed. (MARKET-35, TOB-CURV-7)
- Calls to liquidate an account result in no more than a 1 wei difference between `totalborrows` and `accountDebt`. (MARKET-42, TOB-CURV-7)
- Calls to `combineAllLocks` that combine previously created locks, none of which are continuous, into a noncontinuous lock terminal result in no change in user points. (VECVE-17, TOB-CURV-8)
- Calls to `updateCollateralToken` with large price deviations or faulty oracles error out with the `PriceError` error. (MARKET-7, TOB-CURV-9)
- Soft liquidations of exactly zero tokens fail with the `InvalidAmount` or `InvalidParameter` error. (DTOK-19, TOB-CURV-10)
- Liquidations of a non-exact amount decrease the collateral balance of the account. (DTOK-23, TOB-CURV-11)
- Liquidations of a non-exact amount increase the collateral token balance by the amount seized for the liquidation minus the amount seized for the protocol. (DTOK-25, TOB-CURV-12)

- Calls to `repay` with the proper preconditions succeed. (DTOK-12, TOB-CURV-13)

Stateful Deployment State Tests

These properties check whether the system is deployed correctly with the respective addresses set to the correct values. While they check addresses used for deploying the `VeCVE` and `MarketManager` contracts specifically, they can be extended to check other contract deployment states as well.

ID	Property	Result
CURV-1	The central registry's `daoAddress` value is set to the second fuzzing caller.	**Passed**
CURV-2	The central registry's `timelock` address is set to the deployer.	**Passed**
CURV-3	The central registry's `emergencyCouncil` address is set to the deployer.	**Passed**
CURV-4	The central registry's `genesisEpoch` value is set to `0`.	**Passed**
CURV-5	The central registry's `sequencer` value is set to `address(0)`.	**Passed**
CURV-6	The central registry grants the deployer permissions.	**Passed**
CURV-7	The central registry grants the deployer elevated permissions.	**Passed**
CURV-8	The central registry's `cve` address is set up correctly.	**Passed**
CURV-9	The central registry's `veCVE` address is set up correctly.	**Passed**
CURV-10	The central registry's `cveLocker` address is set up correctly.	**Passed**
CURV-11	The central registry's `protocolMessagingHub` address is set up correctly.	**Passed**
CURV-12	The `CVE` contract is mapped to the `centralRegistry` contract correctly.	**Passed**

CURV-13	The `CVE` contract's team address is set to the deployer.	**Passed**
CURV-14	The `CVE` contract's DAO treasury allocation is set to 10,000 ether.	**Passed**
CURV-15	The `CVE` contract's DAO team allocation per month is greater than zero.	**Passed**
CURV-16	The `MarketManager` contract's gauge pool is set up correctly.	**Passed**

VeCVE Functional Invariants

We ran the following invariants using Medusa and Echidna to test functions in the `VeCVE` contract to ensure that they behave as expected. They include checks of preconditions and postconditions expected to hold in the system.

ID	Property	Result
VECVE-1	Calls to `createLock` with a specified amount when the system is not in the shutdown state succeed; the value of `preLockCVEBalance` matches the sum of `postLockCVEBalance` and `amount`, and the sum of `preLockVECVEBalance` and `amount` matches the value of `postLockVECVEBalance`.	**Passed**
VECVE-2	Calls to `createLock` with an `amount` value less than `WAD` fail and revert with an error message indicating an invalid lock amount.	**Passed**
VECVE-3	Calls to `createLock` with `amount` set to `0` fail and revert with an error message indicating an invalid lock amount.	**Passed**
VECVE-4	Calls to `combineAllLocks` that combine all continuous locks into a single continuous lock terminal result in identical user points before and after the operation.	**Passed (13d350)** **Failed (TOB-CURV-1)**
VECVE-5	Calls to `combineAllLocks` that combine all continuous locks into a single continuous lock terminal result in user points post-operation that are greater than the result of the user's veCVE balance * the multiplier / `WAD`.	**Passed**
VECVE-6	Calls to `combineAllLocks` that combine all continuous locks into a single continuous lock terminal result in a `chainUnlocksByEpoch` value equal to `0`.	**Passed**
VECVE-7	Calls to `combineAllLocks` that combine all continuous locks into a single continuous lock terminal result in a `userUnlocksByEpoch` value equal to `0`.	
VECVE-8	Calls to `combineAllLocks` that combine all noncontinuous locks into a single noncontinuous lock	**Passed**

	terminal result in a combined lock amount that matches the sum of the original lock amounts.	
VECVE-9	Calls to `combineAllLocks` that combine all continuous locks into a single continuous lock terminal result in a value of user points times the `CL_POINT_MULTIPLIER` that is greater than or equal to the user's veCVE balance.	**Passed**
VECVE-10	Calls to `combineAllLocks` that combine some noncontinuous locks into a single continuous lock terminal result in increased user points post-operation.	**Passed (13d350)** **Failed (TOB-CURV-2)**
VECVE-11	Calls to `combineAllLocks` that combine noncontinuous locks into continuous lock terminals result in a decrease in the `userUnlockByEpoch` value for each respective epoch.	**Passed**
VECVE-12	Calls to `combineAllLocks` that combine noncontinuous locks into continuous lock terminals result in a decrease in the `chainUnlockByEpoch` value for each respective epoch.	**Passed**
VECVE-13	Calls to `combineAllLocks` that combine noncontinuous locks into continuous lock terminals result in a `chainUnlockByEpochs` value equal to `0`.	**Passed**
VECVE-14	Calls to `combineAllLocks` that combine noncontinuous locks into continuous lock terminals result in a `userUnlocksByEpoch` value equal to `0`.	**Passed**
VECVE-15	Calls to `combineAllLocks` that combine any locks into a noncontinuous lock terminal result in a combined terminal amount that matches the sum of the original lock amounts.	**Passed**
VECVE-16	Calls to `combineAllLocks` that combine continuous locks into a noncontinuous terminal result in a decrease in user points.	**Passed**
VECVE-17	Calls to `combineAllLocks` that combine previously created locks, none of which are continuous, into a noncontinuous lock terminal result in no change in user	**Failed (TOB-CURV-8)**

	points.	
VECVE-18	Calls to `combineAllLocks` that combine some continuous locks into a noncontinuous lock terminal result in a user veCVE balance equal to the user points.	**Passed (8c9a2dd)** **Failed (TOB-CURV-4, TOB-CURV-6)**
VECVE-19	Calls to `processExpiredLock` fail when the lock index is incorrect or exceeds the length of created locks.	**Passed**
VECVE-20	Calls to `disableContinuousLock` result in a decrease in user points.	**Passed**
VECVE-21	Calls to `disableContinuousLock` result in a decrease in chain points.	**Passed**
VECVE-22	Calls to `disableContinuousLock` result in an increase in the value of `chainUnlocksByEpoch`.	**Passed**
VECVE-23	Calls to `disableContinuousLock` result in a sum of `preUserUnlocksByEpoch` and `amount` that matches the value of `postUserUnlocksByEpoch`.	**Passed**
VECVE-24	Attempts to extend a lock that is already continuous fail and revert with an error message indicating a lock type mismatch.	**Passed**
VECVE-25	Attempts to extend a lock when the system is in the shutdown state fail and revert with an error message indicating that the system is shut down.	**Passed**
VECVE-26	Calls to shut down the `VeCVE` contract by a caller with elevated permissions result in a `veCVE.isShutdown` value of 2.	**Passed**
VECVE-27	Calls to shut down the `VeCVE` contract by a caller with elevated permissions result in a `cveLocker.isShutdown` value of 2.	**Passed**

VECVE-28	Calls to shut down the VeCVE contract by a caller with elevated permissions when the system is not already shut down never revert unexpectedly.	**Passed**
VECVE-29	Calls to `extendLock` with `continuousLock` set to `true` set the post-extension lock time to `CONTINUOUS_LOCK_VALUE`.	**Passed**
VECVE-30	Calls to `extendLock` for a noncontinuous extension in the same epoch do not change the unlock epoch.	**Passed**
VECVE-31	Calls to `extendLock` for a noncontinuous extension in a future epoch increase the unlock time.	**Passed**
VECVE-32	Calls to `extendLock` with the correct preconditions do not revert.	**Passed**
VECVE-33	Calls to extend a continuous lock and increase its amount succeed.	**Passed**
VECVE-34	Calls to extend a continuous lock and increase its amount succeed and result in a `preLockCVEBalance` that matches the sum of `postLockCVEBalance` and `amount`.	**Passed**
VECVE-35	Calls to extend a continuous lock and increase its amount succeed and result in a sum of `preLockVECVEBalance` and `amount` that matches the value of `postLockVECVEBalance`.	**Passed**
VECVE-36	Calls to extend a noncontinuous lock and increase its amount succeed.	**Passed**
VECVE-37	Calls to extend a noncontinuous lock and increase its amount succeed and result in a value of `preLockCVEBalance` that matches the sum of `postLockCVEBalance` and `amount`.	**Passed**
VECVE-38	Calls to extend a noncontinuous lock and increase its amount succeed and result in a sum of `preLockVECVEBalance` and `amount` that matches the value of `postLockVECVEBalance`.	**Passed**

VECVE-39	Calls to process an expired lock for an existing lock without the relock option in a shutdown `VeCVE` contract complete successfully.	**Passed**
VECVE-40	Calls to process a lock without the relock option in a shutdown `VeCVE` contract result in a decrease in user points.	**Passed**
VECVE-41	Calls to process a lock without the relock option in a shutdown `VeCVE` contract result in a decrease in chain points.	**Passed**
VECVE-42	Calls to process a noncontinuous lock without the relock option in a shutdown `VeCVE` contract result in a difference of `preChainUnlocksByEpoch` and `amount` that is equal to `postChainUnlocksByEpoch`.	**Passed**
VECVE-43	Calls to process a noncontinuous lock without the relock option in a shutdown `VeCVE` contract result in a difference of `preUserUnlocksByEpoch` and `amount` that is equal to `postUserUnlocksByEpoch`.	**Passed**
VECVE-44	Calls to process a lock without the relock option in a shutdown `VeCVE` contract result in an increase in CVE tokens.	**Passed**
VECVE-45	Calls to process a lock without the relock option in a shutdown `VeCVE` contract result in a decrease in veCVE tokens.	**Passed**
VECVE-46	Calls to process a lock without the relock option in a shutdown `VeCVE` contract result in a decrease in the number of user locks.	**Passed**
VECVE-47	Calls to process a lock with the relock option complete successfully if the unlock time is expired.	**Passed**
VECVE-48	Calls to process a lock with the relock option in a shutdown `VeCVE` contract result in a decrease in chain points.	**Passed**
VECVE-49	Calls to process a noncontinuous lock with the relock	**Passed**

	option in a shutdown `VeCVE` contract result in a difference of `preChainUnlocksByEpoch` and `amount` that is equal to `postChainUnlocksByEpoch`.	
VECVE-50	Calls to process a noncontinuous lock with the relock option in a shutdown `VeCVE` contract result in a difference of `preUserUnlocksByEpoch` and `amount` that is equal to `postUserUnlocksByEpoch`.	**Passed**
VECVE-51	Calls to `processExpiredLocks` without the relock option result in an increase in CVE tokens.	**Passed**
VECVE-52	Calls to `processExpiredLocks` without the relock option result in a decrease in veCVE tokens.	**Passed**
VECVE-53	Calls to `processExpiredLocks` without the relock option result in no change in user points.	**Passed**
VECVE-54	Calls to `processExpiredLocks` without the relock option result in no change in chain points if the epochs to claim equals `0`.	**Passed**
VECVE-55	Calls to `processExpiredLocks` with the relock option do not change the number of locks a user has.	**Passed (`13d350`)** **Failed (TOB-CURV-4)**
VECVE-56	Calls to `combineAllLocks` are not possible when the system is shut down.	**Passed (`13d350`)** **Failed (TOB-CURV-5)**
VECVE-57	Calls to `processExpiredLocks` without the relock option decrease user points if the epochs to claim is greater than zero.	**Passed**
VECVE-58	Calls to `createLock` with the correct preconditions do not revert.	**Passed**
VECVE-59	Calls to `combineAllLocks` with the correct preconditions that combine noncontinuous locks into continuous lock	**Passed**

	terminals are successful.	
VECVE-60	Calls to `combineAllLocks` with the correct preconditions that combine continuous locks into noncontinuous lock terminals are successful.	**Passed**

VeCVE System Invariants

Using Medusa and Echidna, we also added system invariants that check the relationship between global system states.

These invariants test the relationships between variables in the contract, including the balances of tokens and of points and locks stored in the system. Unlike functional invariants, these invariants should hold true regardless of the functions that are executed.

ID	Property	Result
S-VECVE-1	The balance of veCVE is equal to the sum of all noncontinuous lock amounts.	**Passed**
S-VECVE-2	User unlocks by epoch are greater than `0` for all noncontinuous locks.	**Passed**
S-VECVE-3	User unlocks by epoch are `0` for all continuous locks.	**Passed**
S-VECVE-4	Chain unlocks by epoch are greater than `0` for all noncontinuous locks.	**Passed**
S-VECVE-5	Chain unlocks by epoch are `0` for all continuous locks.	**Passed**
S-VECVE-6	The sum of all user unlocks for each epoch is less than or equal to the user points.	**Passed**
S-VECVE-7	The VeCVE contract has a zero-value CVE balance only when there are no user locks.	**Passed**

MarketManager Functional Invariants

These invariants check the preconditions and postconditions of `MarketManager`-specific functions based on their success and failure cases. In the case of failed transactions, these tests also check that the function errors out with the correct error message, where relevant.

ID	Property	Result
MARKET-1	Once a new token is listed, `isListed(mtoken)` returns `true`.	**Passed**
MARKET-2	Tokens already added to the `MarketManager` contract cannot be added again.	**Passed**
MARKET-3	Users can deposit into an mToken market provided that they have the underlying asset and they have approved the mToken contract.	**Passed**
MARKET-4	When depositing assets into an mToken market, the wrapped token balance for the user increases.	**Passed**
MARKET-5	Calls to `updateCollateralToken` with variables in the correct bounds succeed.	**Passed**
MARKET-6	Calls to `updateCollateralToken` with a divergence in prices that is too large fail with the `PriceError` error.	**Passed**
MARKET-7	Calls to `updateCollateralToken` with large price deviations or faulty oracles error out with the `PriceError` error.	**Passed (8c9a2dd)** **Failed (TOB-CURV-9)**
MARKET-8	Calls to `updateCollateralToken` on a token with a nonzero collateral ratio do not allow the new collateral ratio to be set to `0`.	**Passed**
MARKET-9	Calls to set the collateral caps for a token increase the globally set value for the specific token.	**Passed**

MARKET-10	Calls to set collateral caps for a token with permissions and collateral values set succeed.	**Passed**
MARKET-12	Calls to `updateCollateralToken` with inputs within the correct bounds revert if the price feed is out of date.	**Passed**
MARKET-13	After collateral is posted, the user's posted collateral position for the respective asset increases.	**Passed**
MARKET-14	After collateral is posted, calls to `hasPosition` on the user's mToken return `true`.	**Passed**
MARKET-15	After collateral is posted, the global collateral for the mToken increases by the amount posted.	**Passed**
MARKET-16	When the price feed is up to date, `address` is set to `mtoken`, tokens are bound correctly, and the caller is correct, calls to `postCollateral` succeed.	**Passed**
MARKET-17	Attempts to post too much collateral revert.	**Passed**
MARKET-18	Calls to `removeCollateral` decrease the global posted collateral by the removed amount.	**Passed**
MARKET-19	Calls to `removeCollateral` reduce the user's posted collateral by the removed amount.	**Passed**
MARKET-20	Users with a liquidity shortfall are not permitted to remove collateral; the `removeCollateral` function fails with the insufficient collateral selector hash error.	**Passed**
MARKET-21	Users who do not have a liquidity shortfall and meet the expected preconditions can successfully call `removeCollateral`.	**Passed**
MARKET-22	Users who have a collateral value of zero after removing their collateral and who then close their position no longer have a position in the asset.	**Passed**
MARKET-23	Calls to `removeCollateral` for a nonexistent position revert with the invariant error hash.	**Passed**

MARKET-24	Calls to `removeCollateral` that remove more tokens than the user has in collateral revert with the insufficient collateral hash.	**Passed**
MARKET-25	Calls to `reduceCollateralIfNecessary` not made within the context of the mToken fail.	**Passed**
MARKET-26	Calls to `closePosition` with the correct preconditions remove a position in the mToken when the collateral posted for the user is greater than `0`.	**Passed**[1]
MARKET-27	Calls to `closePosition` with the correct preconditions set `collateralPosted` for the user's mToken to `0` when the collateral posted for the user is greater than `0`.	**Passed**[1]
MARKET-28	Calls to `closePosition` with the correct preconditions reduce the user asset list by one element when the collateral posted for the user is greater than `0`.	**Passed**[1]
MARKET-29	Calls to `closePosition` with the correct preconditions succeed when the collateral posted for the user is greater than `0`.	**Passed**[1]
MARKET-30	Calls to `closePosition` when the user has a liquidity shortfall revert with the insufficient collateral error.	**Passed**[1]
MARKET-31	Calls to `closePosition` with the correct preconditions remove a position in the mToken when the collateral posted for the user is equal to `0`.	**Passed**[1]
MARKET-32	Calls to `closePosition` with the correct preconditions set `collateralPosted` for the user's mToken to `0` when the collateral posted for the user is equal to `0`.	**Passed**[1]
MARKET-33	Calls to `closePosition` with the correct preconditions reduce the user asset list by one element when the collateral posted for the user is equal to `0`.	**Passed**[1]

MARKET-34	Calls to `closePosition` with the correct preconditions succeed when the collateral posted for the user is equal to `0`.	**Passed**[1]
MARKET-35	Calls to liquidate an entire account with the correct preconditions succeed.	**Further Investigation Required** (TOB-CURV-7)
MARKET-36	Calls to liquidate an entire account zero out the user's balance for every collateral token they deposited.	**Passed**
MARKET-37	Calls to liquidate an entire account remove the user's position in every asset.	**Passed**
MARKET-38	Attempts to liquidate an entire account (hard liquidation) fail with the `NoLiquidationAvailable` error if the collateral is greater than or equal to the debt.	**Passed**
MARKET-39	Attempts by users to liquidate their entire accounts (hard liquidation) themselves fail with the `Unauthorized` error.	**Passed**
MARKET-40	Attempts to liquidate an entire account (hard liquidation) fail with the `Paused` error if the seize feature is paused.	**Passed**
MARKET-41	Calls to `removeCollateral` with zero tokens fail.	**Passed**
MARKET-42	Calls to liquidate an account result in no more than a 1 wei difference between `totalborrows` and `accountDebt`.	**Further Investigation Required** (TOB-CURV-7)

[1] These invariants were removed in commit `e61711d` of the `MarketManager` contract due to the introduction of an implicit prune. While it is included in the final deliverable for the fuzz testing suite, the `closePosition` function was removed.

MarketManager Role-Based Access Control Invariants

These invariants check the correctness of privileged functions in the `MarketManager` contract, such as those that pause aspects of the system, and ensure that the system states change according to these values.

ID	Property	Result
AC-MARKET-1	Calls to `setMintPaused` with the correct preconditions do not revert.	**Passed**
AC-MARKET-2	Calls to `setMintPaused(mtoken, true)` with the correct authorization set `mintPaused` to 2.	**Passed**
AC-MARKET-3	Calls to `setMintPaused(mtoken, false)` with the correct authorization set `mintPaused` to 1.	**Passed**
AC-MARKET-4	Calls to `setRedeemPaused` with the correct preconditions succeed.	**Passed**
AC-MARKET-5	Calls to `setRedeemPaused(true)` with the correct authorization set `redeemPaused` to 2.	**Passed**
AC-MARKET-6	Calls to `setRedeemPaused(false)` with the correct authorization set `redeemPaused` to 1.	**Passed**
AC-MARKET-7	Calls to `setTransferPaused` with the correct preconditions do not revert.	**Passed**
AC-MARKET-8	Calls to `setTransferPaused(true)` with the correct authorization set `transferPaused` to 2.	**Passed**
AC-MARKET-9	Calls to `setTransferPaused(false)` with the correct authorization set `transferPaused` to 1.	**Passed**
AC-MARKET-10	Calls to `setSeizePaused` with the correct authorization succeed.	**Passed**
AC-MARKET-11	Calls to `setSeizePaused(true)` set `seizePaused` to 2.	**Passed**

AC-MARKET-12	Calls to `setSeizePaused(false)` set `seizePaused` to 1.	**Passed**
AC-MARKET-13	Calls to `setBorrowPaused` with the correct preconditions succeed.	**Passed**
AC-MARKET-14	Calls to `setBorrowPaused(mtoken, true)` set `borrowPaused` to 2.	**Passed**
AC-MARKET-15	Calls to `setBorrowPaused(mtoken, false)` set `borrowPaused` to 1.	**Passed**

MarketManager State Check Invariants

These invariants check that functions revert and succeed when they are expected to. They specifically target the `MarketManager` contract, checking the success and failure cases of functions such as `canMint`, `canRedeem`, `canTransfer`, `canBorrow`, and `canSeize`.

ID	Property	Result
SC-MARKET-1	The `canMint` function does not revert when `mintPaused` is set to 1 and the given token is listed in the system.	**Passed**
SC-MARKET-2	The `canMint` function reverts when the given token is not listed.	**Passed**
SC-MARKET-3	The `canMint` function reverts when `mintPaused` is set to 2.	**Passed**
SC-MARKET-4	The `canRedeem` function succeeds when `redeemPaused` is set to 1, the given mToken is listed, `MIN_HOLD_PERIOD` has passed since posting, and the user does not have a liquidity deficit.	**Passed**
SC-MARKET-5	The `canRedeem` function reverts when `redeemPaused` is set to 2.	**Passed**
SC-MARKET-6	The `canRedeem` function reverts when the given token is not listed.	**Passed**
SC-MARKET-7	The `canRedeem` function reverts when the user has a `liquidityDeficit` greater than zero.	**Passed**
SC-MARKET-8	The `canRedeem` function returns (without erroring out) when no position exists.	**Passed**
SC-MARKET-9	The `canRedeemWithCollateralRemoval` function fails when it is not called by the given mToken address.	**Passed**
SC-MARKET-10	The `canTransfer` function passes when all preconditions are met.	**Passed**
SC-MARKET-11	The `canTransfer` function fails when `transferPaused` is set to 2.	**Passed**

SC-MARKET-12	The `canTransfer` function fails when the given mToken is not listed.	**Passed**
SC-MARKET-13	The `canTransfer` function fails when `redeemPaused` is set to 2.	**Passed**
SC-MARKET-14	The `canBorrow` function succeeds when `borrowPaused` is set to 1 and the given mToken is listed.	**Passed**
SC-MARKET-15	The `canBorrow` function fails when `borrowPaused` is set to 2.	**Passed**
SC-MARKET-16	The `canBorrow` function fails when the given mToken is unlisted.	**Passed**
SC-MARKET-17	The `canBorrow` function fails when a liquidity deficit exists.	**Passed**
SC-MARKET-18	The `canBorrowWithNotify` function fails when it is called directly.	**Passed**
SC-MARKET-19	The `canRepay` function succeeds when the given mToken is listed and `MIN_HOLD_PERIOD` has passed.	**Passed**
SC-MARKET-20	The `canRepay` function reverts when the given mToken is not listed.	**Passed**
SC-MARKET-21	The `canRepay` function reverts when `MIN_HOLD_PERIOD` has not passed.	**Passed**
SC-MARKET-22	The `canSeize` function succeeds when `seizePaused` is 1, the given collateral and debt token are listed, and both tokens have the same market manager.	**Passed**
SC-MARKET-23	The `canSeize` function reverts when `seizePaused` is 2.	**Passed**
SC-MARKET-24	The `canSeize` function reverts when the given collateral or debt token is not listed in the market manager.	**Passed**
SC-MARKET-	The `canSeize` function reverts when both tokens do not	**Passed**

25	have the same market manager.	

MarketManager System Invariants

These invariants implement higher-level checks, including checks on the relationships between variables and the larger global system state in the `MarketManager` contract.

ID	Property	Result
S-MARKET-1	A user's cToken balance is always greater than the total collateral posted for that cToken.	**Passed**
S-MARKET-2	If a token's posted market collateral is `0`, its posted collateral is equal to the maximum collateral cap.	**Passed**
S-MARKET-3	If a token's collateral cap is nonzero, the posted market collateral is always less than the maximum collateral cap.	**Passed**
S-MARKET-4	The total supply of a token never goes down to zero once it has been listed.	**Passed**
S-MARKET-5	If no positions in an asset need to be pruned, the collateral posted for the asset is zero.	**Passed**
S-MARKET-6	If no positions in an asset need to be pruned, users cannot take new positions in the asset.	**Passed**

MarketManager Liquidation Conditions

These invariants check the bounds of inputs involved in the liquidation process. Analysis of these bounds is provided in appendix F.

ID	Property	Result
LIQ-1	The value of `baseCFactor` is bound between `MIN_BASE_CFACTOR` and `MAX_BASE_CFACTOR`.	**Passed**
LIQ-2	The value of `lFactor` is bound between 1 and `WAD`.	**Passed**
LIQ-3	The value of `cFactor` resulting from the associated calculation is bound between `baseCFactor` and `WAD`.	**Passed**
LIQ-4	The value of `liqBaseIncentive` is bound between `MIN_LIQUIDATION_INCENTIVE` and `MAX_LIQUIDATION_INCENTIVE`.	**Passed**
LIQ-5	The value of `incentive` resulting from the associated calculation is bound between `MIN_LIQUIDATION_INCENTIVE` and `MAX_LIQUIDATION_INCENTIVE`.	**Passed**
LIQ-6	If `cFactor` is `0`, `maxAmount` is `0`.	**Passed**
LIQ-7	If `cFactor` is equal to `WAD`, `maxAmount` is equal to `debtBalanceCached`.	**Passed**
LIQ-8	If `cFactor` is noninclusively bound between `0` and `WAD`, `maxAmount` is bound between `0` and `debtBalanceCached`.	**Passed**
LIQ-9	If the collateral token has fewer decimals than the debt token, `amountAdjusted` is less than the debt balance.	**Passed**
LIQ-10	If the collateral token has more decimals than the debt token, `amountAdjusted` is greater than `debtBalanceCached`.	**Passed**
LIQ-11	If the collateral token has fewer decimals than the debt token, `amountAdjusted` is less than	**Passed**

	`debtBalanceCached`.	
LIQ-12	If `amountAdjusted` is `0`, the number of tokens to be liquidated is `0`.	**Passed**
LIQ-13	If `debtToCollateralRatio` is `0`, the number of tokens to be liquidated is `0`.	**Passed**

DToken Functional Invariants

These invariants check preconditions and postconditions specific to the `DToken` contract, including those for the borrowing, token repayment, and soft liquidation functions.

ID	Property	Result
DTOK-1	Calls to `DToken.mint` with the correct preconditions succeed.	**Passed**
DTOK-2	The sender's underlying dToken balance decreases by `amount` after minting dTokens.	**Passed**
DTOK-3	The balance of the recipient after minting dTokens increases by the result of `amount * WAD / exchangeRateCached`.	**Passed**
DTOK-4	The dToken `totalSupply` increases by the result of `amount * WAD / exchangeRateCached` after calls to `DToken.mint`.	**Passed**
DTOK-5	Calls to the `borrow` function with the proper preconditions succeed when the dToken market has not accrued interest.	**Passed**
DTOK-6	If interest has not accrued in the dToken market, `totalBorrows` increases after calls to `borrow`.	**Passed**
DTOK-7	If interest has not accrued in the dToken market, the underlying balance of the caller increases by `amount`.	**Passed**
DTOK-8	Calls to the `borrow` function with the proper preconditions succeed when the dToken market is accruing interest.	**Passed**
DTOK-9	If interest has accrued in the dToken market, `totalBorrows` increases by the amount accrued.	**Passed**
DTOK-10	If interest has accrued in the dToken market, the balance of the underlying asset increases by the amount accrued.	**Passed**
DTOK-11	Calls to `repay` with an `amount` value that is too high error	**Passed**

	out gracefully.	**(8c9a2dd)**
		Failed (TOB-CURV-3)
DTOK-12	Calls to `repay` with the proper preconditions succeed.	**Further Investigation Required (TOB-CURV-13)**
DTOK-13	Calls to `repay` with any `amount` value when no interest is accruing in the dToken market make `totalBorrows` equal to the difference of `preTotalBorrows` and `amount`.	**Passed**
DTOK-14	Calls to `repay` with an `amount` value of `0` cause the caller's `accountDebt` to zero out.	**Passed**
DTOK-15	Users can repay between zero and the value of their `accountDebt` with the `repay` function.	**Passed**
DTOK-16	Calls to `repay` with an `amount` value of `0` result in an underlying balance of debt that is equal to the previous underlying balance minus the account debt.	**Passed**
DTOK-17	Calls to `repay` when the dToken market is accruing interest make `totalBorrows` equal to the result of `totalBorrows - preTotalBorrows - amount - (|new_exchange_rate - old_exchange_rate| * accountDebt)`.	**Passed**
DTOK-18	Calls to the `mint` function when depositing into the `GaugePool` contract revert if the result of `amount * WAD / exchangRate` is `0`.	**Passed**
DTOK-19	Soft liquidations of exactly zero tokens fail with the `InvalidAmount` or `InvalidParameter` error.	**Failed (TOB-CURV-10)**
DTOK-20	Liquidations of a non-exact amount remove the user's position in the collateral token.	**Passed**

DTOK-21	Liquidations of a non-exact amount zero out the collateral posted for the user in the collateral token.	**Passed**
DTOK-22	Liquidations of a non-exact amount zero out the debt balance of the respective debt token.	**Passed**
DTOK-23	Liquidations of a non-exact amount decrease the collateral balance of the account.	**Failed** (TOB-CURV-11)
DTOK-24	Liquidations of a non-exact amount decrease the liquidator's underlying `dTokenBalance` by `debtToLiquidate`.	**Passed**
DTOK-25	Liquidations of a non-exact amount increase the collateral token balance by the amount seized for the liquidation minus the amount seized for the protocol.	**Failed** (TOB-CURV-12)
DTOK-26	Liquidations of an exact amount result in a difference of `priorCollateral` and `currentCollateral` that is equal to the amount seized for the liquidation.	**Passed**
DTOK-27	Liquidations of an exact amount result in a decrease of the account debt by `debtToLiquidate`.	**Passed**
DTOK-28	Liquidations of an exact amount result in an underlying token balance of the `msg.sender` after liquidation that is equal to the previous underlying balance plus the debt to liquidate.	**Passed**
DTOK-29	Liquidations of an exact amount result in an increase of the collateral token balance of the sender by the amount seized by the liquidation minus the amount seized by the protocol.	**Passed**

DToken System Invariants

These invariants check higher-level relationships between token balances and internal accounting slots in the `DToken` contract, as well as the contract's getter functions to ensure that they behave correctly.

ID	Property	Result
S-DTOK-1	The `marketUnderlyingHeld` value for a dToken is equal to the underlying asset's `balanceOf` value.	**Passed**
S-DTOK-2	The number of decimals for a dToken is equal to the number of decimals for the underlying token.	**Passed**
S-DTOK-3	The `isCToken` function for a dToken does not return `true`.	**Passed**

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	All arithmetic in the codebase is thoroughly documented, and underlying assumptions for unchecked blocks are clearly identified. Areas in which rounding is problematic are clearly identified and described in the codebase. Stateless fuzz testing and unit testing have targeted some of these arithmetic paths; however, the stateful fuzz testing suite found a few obscure bugs in functions, which warrant additional investigation. If Curvance invests further effort into expanding the current stateful fuzz testing suite, the rating for this category could be satisfactory.	**Moderate**
Auditing	All functions emit events where appropriate, making it easy to identify when functions are executed. The documentation in the codebase states when events are to be used. Data validation checks are added throughout the codebase to ensure that users know the cause of transaction failures. Curvance is working on an incident response and monitoring plan, which was not reviewed during this engagement.	**Satisfactory**
Authentication / Access Controls	All targeted contracts (VeCVE- and `MarketManager`-related) have access controls on the functions that set system variables to “paused” to mitigate ongoing attacks. These functions can be called only by authorized administrators of the `centralRegistry`, where users are set. There is clear NatSpec documentation on privileged functions describing the abilities the privileged access can grant. According to Curvance, the use of EOAs in this system is limited; it is assumed to be controlled through the DAO.	**Satisfactory**

Complexity Management	At the start of this engagement, functions such as those for combining all locks and updating collateral values were very hard to understand. After much refactoring due to issues raised during the invariant development process and additional feedback we gave on code readability, these functions were heavily simplified, and the `combineLocks` function was removed. Similar changes were made in the `MarketManager`-related contracts, which have evolved significantly as well. There is some redundant logic in arithmetic in some functions, such as `VeCVE`-related functions that manipulate user points; this logic was not pulled into helpers due to contract size. The codebase follows a consistent pattern of the use of underscores to prefix internal functions and the use of camel case, making the code much more readable.	**Satisfactory**
Decentralization	The codebase uses multiple addresses to update and put the system into critical modes and functionalities. A single entity is not in control of all user funds, and access is equally split among different addresses. While system parameters can be changed, Curvance has added more data validation around the limits of those parameters throughout the engagement, such as checks to ensure the collateral ratio cannot be set to `0` if it is already set to a nonzero value. We recommend that Curvance dedicate extra effort into documenting the deployment risks and assumptions of the codebase, and any risks that are inherited from the integration of other protocols. During this invariant development and testing engagement, the risk level of integration with other protocols was not investigated.	**Moderate**
Documentation	At the beginning of the engagement, the documentation on functions was lacking, and it was sometimes unclear what certain functions were doing. Midway through the engagement, the NatSpec documentation on all components of the system was heavily improved, and it now provides much better readability and understanding of the system. We do recommend, however, creating external documentation to help identify system flows and assumptions made by the codebase. We also recommend performing additional	**Satisfactory**

	arithmetic-focused white paper analysis on all formulas in the codebase to ensure that rounding considerations found in this review and through the fuzz testing suite are investigated thoroughly.	
Low-Level Manipulation	The codebase uses a significant amount of assembly, especially for gas-optimized reverts and error messages. All areas of this code are documented heavily. Curvance uses the low-level implementation of Solady for transfers. These areas were not considered in scope for this engagement. While the unit tests implicitly test these cases, a differential fuzzing implementation is missing for contracts containing assembly to make sure that all instances of assembly are consistent with their expected behavior.	**Moderate**
Testing and Verification	The Curvance codebase contains a significant number of unit and integration tests (900 total) and stateless fuzz tests; the test coverage has increased significantly during the engagement. However, due to the complexity of system interactions and functions, we recommend that Curvance continue to expand the existing unit, integration, and stateful fuzz tests.	**Moderate**
Transaction Ordering	Transaction reordering risks were not tested or investigated during this engagement with the `VeCVE` and `MarketManager` components.	**Further Investigation Required**

Summary of Findings

The table below summarizes the findings that resulted from the testing exercise, including their severity ratings and the associated testing tool.

ID	Title	Tool	Severity
1	Combining continuous locks into single continuous lock terminal results in 1 wei in profit	Echidna, Medusa	Undetermined
2	Combining some noncontinuous locks into single continuous lock terminal does not change userPoints	Echidna, Medusa	Undetermined
3	Calls to repay with excessive amount result in underflow and panic	Echidna, Medusa	Low
4	processExpiredLock called with the relock option does not delete the existing lock	Echidna	High
5	Combining locks is still possible after the system is shut down	Echidna	Low
6	combineAllLocks erroneously decreases user points when used with expired lock	Medusa	Medium
7	repayWithBadDebt can be 1 wei off and cause a panic	Echidna	Medium
8	Possible underflow in combineAllLocks due to 1-wad difference in veCVE balance and user points	Echidna	Undetermined
9	Negative prices from OracleRouter cause underflow and panic	Medusa	Undetermined
10	Division-by-zero error in _canLiquidate results in a panic	Echidna	Undetermined
11	Missing validation allows the DAO address to be liquidated	Echidna	Undetermined

12	Missing validation allows the DAO address to be the liquidator	**Echidna**	**Undetermined**
13	The repay function will panic if a user's total borrows and debt balance are 1 wei off	**Echidna**	**Undetermined**

Detailed Findings

1. Combining continuous locks into single continuous lock terminal results in 1 wei in profit

Severity: **Undetermined**

Tool: Echidna, Medusa	
Invariant ID: VECVE-4	Finding ID: TOB-CURV-1
Target: `contracts/token/VeCVE.sol`	

Description

When users combine multiple continuous locks into a single continuous lock terminal, the contract gives users 1 wei in profit.

This is fundamentally because the following invariant is broken:

> The value of `finalLockAmount` (from `_getCLPoints`) minus the sum of the amount in each lock is equal to `finalLockAmount` (from `_getCLPoints`) minus `finalLockAmount`.

When a user has multiple locks created in the system, they can use the `combineAllLocks` function to combine individual locks into a single lock. A lock can have a state of "continuous" (which is locked indefinitely and affords users more points) or "noncontinuous" (which is locked for a specific duration of time). The expected system behavior under these conditions differ when the two lock types are combined.

The VeCVE contract has a concept of "user points," which track the number of points/assets a user has in the lifetime of the system. If all prior locks are continuous, the user points are not expected to change after the combining of locks; otherwise, a user could profit from combining existing locks. Testing this property with Echidna and Medusa confirmed that it is broken.

```
3) FuzzingSuite.combineAllLocks_should_succeed_to_continuous_terminal()
[...]
        => [event] AssertEqFail("Invalid: 6447388898!=6447388899, reason: VE_CVE -
combineAllLocks() - all prior continuous => continuous failed")
        => [panic: assertion failed]
```

Figure 1.1: A snippet of the failure stack in the test for this `combineAllLocks` invariant

Both fuzzers found that the following combination of transactions breaks the invariant that user points do not change. We ported this code to a Foundry test to check its exploitability. The test creates two continuous locks for a single user, combines them, and then asserts equivalence on the pre- and post-combined `userPoints` value.

```
function test_combineAllLocks_correct_user_point_value_with_continuous_lock(
    bool shouldLock,
    bool isFreshLock,
    bool isFreshLockContinuous
) public setRewardsData(shouldLock, isFreshLock, isFreshLockContinuous) {
    veCVE.createLock(1595215587, true, rewardsData, "", 0);
    veCVE.createLock(4266047049, true, rewardsData, "", 0);
    uint256 preCombine = (veCVE.userPoints(address(this)));

    veCVE.combineAllLocks(false, rewardsData, "", 0);

    uint256 postCombine = (veCVE.userPoints(address(this)));
    assertEq(preCombine, postCombine);
}
```

Figure 1.2: The Foundry test checking the pre- and post-combined `userPoints`

The full call sequences are provided in appendix D.

Property Status: Passing
After the changes made in commit `13d350`, the property is now passing after 96 hours with Medusa and 10,000,000 runs with Echidna.

2. Combining some noncontinuous locks into single continuous lock terminal does not change userPoints

Severity: **Undetermined**	
Tool: Echidna, Medusa	
Invariant ID: VECVE-10	Finding ID: TOB-CURV-2
Target: `contracts/token/VeCVE.sol`	

Description

When some noncontinuous locks are combined into a single continuous lock, the user points are expected to increase. This is because the user should be accruing *more* points now that all of their locks are continuous.

However, Echidna and Medusa found multiple instances in which user points do not change when a user combines some noncontinuous locks into a continuous lock terminal. This means that when users expect an increase in their points reward, they will not be given one.

```
3)
FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed()
(addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0,
sender=0x0000000000000000000000000000000000030000)
[...]
        => [event] AssertLtFail("Invalid: 5>=5 failed, reason: VE_CVE -
combineAllLocks() - some prior continuous => continuous failed")
        => [panic: assertion failed]
```

Figure 2.1: A snippet of of the failure stack in the test for this `combineAllLocks` invariant

Due to an incorrect rounding direction in an associated calculation, the `combineAllLocks` function does not return `userPoints` data in the expected bounds.

This is fundamentally because the following invariant is broken:

> The value of `finalLockAmount` (from `_getCLPoints`) minus the sum of the amount in each lock is equal to `finalLockAmount` (from `_getCLPoints`) minus `finalLockAmount`.

After rebasing changes made to `VeCVE.sol`, these properties no longer fail on extended runs.

Property Status: Passing
After the changes made in commit `13d350`, the property is now passing after 96 hours with Medusa and 10,000,000 runs with Echidna.

3. Calls to repay with excessive amount result in underflow and panic

Severity: **Low**

Tool: Echidna, Medusa

Invariant ID: DTOK-11	Finding ID: TOB-CURV-3

Target: `DToken.sol`

Description

There is no validation of the `amount` value passed to the `repay` function; if the user attempts to repay too much, the function will panic (`0x11`) due to arithmetic underflow.

When users repay debt back to the protocol, they specify the amount that they would like to repay.

```
/// @notice Caller repays their own debt
/// @dev    Updates interest before executing the repayment
/// @param amount The amount to repay, or 0 for the full outstanding amount
function repay(uint256 amount) external nonReentrant {
    accrueInterest();

    _repay(msg.sender, msg.sender, amount);
}
```

Figure 3.1: The `repay` function

The `canRepay` function (called by the `_repay` helper function) checks only that the token is properly listed in the system and that the `MINIMUM_HOLD_PERIOD` has passed before the repay attempt. There is no validation to ensure that the value of `amount` does not exceed the value of `accountDebt`; if it does, the calculation of the principal amount (highlighted in yellow in figure 3.1) will underflow and cause a panic.

```
/// @dev First validates that the payer is allowed to repay the loan, then repays
///      the loan by transferring in the repay amount. Emits a repay event on
///      successful repayment.
/// @param payer The address paying off the borrow
/// @param account The account with the debt being paid off
/// @param amount The amount the payer wishes to repay, or 0 for the full
outstanding amount
/// @return The actual amount repaid
function _repay(
    address payer,
```

```
    address account,
    uint256 amount
) internal returns (uint256) {
    // Validate that the payer is allowed to repay the loan
    lendtroller.canRepay(address(this), account);

    // Cache how much the account has to save gas
    uint256 accountDebt = debtBalanceCached(account);

    // If amount == 0, amount = accountDebt
    amount = amount == 0 ? accountDebt : amount;

    SafeTransferLib.safeTransferFrom(
        underlying,
        payer,
        address(this),
        amount
    );

    // We calculate the new account and total borrow balances,
    // failing on underflow:
    _debtOf[account].principal = accountDebt - amount;
    _debtOf[account].accountExchangeRate = marketData.exchangeRate;
    totalBorrows -= amount;

    emit Repay(payer, account, amount);
    return amount;
}
```

Figure 3.2: The `_repay` function

Echidna broke this invariant by calling `repay` with an extremely large amount (one that largely exceeds a user's debt balance). This resulted in a panic, as highlighted in the code snippet below. The error reverts with `Panic(17)` (in decimal form, which when converted to hex, maps to `Panic(0x11)` for an integer underflow error).

```
repay_should_succeed(address,uint256): failed!💥
  Call sequence, shrinking 473/5000:
    list_token_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1)

repay_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1,84196168906531442932
019281574963489722385677050734900937838003259005819529 94)

Traces:
call
0x600515dFe465f600f0c9793FA27Cd2794F3eC0e1::repay(84196168906531442932019281574963489
722385677050734900937838003259005819529 94)
(curvance-contracts/tests/fuzzing/FuzzDToken.sol:254)
 ├─call
0x83d85eEB38A2dC37EAc0239c19b343a7653d8F79::canRepay(@0x600515dFe465f600f0c9793FA27C
d2794F3eC0e1, @0x00a329c0648769A73afAc7F9381E08FB43dBEA72) <no source map>
 |  └─← 0x
```

```
 ├ call
0xee35211C4D9126D520bBfeaf3cFee5FE7B86F221::transferFrom(@0x00a329c0648769A73afAc7F9
381E08FB43dBEA72, @0x600515dFe465f600f0c9793FA27Cd2794F3eC0e1,
84196168906531442932019281574963489722385677050734900937838003259005819529 94) <no
source map>
 │
├ Transfer(84196168906531442932019281574963489722385677050734900937838003259005819952
994) (curvance-contracts/contracts/libraries/ERC20.sol:273)
 │  └ ← (true)
└ error Revert Panic(17) <no source map>
AssertFail(«DTOKEN - repay should succeed with correct preconditions»)
(curvance-contracts/tests/fuzzing/PropertiesHelper.sol:20)
```

Figure 3.3: The shortened Echidna trace for the repay function

Property Status: Passing

After the changes made in commit 8c9a2dd, the property is now passing after 96 hours with Medusa and 100,000,000,000 runs with Echidna.

4. processExpiredLock called with the relock option does not delete the existing lock

Severity: **High**	
Tool: Echidna	
Invariant ID: VECVE-18, VECVE-55	Finding ID: TOB-CURV-4
Target: `contracts/token/VeCVE.sol`	

Description

The `processExpiredLock` function is intended to replace an existing lock with a new lock with an updated timestamp. However, in a certain case, `processExpiredLock` currently creates a new lock but does not delete the old one. Attackers could use this issue to overinflate their voting power. The issue also causes additional Echidna failures in the rest of the test suite.

The `processExpiredLock` function has separate logic for when users decide to relock their funds. If they do, the function appends a new lock without removing the existing one. As a result, the user will have one more lock than they should have in the contract.

```
function processExpiredLock(
    uint256 lockIndex,
    bool relock,
    bool continuousLock,
    RewardsData calldata rewardsData,
    bytes calldata params,
    uint256 aux
) external nonReentrant {
    Lock[] storage locks = userLocks[msg.sender];

    // Length is index + 1 so has to be less than array length.
    if (lockIndex >= locks.length) {
        _revert(_INVALID_LOCK_SELECTOR);
    }

    if (block.timestamp < locks[lockIndex].unlockTime && isShutdown != 2) {
        _revert(_INVALID_LOCK_SELECTOR);
    }

    // Claim any pending locker rewards.
    _claimRewards(msg.sender, rewardsData, params, aux);

    Lock memory lock = locks[lockIndex];
```

```
    uint256 amount = lock.amount;

    // If the locker is shutdown, do not allow them to relock,
    // we'd want them to exit locked positions.
    if (isShutdown == 2) {
        relock = false;
        // Update their points to reflect the removed lock
        _updateDataFromEarlyUnlock(msg.sender, amount, lock.unlockTime);
    }

    if (relock) {
        // Token points will be caught up by _claimRewards call so we can
        // treat this as a fresh lock and increment rewards again.
        _lock(msg.sender, amount, continuousLock);
    } else { // BUG? user points is not changed if wants to relock
        _burn(msg.sender, amount);
        _removeLock(locks, lockIndex);

        // Transfer the user the unlocked CVE
        SafeTransferLib.safeTransfer(cve, msg.sender, amount);

        emit Unlocked(msg.sender, amount);

        // Check whether the user has no remaining locks and reset their
        // index, that way if in the future they create a new lock,
        // they do not need to claim epochs they have no rewards for.
        if (locks.length == 0 && isShutdown != 2) {
            cveLocker.resetUserClaimIndex(msg.sender);
        }
    }
}
```

Figure 4.1: The `processExpiredLock` function

One of our invariants tested calling `processExpiredLock` on a lock that had expired and had the relock option enabled while the system was not shut down. The assertion in this invariant checked whether the number of locks the user had in their account changed. This immediately failed when run with Echidna; the number highlighted in yellow in figure 4.2 identifies the number of locks the user had before calling the function, and the number highlighted in red identifies the number of locks the user had after. Noticeably, there is a difference between these values because the `processExpiredLock` function appends the lock instead of replacing the existing lock with it.

```
processExpiredLock_should_succeed_if_unlocktime_expired_and_not_shutdown_with_relock
(): failed!💥
  Call sequence:
    CVE_is_deployed() Time delay: 30764697 seconds Block delay: 14141

processExpiredLock_should_succeed_if_unlocktime_expired_and_not_shutdown_with_relock
()
```

```
Traces:
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:788)
 └─← (1)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::0x9521e5bb0000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000001000000000000000000000000000000000000000000000000000000000000000001000000
0000000000000000062d69f6867a0a084c6d313943dc22023bc263691000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000120000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000 (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:810)
 ├─call
0x0A64DF94bc0E039474DB42bb52FEca0c1d540402::epochsToClaim(@0x00a329c0648769A73afAc7F
9381E08FB43dBEA72) <no source map>
 |  └─← (0)
 ├─Transfer(100000000000000000) <no source map>
 └─← 0x
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1260)
 └─← (2)

AssertEqFail(«Invalid: 1!=2, reason: VE_CVE - when relocking, the number of locks
should be equivalent»)
(curvance-contracts/tests/fuzzing/helpers/PropertiesHelper.sol:45)
```

Figure 4.2: The `processExpiredLock` assertion failure

This invariant also caused VECVE-18 to fail (TOB-CURV-6) because the total number of user points calculated did not match the actual CVE balance.

```
combineAllLocks_should_succeed_to_non_continuous_terminal(): failed!💥
  Call sequence:

processExpiredLock_should_succeed_if_unlocktime_expired_and_not_shutdown_no_relock()
    create_lock_when_not_shutdown(0,false)
    create_lock_when_not_shutdown(0,false)
    combineAllLocks_should_succeed_to_non_continuous_terminal()

Traces
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1256)
 └─← (2)
```

```
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userPoints(@0x00a329c0648769A73afAc7F938
1E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:309)
 └← (3000000000000000000)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::0x1bb1521a000000000000000000000000000000
0000000000000000000000000000000000000000000000000062d69f6867a0a084c6d313943d
c22023bc263691000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000e000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:316)
 ├ call
0x0A64DF94bc0E039474DB42bb52FEca0c1d540402::epochsToClaim(@0x00a329c0648769A73afAc7F
9381E08FB43dBEA72) <no source map>
 |  └← (0)
 └← 0x
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userPoints(@0x00a329c0648769A73afAc7F938
1E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:327)
 └← (3000000000000000000)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::balanceOf(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:349)
 └← (2000000000000000000)
AssertEqFail(«Invalid: 2000000000000000000!=3000000000000000000, reason: VE_CVE -
combineAllLocks() balance should equal post combine user points»)
(curvance-contracts/tests/fuzzing/helpers/PropertiesHelper.sol:45)
```

Figure 4.3: The `combineAllLocks` assertion failure after `processExpiredLock` adds one too many locks

Note that the failure of this property also initially resulted in multiple additional failures and complex call sequences, which are further identified in appendix E.

Property Status: Passing
After the changes made in commit `13d350`, the property is now passing after 96 hours with Medusa and 10,000,000 runs with Echidna.

5. Combining locks is still possible after the system is shut down

Severity: **Low**

Tool: Echidna

Invariant ID: VECVE-56	Finding ID: TOB-CURV-5

Target: `contracts/token/VeCVE.sol`

Description

Users can still combine locks when the system is shut down, which should not technically be possible, as it would allow user points and rewards to continue to change during a system shutdown.

```
combineAllLocks_called_when_shutdown_should_revert(): failed!💥
  Call sequence:
    create_lock_when_not_shutdown(0,false)
    shutdown_success_if_elevated_permission()
    combineAllLocks_called_when_shutdown_should_revert()

Traces:
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1256)
 └─ ← (2)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::isShutdown()
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:137)
 └─ ← (2)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::0x1bb1521a0000000000000000000000000000
0000000000000000000000000000000100000000000000000000062d69f6867a0a084c6d313943d
c22023bc263691000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000e0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:139)
 ├─ call
0x0A64DF94bc0E039474DB42bb52FEca0c1d540402::epochsToClaim(@0x00a329c0648769A73afAc7F
```

```
9381E08FB43dBEA72) <no source map>
 |  └← (0)
 └← 0x
AssertFail(«VE_CVE - combine all locks when shut down should not be possible»)
(curvance-contracts/tests/fuzzing/helpers/PropertiesHelper.sol:20)
```

Figure 5.1: The shutdown assertion failure in Echidna

Property Status: Passing
After the changes made in commit `13d350`, the property is now passing after 96 hours with Medusa and 10,000,000 runs with Echidna.

6. combineAllLocks erroneously decreases user points when used with expired lock

Severity: **Medium**	
Tool: Medusa	
Invariant ID: VECVE-18	Finding ID: TOB-CURV-6
Target: `contracts/token/VeCVE.sol`	

Description

The `combineAllLocks` function fails to account for expired locks. As a result, if a user combines locks that include an expired lock into a noncontinuous lock terminal, they will have a mismatch in their user points and veCVE balance.

The following proof of concept written in Foundry creates multiple locks, warps time multiple times so that one of the locks expires, and eventually calls `veCVE.combineAllLocks` to combine the created locks into a noncontinuous terminal. As the final lock is noncontinuous, the user is no longer supposed to receive additional multiplier rewards; therefore, their number of user points should be equal to their veCVE balance. This test fails because the `combineAllLocks` function accounts for the expired lock, thereby decreasing the number of user points, causing it to fall out of sync with the veCVE balance.

```
function test_combine_all_locks_to_non_continuous_terminal_one_wad() public
{
    RewardsData memory emptyRewards = RewardsData(address(0),false,false,false);
    setUp();
    veCVE.createLock(1e18, false, emptyRewards, "", 0);
    // warp time
    vm.warp(block.timestamp + 29774646);
    veCVE.createLock(1e18, false, emptyRewards, "", 0);

    vm.warp(block.timestamp + 986121 + 675582 + 1000);
    veCVE.processExpiredLock(0, false, false, emptyRewards, "", 0);
    assertEq(veCVE.userPoints(address(this)), 2000000000000000000);
    assertEq(veCVE.queryUserLocksLength(address(this)),1);

    veCVE.createLock(1000000024330411045, false, emptyRewards, "", 0);
```

```
    veCVE.combineAllLocks(false, emptyRewards, "", 0);

    assertEq(veCVE.userPoints(address(this)), veCVE.balanceOf(address(this)));

}
```

Figure 6.1: The `combineAllLocks` proof of concept in Foundry

After manually shrinking Medusa's call sequence for testing this invariant (as it is 87 sequences long) to the final transaction, we can see a 1e18 difference between the respective user's user points and veCVE balance.

```
87) FuzzingSuite.combineAllLocks_should_succeed_to_non_continuous_terminal()
(block=2689116, time=31886963, gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000020000)
[Execution Trace]
 => [call] FuzzingSuite.combineAllLocks_should_succeed_to_non_continuous_terminal()
(addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0,
sender=0x0000000000000000000000000000000000020000)
[...]
        => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (33153376269137521925)]
        => [call] VeCVE.balanceOf(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (32153376269137521925)]
        => [event] AssertEqFail("Invalid: 32153376269137521925!=33153376269137521925,
reason: VE_CVE - combineAllLocks() balance should equal post combine user points")
        => [panic: assertion failed]
```

Figure 6.2: The assertion failure in Medusa

Property Status: Passing

After the changes made in commit 8c9a2dd, the property is now passing after 96 hours with Medusa and 10,000,000 runs with Echidna.

7. repayWithBadDebt can be 1 wei off and cause a panic

Severity: **Medium**

Tool: Echidna

Invariant ID: MARKET-35, MARKET-42	Finding ID: TOB-CURV-7

Target: `contracts/market/collateral/DToken.sol`

Description

The `liquidateAccount` function calls `accrueInterest` to calculate the interest before other calculations are done. The interest calculation updates the values of the debt accumulated and exchange rates, resulting in a `debtBalanceCached` value that is 1 wei greater than `totalBorrows`. Due to the use of native Solidity arithmetic, this causes an underflow in the result of the calculation highlighted in red in figure 7.1, which appears in the `repayWithBadDebt` function, called by `liquidateAccount`.

```
function repayWithBadDebt(
    address liquidator,
    address account,
    uint256 repayRatio
) external nonReentrant {
    // We check self liquidation in marketManager before
    // this call, so we do not need to check here.

    // Make sure the marketManager itself is calling since
    // then we know all liquidity checks have passed.
    if (msg.sender != address(marketManager)) {
        _revert(_UNAUTHORIZED_SELECTOR);
    }

    // We do not need to check for interest accrual here since its done
    // at the top of liquidateAccount, inside market manager contract,
    // that calls this function.

    // Cache account debt balance to save gas.
    uint256 accountDebt = debtBalanceCached(account);
    uint256 repayAmount = (accountDebt * repayRatio) / WAD;

    // We do not need to check for listing here as we are
    // coming directly from the marketManager itself,
    // so we know it is listed.

    // Process a partial repay directly by transferring underlying tokens
```

```
    // back to the dToken contract.
    SafeTransferLib.safeTransferFrom(
        underlying,
        liquidator,
        address(this),
        repayAmount
    );

    // Wipe out the accounts debt since we are recognizing
    // unpaid debt as bad debt.
    delete _debtOf[account].principal;
    totalBorrows -= accountDebt;

    emit Repay(liquidator, account, repayAmount);
    emit BadDebtRecognized(liquidator, account, accountDebt - repayAmount);
}
```

Figure 7.1: The `repayWithBadDebt` function

The following failure was identified by Echidna; under correct liquidation preconditions, the `liquidateAccount` function should have succeeded. However, Echidna found a violation of this property.

```
liquidateAccount_should_succeed(uint256): failed!💥
[...]

0x093514489C4b42Ff54f942f4F91De3F89c797aAb::liquidateAccount(@0x00a329c0648769A73afA
c7F9381E08FB43dBEA72) curvance-contracts/tests/fuzzing/FuzzMarketManager.sol:1007)
 |  |  └error Revert Panic(17) <no source map>
 |  └error Revert Panic(17) <no source map>
 └error Revert Panic(17)
AssertFail(«MARKET-35 liquidateAccount with correct preconditions should succeed»)
```

Figure 7.2: The `liquidateAccount` Echidna failure

By pruning and shortening the failure callstack and pulling out the pertinent values, we found that in a subset of conditions, `totalBorrows` is 1 wei less than `accountDebt`, due to rounding.

```
>>> interestAccumulated = 2701176
>>> borrowsPrior = 11509777705484
>>> debtAccumulated =
(interestAccumulated * borrowsPrior) /
1e18
>>> debtAccumulated
31.08993530338845
>>> totalBorrowsNew = debtAccumulated +
```

```
>>> principal = 11509777705485
>>> exchangeRate = 1000000000316734276
>>> accountExchangeRate =
1000000000314033100
>>> debtBalanceCached = (principal *
exchangeRate) / accountExchangeRate
>>> debtBalanceCached
11509777705516.09
```

```
borrowsPrior
>>> totalBorrowsNew
11509777705515.09
```

Figure 7.3: The numbers resulting in a 1-wei off error

Property Status: Further Investigation Required
After further investigation, Curvance determined that even though the rounding introduces a 1-wei difference between `totalBorrows` and `debtBalance`, all values round in the correct direction.

As a result, the fuzz testing suite now checks against a 1-wei difference and will throw an assertion failure if this difference is more than 1 wei (figure 7.4). Echidna has determined that the difference can be at least 2 wei; however; this range and its further implications should be further explored.

```
function liquidateAccount_should_succeed(uint256 amount) public {
    require(marketManager.seizePaused() != 2);
    address account = address(this);
    amount = _preLiquidate(amount, DAI_PRICE, USDC_PRICE);

    IMToken[] memory assets = marketManager.assetsOf(account);

    hevm.prank(msg.sender);
    try this.prankLiquidateAccount(account) {
        [...]
    } catch Panic(uint256 errorCode) {
        if (errorCode == PANIC_UNDER_OVER_FLOW_CODE) {
            for (uint256 i = 0; i < assets.length; i++) {
                if (assets[i].isCToken()) {
                    continue;
                }
                uint256 totalBorrows = IMToken(assets[i]).totalBorrows();
                uint256 accountDebt = IMToken(assets[i]).debtBalanceCached(
                    address(this)
                );
                if (totalBorrows < accountDebt) {
                    emit LogUint256(
                        "difference between totalBorrows and accountDebt",
                        accountDebt - totalBorrows
                    );
                    // The system has a *known* limitation in rounding that
totalBorrows and accountDebt can be off by one wei
                    // This check ensures that if there is a diff, it must be no
more than 1 wei, otherwise Echidna will throw
                    assertEq(
                        accountDebt - totalBorrows,
                        1,
                        "MARKET-42 - difference between accountdebt and totalborrows
exceeds 1"
```

```
                    );
                }
            }
        } else {
            emit LogUint256("panic code:", errorCode);
            assertWithMsg(
                false,
                "MARKET-35 liquidateAccount panicked unexpectedly"
            );
        }
    } catch (bytes memory revertData) {
        uint256 errorSelector = extractErrorSelector(revertData);

        assertWithMsg(
            false,
            "MARKET-35 liquidateAccount with correct preconditions should succeed"
        );
    }
}
```

Figure 7.4: The `liquidateAccount_should_succeed` function

8. Possible underflow in combineAllLocks due to 1-wad difference in veCVE balance and user points

Severity: **Undetermined**	
Tool: Echidna	
Invariant ID: VECVE-17	Finding ID: TOB-CURV-8
Target: `contracts/token/VeCVE.sol`	

Description

The VeCVE contract has a concept of "user points," which track the number of points/assets a user has in the lifetime of the system; the contract also tracks users' CVE and veCVE balance. In some cases, combining a supposedly valid sequence of locks causes an underflow in arithmetic related to these values in the VeCVE contract.

The following Foundry test reproduces the underflow:

```
function test_combine_all_locks_underflow() public {
    setUp();
    RewardsData memory emptyRewards = RewardsData(
        false,
        false,
        false,
        false
    );
    veCVE.createLock(1e18, false, emptyRewards, "", 0);

    vm.warp(block.timestamp + 1668393);
    veCVE.createLock(1159307271353364048, false, emptyRewards, "", 0);

    vm.warp(block.timestamp + 15388973);

    veCVE.increaseAmountAndExtendLock(
        12337192967826718984,
        1,
        false,
        emptyRewards,
        "",
        0
    );

    vm.warp(block.timestamp + 24537335);
    // this is when lock at index 0 becomes continuous
```

```
        veCVE.processExpiredLock(0, true, true, emptyRewards, "", 0);

        // this is when lock at index 1 becomes continuous
        veCVE.increaseAmountAndExtendLock(
            18217003917551520407,
            1,
            true,
            emptyRewards,
            "",
            0
        );

        veCVE.combineAllLocks(true, emptyRewards, "", 0);
    }
```

Figure 8.1: The Foundry test for reproducing the `combineAllLocks` underflow

The failure was traced down to the line in VeCVE highlighted in red in figure 8.2; the value of `veBalanceOf` is expected to be strictly greater than or equal to the value of `currentPoints`. This invariant does not hold when there is a 1-wad difference between the two values.

```
/// @notice Combines all locks into a single lock,
///         and processes any pending locker rewards.
/// @param continuousLock Whether the combined lock should be continuous
///                       or not.
/// @param rewardsData Rewards data for CVE rewards locker.
/// @param params Parameters for rewards claim function.
/// @param aux Auxiliary data.
function combineAllLocks(
    bool continuousLock,
    RewardsData calldata rewardsData,
    bytes calldata params,
    uint256 aux
) external nonReentrant {
    [...]

        // Multiply balanceOf by CL Multiplier since terminal lock
        // is continuous, and terminal points should be multiplied
        // above their veCVE balance.
        veBalanceOf = veBalanceOf * CL_POINT_MULTIPLIER;

        // Check if points need to be adjusted, true when there are
        // non-continuous locks currently so points need to increase.
        if (veBalanceOf != currentPoints) {
            _incrementPoints(msg.sender, veBalanceOf - currentPoints);
        }

        // Return without updating token unlocks since the terminal
        // lock is continuous.
        return;
    }
```

Figure 8.2: The underflow error in the `combineAllLocks` function

We pruned and shortened the failure callstack. Figure 8.3 shows the output from Echidna reporting the assertion failure:

```
  ├─ LogUint256(«vebalance of:», 76230009609125182414) <no source map>
  ├─ LogUint256(«current points», 77230009609125182414) <no source map>
  └─ error Revert Panic(17) <no source map>
AssertFail(«VE_CVE - combineAllLocks() failed unexpectedly with correct
preconditions»)
```

Figure 8.3: The assertion failure from Echidna reporting that `veBalanceOf` does not match `currentPoints`

Figure 8.4 shows the output with the assertion failure from the Foundry test shown in figure 8.1:

```
|   ├─ emit LogUint256(: "vebalance of:", : 32713504156731603439 [3.271e19])
|   ├─ emit LogString(: "veBalanceOf != current points, attempting to increment
points")
|   ├─ emit LogUint256(: "vebalance of:", : 65427008313463206878 [6.542e19])
|   ├─ emit LogUint256(: "curent points", : 66427008313463206878 [6.642e19])
|   └─ ← panic: arithmetic underflow or overflow (0x11)
└─ ← panic: arithmetic underflow or overflow (0x11)
```

Figure 8.4: The assertion failure from Foundry reporting the underflow revert

Property Status: Failing

Curvance has decided to introduce a blackout period for the claiming of rewards, which will prevent this particular scenario from occurring. Implementing this blackout period will require larger changes in the codebase, so the property is still failing; in the meantime, Curvance made changes so that the `combineAllLocks` function is called only if epochs are available for delivery.

9. Negative prices from OracleRouter cause underflow and panic

Severity: **Undetermined**

Tool: Medusa

Invariant ID: MARKET-7	Finding ID: TOB-CURV-9

Target: `contracts/market/MarketManager.sol`

Description

The `MarketManager` contract uses the `OracleRouter` contract (previously called `PriceRouter`) to consume oracle prices, evaluate assets, and set system variables, including variables for collateral tokens.

Based on testing with Medusa, we found that the `MarketManager` contract does not check that input from `OracleRouter` is within safe bounds. The Medusa test sets the oracle price to `-40366860537506782850827976585213507643726085575883236363156016016024833339816`. This value causes an underflow in `MarketManager`.

```
↛ [FAILED] Assertion Test:
FuzzingSuite.updateCollateralToken_should_succeed(address,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256)
Test for method
"FuzzingSuite.updateCollateralToken_should_succeed(address,uint256,uint256,uint256,uint256,uint256,uint256,uint256,uint256)" resulted in an assertion failure after the following call
sequence:[...]Lendtroller.updateCollateralToken(0x15af364944c1d234f9531be3ad42e33fd0a5c5ed, 1823, 3167, 2974, 1661, 2965, 135, 7561)
(addr=0x6D26dc80248ac0Aaf692147Ca1F8098028a8cbc4, value=0,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
              => [call]
CentralRegistry.hasElevatedPermissions(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0x587be02D13c624E65b3D98C33fdf3Eea13aAAf97, value=<nil>,
sender=0x6D26dc80248ac0Aaf692147Ca1F8098028a8cbc4)
                     => [return (true)]
              => [call] MockCToken.isCToken()
(addr=0x15AF364944c1d234f9531Be3Ad42E33fD0A5c5ED, value=<nil>,
sender=0x6D26dc80248ac0Aaf692147Ca1F8098028a8cbc4)
                     => [return (true)]
              => [call] CentralRegistry.priceRouter()
                           => [call] MockV3Aggregator.latestRoundData()
(addr=0x48E4F3f3daE11341ff2eDF60Af6857Ae08C871C5, value=<nil>,
sender=0xE31B891AbFc185fCb3A61bb1AaD756D3eEDAF19E)
```

```
                                    => [return (1,
-40366860537506782850827976585213507643726085575883236363156016016024833339816,
1256497, 1256497, 1)]
                               => [panic: arithmetic underflow]
                          => [panic: arithmetic underflow]
                     => [panic: arithmetic underflow]
       => [event] AssertFail("MARKET MANAGER - updateCollateralToken should
succeed")
       => [panic: assertion failed]
```

Figure 9.1: The assertion failure from Medusa when a negative oracle price is passed

Property Status: Passing

After changes made in commit `8c9a2dd`, the property is now passing after 96 hours with Medusa and 10,000,000 runs with Echidna. The contract now reverts with the `PriceError` error on such input.

10. Division-by-zero error in _canLiquidate results in a panic

Severity: **Undetermined**

Tool: Echidna

Invariant ID: DTOK-19	Finding ID: TOB-CURV-10

Target: `contracts/market/collateral/DToken.sol`

Description

When a user attempts to soft liquidate an account with zero tokens, the code is intended to error out with the `InvalidAmount` or `InvalidParameter` error. Instead, the code currently panics on the `_canLiquidate` function when attempting to calculate the debt-to-collateral ratio.

```
function _canLiquidate(
    address debtToken,
    address collateralToken,
    address account,
    uint256 debtAmount,
    bool liquidateExact
) internal view returns (uint256, uint256, uint256) {
    [...]
    {
        uint256 cFactor = cToken.baseCFactor +
            ((cToken.cFactorCurve * data.lFactor) / WAD);
        uint256 incentive = cToken.liqBaseIncentive +
            ((cToken.liqCurve * data.lFactor) / WAD);
        maxAmount =
            (cFactor * IMToken(debtToken).debtBalanceCached(account)) /
            WAD;

        // Get the exchange rate, and calculate the number of
        // collateral tokens to seize.
        debtToCollateralRatio =
            (incentive * data.debtTokenPrice * WAD) /
            (data.collateralTokenPrice *
                IMToken(collateralToken).exchangeRateCached());
    }
```

Figure 10.1: The `_canLiquidate` function's calculation of `debtToCollateralRatio`

This was discovered using the following targeted fuzz test, which attempts to liquidate exactly `0` tokens and expects an `InvalidAmount` or `InvalidParameter` error.

```
function liquidate_should_fail_with_exact_with_zero(
    address dtoken,
    address collateralToken
) public {
    require(marketManager.seizePaused() != 2);

    address account = address(this);
    _isSupportedCToken(collateralToken);
    _isSupportedDToken(dtoken);
    uint256 amount = 0;
    // Structured for non exact liquidations, debt amount to liquidate = max
    uint256 collateralPostedFor = _collateralPostedFor(
        address(collateralToken)
    );

    _preLiquidate(amount, DAI_PRICE, USDC_PRICE);

    // expect the above to fail
    hevm.prank(msg.sender);
    try
        DToken(dtoken).liquidateExact(
            account,
            amount,
            IMToken(collateralToken)
        )
    {} catch (bytes memory revertData) {
        uint256 errorSelector = extractErrorSelector(revertData);

        assertWithMsg(
            errorSelector == invalid_amount ||
                errorSelector ==
                marketManager_invalidParameterSelectorHash,
            "DTOK-X liquidateExact should fail with amount 0"
        );
    }
}
```

Figure 10.2: The fuzz test for attempting to liquidate exactly `0` tokens

Instead, the test errors out with `Panic(18)`, as shown in figure 10.3:

```
|  ├─call
0x21e2f36B03B601D016282916e94477388C5698e2::debtBalanceCached(@0x00a329c0648769A73af
Ac7F9381E08FB43dBEA72) <no source map>
|  |  └─← (3576935868421631400)
|  ├─call 0x1FD8397e8108ada12eC07976D92F773364ba46e7::exchangeRateCached() <no
source map>
|  |  └─← (1000000000000000000)
|  └─error Revert Panic(18) <no source map>
```

```
  └ error Revert Panic(18) <no source map>
AssertFail(«DTOK-X liquidateExact should fail with amount 0»)
(curvance-contracts/tests/fuzzing/helpers/PropertiesHelper.sol:20)
```

Figure 10.3: The fuzz test results in `Panic(18)`

Property Status: Failing

Because this issue was found toward the end of the invariant development and testing engagement, we did not rerun this property against adjusted or changed code.

11. Missing validation allows the DAO address to be liquidated

Severity: **Undetermined**

Tool: Echidna

Invariant ID: DTOK-23	Finding ID: TOB-CURV-11

Target: `contracts/market/collateral/DToken.sol`

Description

When executing a soft liquidation of an account, the code does not check that the DAO address cannot create positions or have its positions liquidated.

Liquidation of an account owned by the DAO breaks the invariant that the user's collateral balance before and after the liquidation are equal to the total number of assets that are seized for liquidation (figure 11.1). The invariant fails in this case because a portion of the seized assets is sent to the DAO address to pay the liquidation protocol fee, so the DAO's collateral balance would not decrease by exactly the amount seized for liquidation.

```
assertEq(
    collateralBalanceBefore -
        IMToken(collateralToken).balanceOf(address(this)),
    seizedForLiquidation,
    "DTOK-23 soft liquidate should reduce collateral balance for account"
);
```

Figure 11.1: A snippet of the failing invariant in the `MarketManager` contract

Figure 11.2 shows the assertion failure from the fuzz test:

```
liquidate_should_succeed_with_non_exact(uint256): failed!💥
  Call sequence:
    *wait* Time delay: 524985 seconds Block delay: 48244
Traces:
    FuzzingSuite.setUpFeeds()

FuzzingSuite.list_token_should_succeed(0x21e2f36b03b601d016282916e94477388c5698e2)
Gas: 12500000

FuzzingSuite.updateCollateralToken_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1,887192370226704489960101617487114324310624488298680364034339031837070479310
7,614621778738386235624065957385891686258890098390955543076192334483662471046 91,3214
452572,143255024352237645468438987921911973439935563055607973253249038155949996978 23
,115792089237316195423570985008687907853269984665640564039457584007913043643789,1157
```

```
92089237316195423570985008687907853269984665640564039457584007912869166243,3601)
Gas: 12500000

FuzzingSuite.setCToken_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1,658
603257911427518958798575924450352984054259822384216228430634643481445828 95) Gas:
12500000

FuzzingSuite.mint_should_actually_succeed(0x21e2f36b03b601d016282916e94477388c5698e2
,115792089237316195423570985008687907853269984665640564039457584007913129629737)
Gas: 12500000
    FuzzingSuite.liquidate_should_succeed_with_non_exact(77066) Gas: 12500000
    AssertEqFail(«Invalid: 999196643756136751!=1000000000000000001, reason: DTOK-23
soft liquidate should zero out collateral balance for account»)
```

Figure 11.2: The fuzz test's assertion check on the change in collateral balance

Figure 11.3 shows the values involved in this failure and both the expected and actual assertions.

```
collateralBalanceBefore - collateralBalanceAfter = 999196643756136751
seizedForLiquidation = 1000000000000000001
seizedForProtocol = 803356243863250

expected: collateralBalanceBefore - collateralBalanceAfter == seizedForLiquidation
actual: collateralBalanceBefore - collateralBalanceAfter + seizedForProtocol =
seizedForLiquidation
```

Figure 11.3: The expected assertion and the actual assertion

Property Status: Failing

Because this issue was found toward the end of the invariant development and testing engagement, we did not rerun this property against adjusted or changed code.

12. Missing validation allows the DAO address to be the liquidator

Severity: **Undetermined**

Tool: Echidna

Invariant ID: DTOK-25	Finding ID: TOB-CURV-12

Target: `contracts/market/collateral/DToken.sol`

Description

When executing a soft liquidation of an account, the code does not check that the DAO address cannot liquidate positions.

Liquidation of an account by the DAO breaks the invariant that the number of collateral tokens a liquidator has is equal to the liquidated account's prior collateral token balance plus the collateral tokens meant for the liquidator. The DAO address receives the liquidation fee for the protocol, so the liquidator's collateral tokens do not increase by only the collateral tokens meant for the liquidator (it also increases by the liquidation fee).

```
assertEq(
    IERC20(collateralToken).balanceOf(msg.sender),
    preSenderCollateral + collateralTokensForLiquidator,
    "DTOK-25 soft liquidate: collateral token balance of sender must increase by
(amount seized by liquidation - amount seized for protocol)"
);
```

Figure 12.1: The fuzz test's assertion failure when the liquidator is the DAO address

Figure 12.2 shows the assertion failure from the fuzz test:

```
liquidate_should_succeed_with_non_exact(uint256): failed!💥
  Call sequence:
    *wait* Time delay: 524985 seconds Block delay: 48244
Traces:
    FuzzingSuite.setUpFeeds()

FuzzingSuite.list_token_should_succeed(0x21e2f36b03b601d016282916e94477388c5698e2)
Gas: 12500000

FuzzingSuite.updateCollateralToken_should_succeed(0x600515dfe465f600f0c9793fa27cd279
4f3ec0e1,887192370226704489960101617487114324310624488298680364034339031837070479310
7,614621778738386235624065957385891686258890098390955543076192334483662471046 91,3214
452572,143255024352237645468438987921911973439935563055607973253249038155949996978 23
,115792089237316195423570985008687907853269984665640564039457584007913043643789,1157
```

```
92089237316195423570985008687907853269984665640564039457584007912869166243,3601)
Gas: 12500000

FuzzingSuite.setCToken_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1,658
603257911427518958798575924450352984054259822384216228430634643481445828 95) Gas:
12500000

FuzzingSuite.mint_should_actually_succeed(0x21e2f36b03b601d016282916e94477388c5698e2
,115792089237316195423570985008687907853269984665640564039457584007913129629737)
Gas: 12500000

FuzzingSuite.liquidateAccount_should_fail_if_self_account(789080699425037388652328886
2861140710707897408958482457526080960647 64) Gas: 12500000
    FuzzingSuite.liquidate_should_succeed_with_non_exact(77066) Gas: 12500000
    AssertEqFail(«Invalid: 999196643756136751!=1000000000000000001, reason: DTOK-23
soft liquidate should reduce out collateral balance for account»)
```

Figure 12.2: The fuzz test's assertion check on the change in collateral balance

Figure 12.3 shows the values involved in this failure and both the expected and actual assertions. When the liquidator is the DAO address, the liquidator's balance also increases by the value of `seizedForProtocol`:

```
collateralBalanceAfter = 999196643756136751
seizedForLiquidation = 1000000000000000001
seizedForProtocol = 803356243863250

expected:
   collateralBalanceAfter = collateralBalanceBefore + seizedForLiquidation
actual:
   collateralBalanceAfter + seizedForProtocol = collateralBalanceBefore +
seizedForLiquidation
```

Figure 12.3: The expected assertion and the actual assertion

Property Status: Failing
Because this issue was found toward the end of the invariant development and testing engagement, we did not rerun this property against adjusted or changed code.

13. The repay function will panic if a user's total borrows and debt balance are 1 wei off

Severity: **Undetermined**	
Tool: Echidna	
Invariant ID: DTOK-12	Finding ID: TOB-CURV-13
Target: `DToken.sol`	

Description

As an effect of rounding in the protocol, the `repay` function can be off by 1 wei and cause a panic.

When users repay debt back to the protocol, they specify the amount that they would like to repay.

```
/// @notice Caller repays their own debt
/// @dev    Updates interest before executing the repayment
/// @param amount The amount to repay, or 0 for the full outstanding amount
function repay(uint256 amount) external nonReentrant {
    accrueInterest();

    _repay(msg.sender, msg.sender, amount);
}
```

Figure 13.1: The `repay` function

The `canRepay` function (called by the `_repay` helper function) checks only that the token is properly listed in the system and that the `MINIMUM_HOLD_PERIOD` has passed before the repay attempt. There is no validation to ensure that the value of `amount` does not exceed the value of `accountDebt`; if it does, the calculation of the principal amount (highlighted in yellow in figure 13.1) will underflow and cause a panic.

```
/// @dev First validates that the payer is allowed to repay the loan, then repays
///      the loan by transferring in the repay amount. Emits a repay event on
///      successful repayment.
/// @param payer The address paying off the borrow
/// @param account The account with the debt being paid off
/// @param amount The amount the payer wishes to repay, or 0 for the full
outstanding amount
/// @return The actual amount repaid
function _repay(
```

```
    address payer,
    address account,
    uint256 amount
) internal returns (uint256) {
    // Validate that the payer is allowed to repay the loan
    lendtroller.canRepay(address(this), account);

    // Cache how much the account has to save gas
    uint256 accountDebt = debtBalanceCached(account);

    // If amount == 0, amount = accountDebt
    amount = amount == 0 ? accountDebt : amount;

    SafeTransferLib.safeTransferFrom(
        underlying,
        payer,
        address(this),
        amount
    );

    // We calculate the new account and total borrow balances,
    // failing on underflow:
    _debtOf[account].principal = accountDebt - amount;
    _debtOf[account].accountExchangeRate = marketData.exchangeRate;
    totalBorrows -= amount;

    emit Repay(payer, account, amount);
    return amount;
}
```

Figure 13.2: The _repay function

Echidna broke this invariant by finding inputs that would cause the principal amount calculation to underflow and the function to panic.

```
repay_within_account_debt_should_succeed(address,uint256): failed!💥
  Call sequence:
    *wait* Time delay: 524985 seconds Block delay: 48244
Traces:
    FuzzingSuite.setUpFeeds()

FuzzingSuite.list_token_should_succeed(0x21e2f36b03b601d016282916e94477388c5698e2)

FuzzingSuite.updateCollateralToken_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1,887192370226704489960101617487114324310624488298680364034339031837070479310
7,614621778738386235624065957385891686258890098390955543076192334483662471046 91,3214
452572,143255024352237645468438987921911973439935563055607973253249038155949996978 23
,115792089237316195423570985008687907853269984665640564039457584007913043643789,1157
920892373161954235709850086879078532699846656405640394575840079128691662 43,3601)

FuzzingSuite.setCToken_should_succeed(0x600515dfe465f600f0c9793fa27cd2794f3ec0e1,658
603257911427518958798575924450352984054259822384216228430634643481445828 95)
```

```
FuzzingSuite.mint_should_actually_succeed(0x21e2f36b03b601d016282916e94477388c5698e2
,1157920892373161954235709850086879078532699846656405640394575840079131296297 37)
    FuzzingSuite.liquidateAccount_should_fail_if_self_account(10032978595816)

FuzzingSuite.mint_should_actually_succeed(0x21e2f36b03b601d016282916e94477388c5698e2
,1157920892373161954235709850086879078532699846656405640394575840079131296297 37)

FuzzingSuite.repay_within_account_debt_should_succeed(0x21e2f36b03b601d016282916e944
77388c5698e2,0)
      ├─Log(«total borrows», 10032978595877)
      ├─Log(«amount», 10032978595878)
      └─[error[0m Revert Panic(17)
      AssertFail(«DTOK-12 repay should succeed with correct preconditions»)
```

Figure 13.3: The shortened Echidna trace for the repay

Property Status: Further Investigation Required

Because this issue was found toward the end of the invariant development and testing engagement, we did not rerun this property against adjusted or changed code.

A. Finding Severity Levels

The following table describes the severity levels used in this document.

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.

B. Code Maturity Categories

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

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
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
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Fuzz Testing Suite Expansion Recommendations

This appendix provides recommendations for adding enhancements and features to the fuzz testing suite to increase its coverage of the VeCVE and market manager components' system states.

System-Wide Functionality

- **Add tests for the `rescueToken` functions of each component.** The testing suite does not contain entry points for these functions.
- **Add support for multiple callers.** Other than for the liquidation-related functions, the testing suite mostly tests functionalities with `address(this)` (the fuzzing contract) as the caller. The testing suite can be expanded to support multiple users.

VeCVE Component

- **Add tests for creating a new noncontinuous lock after processing/increasing the new lock's amount.** The current testing suite tests only the path where the new lock is continuous.
- **Add tests for delegation functions.** The `createLockFor` and `increaseAmountAndExtendLockFor` functions are missing coverage in the testing suite.
- **Test the full range of input values for creating locks.** The testing suite tests only the bounds between `WAD` and `uint64`.
- **Add tests for postconditions of the `processExpiredLock` function for when the unlock time is expired, the system is not shut down, and the user intends to relock.** The current assertion checks only that the number of locks stays the same, thus missing coverage of the rest of the function.
- **Add a property to check that attempts to disable a continuous lock by passing a noncontinuous lock fail with the `LockTypeMismatch` error.** The testing suite does not check this corner case.
- **Add a property to check that if the value of `veBalanceOf` does not equal the value of `lockedAmount`, the function reverts with the `InvariantError` error.** This invariant was identified after the VeCVE component was tested; therefore, it is missing coverage of this area.
- **Add a property to call `combineAllLocks` in the case where the value of `veBalance` is greater than the value of `currentPoints` to check that the value of user points increments correctly.** These branches were introduced after a refactor of `combineAllLocks` and are missing coverage of this corner case.

- **Add a property to test the bridging of veCVE locks.** This functionality was added after creating and running the testing suite, so it is missing coverage.
- **Test the correctness of getter functions such as `getUnlockPenalty` and `getVotesForSingleLockForTime`.** These functions are missing coverage in the testing suite.

Market Manager Components

- **Add a test to call `_checkAuthorizedPermissions` with an unauthorized user to ensure that it errors out with the `UNAUTHORIZED_SELECTOR` error.** This error is not tested in the testing suite.
- **Add a test that calls `postCollateral` with collateral values that exceed the caps to ensure it errors out with the `CollateralCapReached` error.** This path is not covered in the testing suite.
- **Fine-tune the preconditions for `removeCollateral` to test the path involving closing a position.** This path is not covered in the testing suite.
- **Add a test to ensure that if `lFactor` is `0`, `_canLiquidate` errors out with the `NoLiquidationAvailable` error.** This check is missing from the testing suite.
- **Add a test to check that `_canLiquidate` errors out with the `INVALID_PARAMETER_SELECTOR` error if `debtAmount` is greater than `maxAmount` or `liquidatedTokens` is greater than `collateralAvailable` during non-exact liquidations.** This coverage is missing in the testing suite.
- **Add a test to ensure that the correct error message is returned when a user attempts to post an amount of collateral that exceeds their cToken balance.** This error message check is missing from the testing suite.
- **Add tests to check the postconditions of the collateral and account exchange rates after the borrow functions are called.** The testing suite does not check postconditions after increases in collateral.
- **Add state checks for `canLiquidate` and `canLiquidateWithExecution` to make sure these functions execute successfully when they are expected to.** The testing suite is missing coverage of these functions.
- **Add invariant helper functions to test the `reduceCollateralIfNecessary` function.** This function is not covered by the testing suite.
- **Add a test to check the preconditions and postconditions of the `canRedeemWithCollateralRemoval` function.** This function is not covered by the testing suite.

- **Add a test to check the preconditions and postconditions of the `notifyBorrow` function.** This function is not covered by the testing suite.
- **Add a test to check that the correct error message is returned by `canRepay` if the minimum hold period has not passed.** This error is not covered by the testing suite.
- **Add a test to check that the correct error message is returned by `canSeize` if the given `cToken` and `dToken` have a different `MarketManager` contract.** This error is not covered by the testing suite.
- **Add a test to check that the correct error message is returned by `listToken` if it is passed a token that has already been listed in the system.** This error is thrown twice in the `listToken` function, but the testing suite checks only the first instance.
- **Add a test to call `setCTokenCollateralCaps` with no tokens listed to ensure it errors out with the `INVALID_PARAMETER_SELECTION` error.** This error is not covered by the testing suite.
- **Add a test to call `setPositionFolding`.** This function is not covered by the testing suite.
- **Add coverage of interest calculations in <u>all</u> functions tested by the testing suite.** The testing suite has some separate tests for cases in which interest is accrued and not accrued; however, some areas of the codebase are missing coverage of these cases, especially in the case of liquidations. In order to calculate the most accurate number, a helper function should be added to calculate hypothetical interest accrual and to check the results against a static value to ensure exchange rates and values increase accordingly.

Liquidation-Specific Recommendations

- **Add sufficient checks for the 1–2 wei difference between `totalBorrows` and debt balance.** The current codebase checks for this 1–2 wei difference; however, it should be tested more clearly to ensure that the difference cannot exceed these amounts.
- **Add randomness in asset pricing.** The `liquidateAccount` function in the `FuzzMarketManager` contract serves as an example of a dynamic USDC and DAI price generation; however, all other liquidation functions use the default DAI/USDC price constants. These functions should all use dynamic price values to allow the fuzzers to determine what values to use.
- **Add collateral and debt token balance checks for `liquidateAccount`.** The testing suite checks only that the user no longer has collateral posted in their

collateral tokens and has a debt balance of zero; it does not check that the liquidator or the protocol received the correct number of tokens.

- **Add additional assertions in the soft liquidation test (in FuzzDToken) to ensure that the `InvalidAmount` error is thrown on the correct conditions.** Currently, the contract catches this error silently, as additional assertion statements are needed to check whether the call should have reverted. This error can arise when the gauge pool deposit rounds down to zero or the amount to be liquidated is zero.
- **Add tests of interest accrual calculations in the liquidation functions.** Many of the liquidation functions will calculate accrued interest on the debt token prior to calculating liquidation values to minimize the rate movement. This is an area of the codebase that is missing coverage in the testing suite.
- **Add dynamic token integrations, especially with multiple decimals and skewing price feeds.** As mentioned in the market manager components recommendations, the tests of the functionality for posting, borrowing, and repaying of tokens are currently limited to DAI and USDC. Once the rest of the market manager testing suite has been extended to support multiple tokens, we recommend extending it further to test liquidations involving other tokens to ensure that obscure decimals or bounds do not affect existing invariants.

D. Failures for VECVE-4 and VECVE-10

This appendix provides the full callstacks and failures related to invariants VECVE-4 and VECVE-10.

Figures D.1 through D.3 show the callstacks for `combineAllLocks` that led to failures.

```
1)
FuzzingSuite.processExpiredLock_should_succeed(4496454416213147509794643738817082564
6359824528434646094912480309692987885241) (block=33720, time=519386, gas=12500000,
gasprice=1, value=0, sender=0x0000000000000000000000000000000000020000)
2)
FuzzingSuite.create_lock_when_not_shutdown(1336786942781002144905519891768561620156 0
69754264890133884262130056333112787 3, true) (block=64156, time=761575, gas=12500000,
gasprice=1, value=0, sender=0x0000000000000000000000000000000000030000)
3) FuzzingSuite.combineAllLocks_should_succeed_to_continuous_terminal()
(block=71513, time=1359871, gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000020000)
[Execution Trace]
 => [call] FuzzingSuite.combineAllLocks_should_succeed_to_continuous_terminal()
(addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0,
sender=0x0000000000000000000000000000000000020000)
         => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (6447388898)]
         => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (1595215587, 1099511627775)]
         => [call] VeCVE.CONTINUOUS_LOCK_VALUE()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (1099511627775)]
         => [call] VeCVE.clPointMultiplier()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (11000)]
         => [event] LogUint256("adjustment amount:", 159521558)
         => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 1)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (4266047049, 1099511627775)]
         => [call] VeCVE.CONTINUOUS_LOCK_VALUE()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (1099511627775)]
         => [call] VeCVE.clPointMultiplier()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                 => [return (11000)]
```

```
        => [event] LogUint256("adjustment amount:", 586126262)
        => [call] VeCVE.combineAllLocks(true, {desiredRewardToken:
0x01375317aa980daabf22f990a378eccad9b40dc0, shouldLock: false, isFreshLock: false,
isFreshLockContinuous: false}, "", 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=0,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [call]
CVELocker.epochsToClaim(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0x9c6a6B0ec78aA6e4b9bebD9DcEE3F2b071377d07, value=<nil>,
sender=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2)
                        => [return (0)]
                => [return ()]
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (5861262636, 1099511627775)]
        => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (6447388899)]
        => [call] VeCVE.clPointMultiplier()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (11000)]
        => [event] LogUint256("currentadjustment for cl", 586126263)
        => [event] LogUint256("userPointsAdjustmentForContinuous", 586126262)
        => [event] AssertEqFail("Invalid: 6447388898!=6447388899, reason: VE_CVE -
combineAllLocks() - all prior continuous => continuous failed")
        => [panic: assertion failed]
```

Figure D.1: The callstack for combining all prior continuous locks into a continuous lock terminal

```
[Call Sequence]
1) FuzzingSuite.getVotesForSingleLockForTime_correct_calculation(0,
115792089237316195423570985008687907853269958216452065683869244073109405663913,
351186346335046887244657844317416821861065794201033579603167244228093737020850)
(block=29945, time=546461, gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000030000)
2) FuzzingSuite.create_lock_when_not_shutdown(4294967298, false) (block=54248,
time=1063080, gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000010000)
3)
FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed()
(block=79096, time=1428346, gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000030000)
[Execution Trace]
 => [call]
FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed()
(addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0,
sender=0x0000000000000000000000000000000000030000)
        => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
```

```
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (5)]
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (1, 1099511627775)]
        => [call] VeCVE.CONTINUOUS_LOCK_VALUE()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (1099511627775)]
        => [call] VeCVE.clPointMultiplier()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (11000)]
        => [event] LogUint256("adjustment amount:", 0)
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 1)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (4, 31449600)]
        => [call] VeCVE.CONTINUOUS_LOCK_VALUE()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (1099511627775)]
        => [event] LogUint256("adjustment amount:", 0)
        => [call] VeCVE.combineAllLocks(true, {desiredRewardToken:
0x01375317aa980daabf22f990a378eccad9b40dc0, shouldLock: false, isFreshLock: false,
isFreshLockContinuous: false}, "", 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=0,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [call]
CVELocker.epochsToClaim(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0x9c6a6B0ec78aA6e4b9bebD9DcEE3F2b071377d07, value=<nil>,
sender=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2)
                        => [return (0)]
                => [return ()]
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (5, 1099511627775)]
        => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (5)]
        => [event] AssertLtFail("Invalid: 5>=5 failed, reason: VE_CVE -
combineAllLocks() - some prior continuous => continuous failed")
        => [panic: assertion failed]
```

Figure D.2: The callstack for combining some prior continuous locks into a continuous lock terminal

```
→ [FAILED] Assertion Test:
```

```
FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed()
Test for method
"FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed(
)" resulted in an assertion failure after the following call sequence:
[Call Sequence]
1)
FuzzingSuite.processExpiredLock_should_succeed(1157920892373161954235709850086879078
532699846656405640394575840079131296363 36) (block=1417, time=117330, gas=12500000,
gasprice=1, value=0, sender=0x0000000000000000000000000000000000030000)
2) FuzzingSuite.create_lock_when_not_shutdown(0, false) (block=33733, time=421859,
gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000010000)
3)
FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed()
(block=83842, time=958953, gas=12500000, gasprice=1, value=0,
sender=0x0000000000000000000000000000000000020000)
[Execution Trace]
 => [call]
FuzzingSuite.combineAllLocks_non_continuous_to_continuous_terminals_should_succeed()
(addr=0xA647ff3c36cFab592509E13860ab8c4F28781a66, value=0,
sender=0x0000000000000000000000000000000000020000)
        => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (4724460067)]
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (4294963697, 1099511627775)]
        => [call] VeCVE.CONTINUOUS_LOCK_VALUE()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (1099511627775)]
        => [call] VeCVE.clPointMultiplier()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (11000)]
        => [event] LogUint256("adjustment amount:", 429496369)
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 1)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (1, 31449600)]
        => [call] VeCVE.CONTINUOUS_LOCK_VALUE()
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (1099511627775)]
        => [event] LogUint256("adjustment amount:", 429496369)
        => [call] VeCVE.combineAllLocks(true, {desiredRewardToken:
0x01375317aa980daabf22f990a378eccad9b40dc0, shouldLock: false, isFreshLock: false,
isFreshLockContinuous: false}, "", 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=0,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [call]
```

```
CVELocker.epochsToClaim(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0x9c6a6B0ec78aA6e4b9bebD9DcEE3F2b071377d07, value=<nil>,
sender=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2)
                        => [return (0)]
                => [return ()]
        => [call] VeCVE.userLocks(0xa647ff3c36cfab592509e13860ab8c4f28781a66, 0)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (4294963698, 1099511627775)]
        => [call] VeCVE.userPoints(0xa647ff3c36cfab592509e13860ab8c4f28781a66)
(addr=0xaa80b404C0c9c17a62129b00DA58e257Db87E1B2, value=<nil>,
sender=0xA647ff3c36cFab592509E13860ab8c4F28781a66)
                => [return (4724460067)]
        => [event] AssertLtFail("Invalid: 4724460067>=4724460067 failed, reason:
VE_CVE - combineAllLocks() - non prior continuous => continuous failed")
        => [panic: assertion failed]
```

Figure D.3: The callstack for combining previously created locks, none of which are continuous, into a continuous lock terminal

E. Downrunning Implications of VECVE-55

The `processExpiredLock` failure of invariant VECVE-55 causes many other functions to fail with assertion errors. Figures E.1 and E.2 show two of those failures.

```
processExpiredLock_should_succeed_if_unlocktime_expired(bool): failed!💥
  Call sequence:
    extractErrorSelector("\NUL") Gas: 1250000000 Time delay: 17850676 seconds Block
delay: 1535
    processExpiredLock_should_succeed_if_unlocktime_expired(true) Gas: 1250000000
Time delay: 12939732 seconds Block delay: 1
    shutdown_success_if_elevated_permission() Gas: 1250000000
    disableContinuousLock_should_succeed_if_lock_exists()

processExpiredLock_should_succeed_if_shutdown(2739256766125766039833248537485672048
3289503369)
    processExpiredLock_should_succeed_if_unlocktime_expired(false)

Traces:
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:747)
 └─← (1)
LogUint256(«processExpiredLock_should_succeed_if_unlocktime_expired - vecve», 1)
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:745)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1219)
 └─← (1)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userLocks(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72, 0) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1210)
 └─← (1000000000000000000, 1555545600)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userLocks(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72, 0) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1179)
 └─← (1000000000000000000, 1555545600)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userPoints(@0x00a329c0648769A73afAc7F938
1E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:755)
 └─← (1000000000000000000)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::chainPoints()
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:756)
 └─← (1000000000000000000)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::currentEpoch(1555545600)
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:757)
 └─← (1286)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::chainUnlocksByEpoch(1286)
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:758)
 └─← (1000000000000000000)
call
```

```
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userUnlocksByEpoch(@0x00a329c0648769A73a
fAc7F9381E08FB43dBEA72, 1286) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:761)
 └← (100000000000000000)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::balanceOf(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:765)
 └← (100000000000000000)
call
0x492934308E98b590A626666B703A6dDf2120e85e::balanceOf(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:766)
 └← (5040000828500000000000000)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::0x9521e5bb0000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000001000000
0000000000000000062d69f6867a0a084c6d313943dc22023bc263691000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000 (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:769)
 ├ call
0x0A64DF94bc0E039474DB42bb52FEca0c1d540402::epochsToClaim(@0x00a329c0648769A73afAc7F
9381E08FB43dBEA72) <no source map>
 |  └← (0)
 ├ Transfer(100000000000000000) <no source map>
 ├ call
0x492934308E98b590A626666B703A6dDf2120e85e::transfer(@0x00a329c0648769A73afAc7F9381E
08FB43dBEA72, 100000000000000000) <no source map>
 |  └ error Revert InsufficientBalance () <no source map>
 └ error Revert TransferFailed () <no source map>
AssertFail(«VE_CVE - processExpiredLock() failed unexpected if unlock time expired»)
(curvance-contracts/tests/fuzzing/PropertiesHelper.sol:20)
```

Figure E.1: `processExpiredLock` fails if the unlock time expires due to insufficient balance.

```
processExpiredLock_should_succeed_if_shutdown(uint256): failed!💥
  Call sequence:
    extractErrorSelector("\NUL") Gas: 1250000000 Time delay: 19511168 seconds Block
delay: 23007
    processExpiredLock_should_succeed_if_unlocktime_expired(true) Gas: 1250000000
Time delay: 11430890 seconds Block delay: 1
    shutdown_success_if_elevated_permission() Gas: 1250000000
    combineAllLocks_should_succeed_to_non_continuous_terminal() Gas: 1250000000
    processExpiredLock_should_succeed_if_shutdown(0) Gas: 1250000000

Traces:
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:630)
 └← (1)
LogUint256(«processExpiredLock_should_succeed_if_shutdown - vecve», 1)
```

```
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:628)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::isShutdown()
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:632)
 └─← (2)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1219)
 └─← (1)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::queryUserLocksLength(@0x00a329c0648769A7
3afAc7F9381E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1219)
 └─← (1)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userLocks(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72, 0) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:1179)
 └─← (2000000000000000000, 1586995200)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userPoints(@0x00a329c0648769A73afAc7F938
1E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:639)
 └─← (2000000000000000000)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::chainPoints()
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:640)
 └─← (2000000000000000000)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::currentEpoch(1586995200)
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:641)
 └─← (1312)
call 0x46662E22D131Ea49249E0920C286E1484FEEf76E::chainUnlocksByEpoch(1312)
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:642)
 └─← (2000000000000000000)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::userUnlocksByEpoch(@0x00a329c0648769A73a
fAc7F9381E08FB43dBEA72, 1312) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:645)
 └─← (2000000000000000000)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::balanceOf(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:650)
 └─← (2000000000000000000)
call
0x492934308E98b590A626666B703A6dDf2120e85e::balanceOf(@0x00a329c0648769A73afAc7F9381
E08FB43dBEA72) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:652)
 └─← (5040000728500000000000000)
call
0x492934308E98b590A626666B703A6dDf2120e85e::balanceOf(@0x46662E22D131Ea49249E0920C28
6E1484FEEf76E) (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:656)
 └─← (1000000000000000000)
LogUint256(«pre lock vecve balance of vecve:», 1000000000000000000)
(curvance-contracts/tests/fuzzing/FuzzVECVE.sol:654)
call
0x46662E22D131Ea49249E0920C286E1484FEEf76E::0x9521e5bb0000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000001000000
0000000000000000062d69f6867a0a084c6d313943dc22023bc263691000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000120
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000 (curvance-contracts/tests/fuzzing/FuzzVECVE.sol:660)
 ├─ call
0x0A64DF94bc0E039474DB42bb52FEca0c1d540402::epochsToClaim(@0x00a329c0648769A73afAc7F
9381E08FB43dBEA72) <no source map>
 │  └─ ← (0)
 ├─ Transfer(2000000000000000000) <no source map>
 ├─ call
0x492934308E98b590A626666B703A6dDf2120e85e::transfer(@0x00a329c0648769A73afAc7F9381E
08FB43dBEA72, 2000000000000000000) <no source map>
 │  └─ error Revert InsufficientBalance () <no source map>
 └─ error Revert TransferFailed () <no source map>
AssertFail(«VE_CVE - processExpiredLock() failed unexpected»)
(curvance-contracts/tests/fuzzing/PropertiesHelper.sol:20)
```

Figure E.2: `processExpiredLock` fails if the unlock time expires due to insufficient balance.

F. Mathematical Analysis on Bounds of Values during Liquidations

cFactor Calculation

$$cFactor = baseCFactor + \frac{cFactorCurve * lFactor}{WAD}$$

The following are true:

- $cFactorCurve = (WAD - baseCFactor)$
- $lFactor$ is bounded between $[1, WAD]$.

These assumptions give us the following calculation:

- $cFactor = baseCFactor + \frac{(WAD - baseCFactor) * [0, WAD]}{WAD}$

The following is true with a lower bound of $[0, WAD]$:

- $cFactor = baseCFactor$

The following is true with a higher bound of $[0, WAD]$:

- $cFactor = baseCFactor + WAD - baseCFactor$
- $cFactor = WAD$

$baseCFactor$ is bounded between $[0.1 * WAD, 0.5 * WAD]$.

As a result, $CFactor$ is bounded between $[baseCFactor, WAD]$.

Incentive Calculation

$$incentive = liqBaseIncentive + \frac{liqCurve * lfactor}{WAD}$$

Input Bounds

- $liqBaseIncentive$ is bounded between $[.01e18, .3e18 - 1]$.
- $lFactor$ is bounded between $[0, WAD]$.
    - If $lFactor$ is 0, $liqCurve$ is not used.
- $liqCurve$ is equal to the incentive for a hard liquidation minus the incentive for a soft liquidation.

Output Bounds

- $incentive$ is bounded between $[.01e18, .3e18]$.

Maximum Amount Calculation

$$maxAmount = \frac{cFactor * debtBalanceCached}{WAD}$$

Input Bounds

- $cFactor$ is bounded between $[0, WAD]$.
- $debtBalanceCached$ is equal to the amount of debt the user has.

Output Bounds

- If $cFactor$ is 0, then $maxAmount$ is 0.
- If $cFactor$ is WAD, then $maxAmount$ is $debtBalanceCached$.
- The final returned value of $maxAmount$ must always be bounded between $[0, debtBalanceCached]$.

Debt to Collateral Ratio Calculation

$$debtToCollateralRatio = \frac{incentive * debtTokenPrice * WAD}{collateralTokenPrice * exchangeRateCached}$$

Input Bounds

- The bounds of $incentive$ depend on the incentive calculation. (See that calculation earlier in this section.)
- $debtTokenPrice$ and $collateralTokenPrice$ are theoretically unbounded.
- $exchangeRateCached$ is theoretically unbounded.

Amount Adjusted Calculation

$$amountAdjusted = \frac{debtAmount * 10^{collateralTokenDecimals}}{10^{debtTokenDecimals}}$$

Input Bounds

- There are no checks of decimals when tokens are added to the `MarketManager` contract, so theoretically $tokenDecimals$ is unbounded.

Output Bounds

- $amountAdjusted$ can theoretically be 0, which will cause future calculations to zero out.
- If the collateral token and the debt token contain the same number of decimals, $amountAdjusted = debtAmount$.

- If the collateral token has more decimals than the debt token, expect that the $amountAdjusted > debtAmount$.
- If the collateral token has fewer decimals than the debt token, expect that the $amountAdjusted < debtAmount$.

Liquidated Tokens Calculation

$$liquidatedTokens = \frac{amountAdjusted * debtToCollateralRatio}{WAD}$$

Input Bounds

- $amountAdjusted$ is bounded between 0 and the debt amount scaled by the token decimals.
- $debtToCollateralRatio$ depends on the output of the $debtToCollateralRatio$ calculation. (See that calculation earlier in this section.)

Output Bounds

- If $amountAdjusted = 0$, $liquidatedTokens$ must be 0.
- If $debtToCollateralRatio = 0$, $liquidatedTokens$ must be 0.

Debt Amount and Liquidated Tokens

$collateralAvailable = collateralPostedFor\ (account)$

If $liquidatedTokens > collateralAvailable$, then $liquidatedTokens$ will equal $collateralAvailable$ following this calculation:

$$debtAmount = \frac{debtAmount * collateralAvailable}{liquidatedTokens}$$

G. Code Quality Recommendations

During the invariant development and testing exercise, we noticed a few areas of the codebase that would benefit from clarity, which are outlined in the list below. Some of the recommendations have already been implemented through rebases done throughout the engagement.

- **Rename the `DynamicInterestRateModel.getBorrowWithUpdate` function so that it cannot be interpreted as a getter function.** This function updates the value of `_currentRates`, so it is not a getter function.
- **Document the `_revert` function used throughout the codebase for gas-optimized reversions.** Readers of the codebase would benefit from documentation on the uses of this pattern.
- **Combine the redundant code in the `increaseAmoundAndExtendLockFor` and `extendLock` functions into a helper function.** These functions have similar `if/else` conditions to update continuous locks and extend existing locks; pulling this code into a single helper function would reduce code reuse in the codebase.
- **Use a constant to represent `1e14` or add underscores to the existing constant to make it more readable in the `_bpToWad` conversion in `MarketManager`.** This may make it clearer that the basis points are being multiplied by 1e14 to convert to wad.
- **Reorder the arguments of the `_reduceCollateralIfNecessary` function to match the order of the arguments of the `canRedeemWithCollateralRemoval` function.** Because the functions are related, they would benefit from the same order of arguments.
- **Rename the `processExpiredLock` function to communicate that it also processes locks when the system is shut down.** Technically, this function works on expired locks and on locks when the system is shut down; however, the latter behavior is not communicated in the function name.
- **Fix the incorrect NatSpec comment on the `_canLiquidate` function.** The current comment specifies that the function returns the debt and collateral token price, but it actually returns the amount.
- **Rename variables suffixed with `A` and `B` in the `MarketManager` contract to `soft` and `hard` to provide clarity on the bounds of variables, as of commit `1f51117`.** This will enhance the readability in the codebase. This recommendation was implemented in the `MarketManager` rebase.

- **Change the `liqIncA > MAX_INCENTIVE_LIQUIDATION` check to `liqIncB > MAX_INCENTIVE_LIQUIDATION` in the `MarketManager` contract, as of commit `1f51117`.** The current line checks the incorrect bounds. This recommendation was implemented in the `MarketManager` rebase.

H. Unit/Integration Test Recommendations

MarketManager

- **Apply differential testing to high-level implementation and low-level implementation of all functions.** Due to the complexity of this codebase, having differential tests on all implicit and explicit behavior should be done.
- **Test the correctness of arithmetic operations at each stage of execution.** For example, the tests of liquidation functions test against only the final values of the actual functions and their test versions. However, correctness and bounds checks should be made at each stage, which is challenging as the current system relies on changes made in-place.
- **Test for scenarios involving significantly large deviations in prices for liquidations to ensure that system arithmetic behaves as expected.** These include corner and edge cases that may be hit when undefined behavior occurs.

VeCVE

- **Test for scenarios in which locks expand over multiple epoch durations.** This may be challenging to test only with unit testing, so we recommend continuing to expand tests that were introduced as a result of corner cases from the fuzz testing suite and continuing to expand the difficulty and explored range of such tests.