MythX

Project 0x Case Study

CONTENTS

Abstract

Smart contracts facilitate the transfer of value and help determine digital asset behavior. This results in a higher need for formal proofs and computer-aided checks compared to traditional software as traditional software does not typically perform these functions. 0x is an open protocol that enables the peer-to-peer exchange of assets on the Ethereum blockchain. It is one of the largest open protocols with over 30 projects building on top of it, amassing over 713,000 total transactions, and a volume of $750 million.

On December 2nd 2019, the 0x v3.0 release went live. The release included a significantly more complex exchange environment. As such, 0x sought out MythX, along with ConsenSys Diligence to perform a manual security audit, to increase confidence in the correctness of the smart contract code.

MythX performed the following techniques in the 3.0 branch of the 0x monorepo:

- Run MythX Pro to check each smart contract individually for bugs
- Execute fuzzing campaigns on a live multi-contract environment using the Harvey greybox fuzzer
- Formally verify security and correctness properties of the smart contracts using symbolic execution and greybox fuzzing

As a result:

- 37 potential issues were detected by MythX Pro
- 149 potential issues were detected by MythX's extended greybox fuzzing campaign
- 5 custom contract properties were verified through custom checks

Continuously verifying the code using MythX, including the custom checks built-in this project, was recommended to prevent regressions and new security issues.

Problem statement

0x's network of decentralized exchanges has processed over $750 million since inception. As such, 0x wanted to ensure that no funds would be at risk during the transition from v2.0 to v3.0. The v3.0 release has a more complicated exchange environment due to an increased amount of smart contracts interacting with each other, resulting in more complexities and potentially introducing hidden bugs. 0x's need to deeply analyze the code, integrate continuous analysis into their deployment pipeline, and verify specific contract properties led to a natural partnership with MythX. Both automated analysis and human auditing was conducted to ensure high confidence that the v3.0 release would be secure and bug-free.

"Working with the MythX team solidified our perspective on the effectiveness of fuzz testing, and strengthened the trust in the audit report ConsenSys led on our v3.0 release." **- 0x Team**

Smart contract security solution

MYTHX PRO VULNERABILITY SCAN

MythX Pro, a security analysis tool that detects 26 different types of security vulnerabilities by performing static analysis, dynamic analysis, and symbolic execution, was used to detect smart contract bugs on 197 contracts. Each smart contract was compiled individually and checked against a class of known vulnerabilities from the SWC registry. The SWC registry is a database that contains a list of known smart contract vulnerabilities, with each known vulnerability having its own SWC identifier (ID).

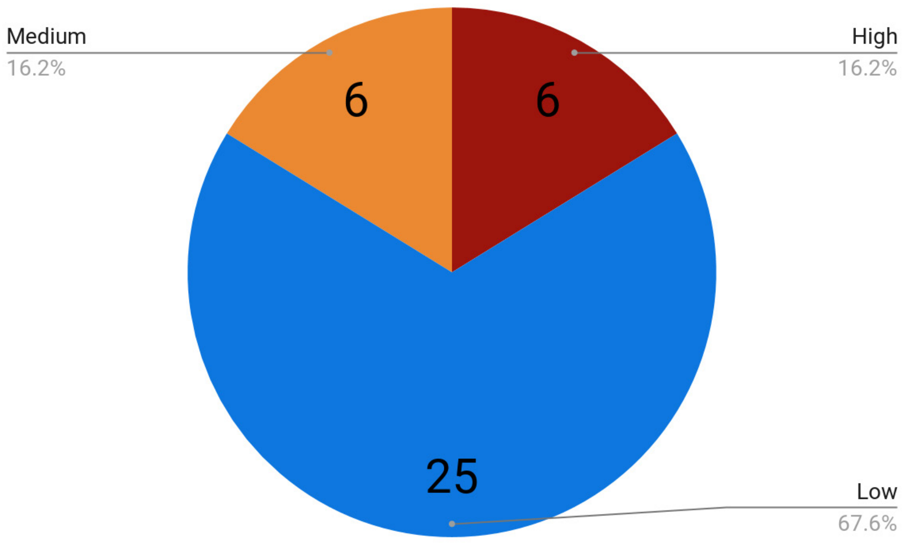
The following table lists the bug classes that were tested for. A checkmark in the "Pass" column indicates that no issues were detected in the category, while an "X" indicates that one or more issues in the category were found.

Table 1: Vulnerabilities checked with MythX Pro

SWC ID	Bug Class	Pass
SWC-100	Function Default Visibility	✓
SWC-101	Integer Overflow and Underflow	✗
SWC-102	Outdated Compiler Version	✓
SWC-103	Floating Pragma	✓
SWC-104	Unchecked Call Return Value	✓
SWC-105	Unprotected Ether Withdrawal	✗
SWC-106	Unprotected SELFDESTRUCT Instruction	✓
SWC-107	Reentrancy	✓
SWC-108	State Variable Default Visibility	✓
SWC-109	Uninitialized Storage Pointer	✓
SWC-110	Assert Violation	✗
SWC-111	Use of Deprecated Solidity Functions	✓
SWC-112	Delegatecall to Untrusted Callee	✓
SWC-113	DoS with Failed Call	✓
SWC-114	Transaction Order Dependence	✓
SWC-115	Authorization through tx.origin	✗
SWC-116	Timestamp Dependence	✗
SWC-118	Incorrect Constructor Name	✓
SWC-119	Shadowing State Variables	✗
SWC-120	Weak Sources of Randomness	✓
SWC-123	Requirement Violation	✓
SWC-124	Write to Arbitrary Storage Location	✓
SWC-127	Arbitrary Jump	✓
SWC-128	Gas Exhaustion	✓
SWC-129	Typographical Error	✓
SWC-130	Right-To-Left-Override Control Character	✓

MythX Pro detected a total of 37 potential issues that needed to be checked. Most findings reflected suspected best practice violations such as variable names shadowing, ignoring failures of external calls, and using tx.origin to determine the control flow. Most of the issues turned out to be expected behavior. However, a few potentially problematic issues such as integer overflows were discovered.

Diagram 1: Vulnerabilities discovered sorted by severity



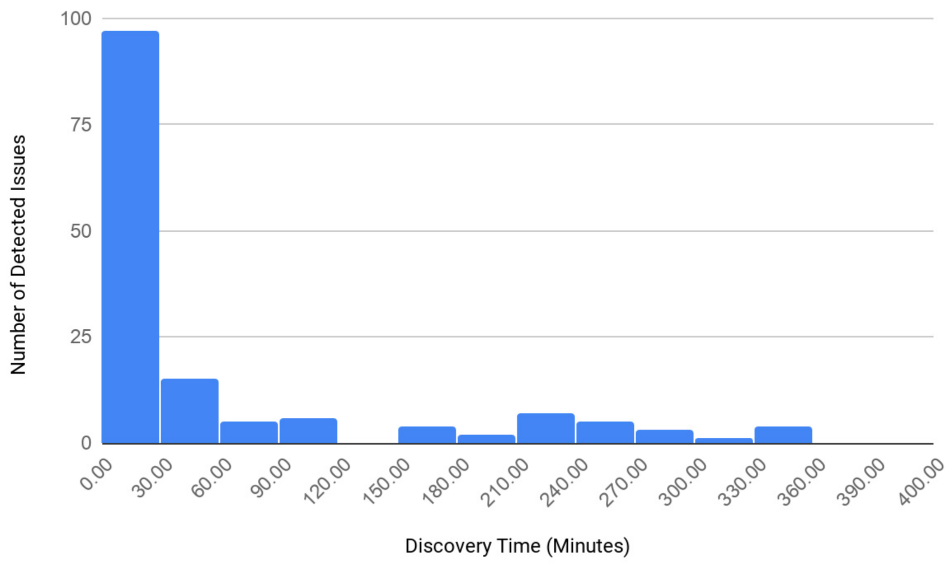
GREYBOX TARGETED FUZZING

Along with using MythX Pro, several 6-hour fuzzing campaigns were executed using the Harvey greybox fuzzer, a lightweight test-generation approach that effectively detects bugs and security vulnerabilities. In order to detect deep security vulnerabilities, Harvey predicts new inputs that are more likely to reveal vulnerabilities in smart contracts, and fuzzes transaction sequences in a targeted and demand-driven way.

All 20 migrated or deployed contracts, and many more contracts via inherited functionality, were fuzzed using Harvey. A custom setup was created wherein all smart contracts were deployed to a Ganache node. This allowed for all the smart contracts and all its dependencies to be analyzed, enabling more elaborate, comprehensive, and accurate fuzzing on the contracts.

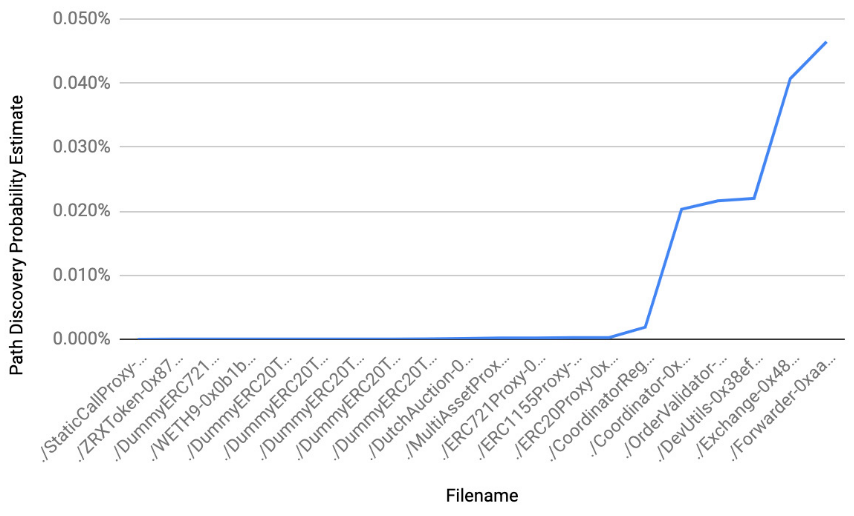
It was also observed that the fuzzer kept finding new issues over time. It was shown that only 65% of the issues were detected within the first 30 minutes, which highlights the importance of long-running custom fuzzing campaigns.

Diagram 2: Issues discovered over 30-minute time periods



In addition, a coverage estimate was generated to estimate the residual risk that a new path or behavior would appear. This is helpful because input fuzzing is a randomized process and does not yield a guarantee that all issues have been discovered. Diagram 4 shows the estimated residual risk for the smart contracts that were analyzed. Having a lower estimated residual risk value indicates that the chances of a new vulnerability or behavior appearing is low.

Diagram 3: Estimated residual risk of the analyzed smart contracts

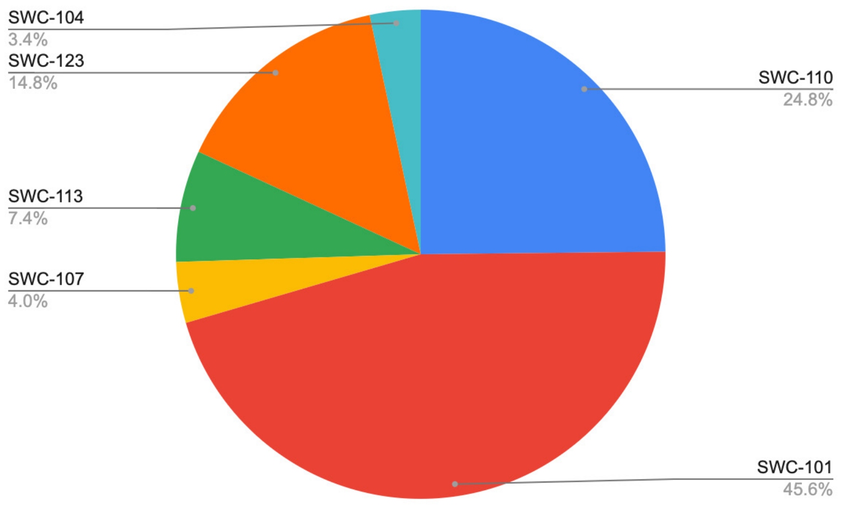


In total, Harvey discovered 149 issues consisting of:

Table 2: Harvey discovered issues

SWC ID	# of Issues
SWC-101 (Integer Overflow and Underflow)	68
SWC-104 (Unchecked Call Return Value)	5
SWC-107 (Reentrancy)	6
SWC-110 (Assert Violation)	37
SWC-113 (DoS with Failed Call)	11
SWC-123 (Requirement Violation)	22
SWC-124 (Write to Arbitrary Storage Location)	0
SWC-127 (Arbitrary Jump with Function Type Variable)	0

Diagram 4: Vulnerabilities found sorted by SWC types



The results were reviewed manually to understand why the issues were flagged, with a greater focus on the issues that were most likely to cause security risks such as re-entrancy and improper handling of external calls. All 22 instances of SWC-104, SWC-107, and SWC-113 were reviewed. The review did not uncover exploitable vulnerabilities. It was also noted that the issues with the highest risk were intended by the 0x developers. For example in

Example 1, the transaction would not be reverted for failing calls, which could result in propagating the failure. However, this issue along with the others detected are mitigated since the contract that could exploit the vulnerabilities is a "trusted" contract, which is another 0x contract.

Example 1: How the transaction will not revert if the call fails

```
For example, the 0x developers intentionally ignored return values in
OrderTransferSimulationUtils.sol (line 117):
(, bytes memory returnData) = address(_EXCHANGE).
call(simulateDispatchTransferFromCallsData);
In LibAssetData.sol (line 82) the return value is not ignored, but the
transaction is not reverted if the call failed:
(bool success, bytes memory returnData) = tokenAddress.
staticcall(balanceOfData);
balance = success && returnData.length == 32 ? returnData.readUint256(0) : 0;
```

VERIFICATION OF CUSTOM PROPERTIES

Custom tests for five security properties were created for the 20 deployed smart contracts to check the intended behavior of specific contracts, also known as functional correctness. Fuzzing, symbolic execution, and SMT solving were used to determine whether the smart contracts behaved correctly with respect to the properties.

Generally, such tests are implemented by inserting runtime assertions into the code. MythX or offline versions of Harvey and Mythril, an analysis tool that performs symbolic execution and SMT solving, are then used to detect counterexamples.

The custom properties were chosen by referring to 0x's design document on bug classes to avoid and to see which bugs were expressible for 0x's codebase. Table 2 shows which properties were checked for. A check mark indicates that this property holds where an "x" indicates detected unintended behavior.

Table 2: List of custom properties checked for against 0x's codebase

Property	Description	Result
Exchange Payment	Verify that the contract account balance is zero ether at the end of a transaction	✓
Asset Transfers Triggered by Unauthorized Sender	Verify that successful asset transfers always originate from an authorized sender	✓
Filling Closed Orders	Verify that only open orders can be filled	✓
Fixed-point Integer Arithmetics	Arithmetic operations on fixed-point signed integers don't overflow	✗
Asset Proxy	Actors cannot execute proxy calls unless they are explicitly listed in the authorities array by the owner	✓

Three instances of integer underflows were detected for the property, Fixed-point Integer Arithmetics.

Example 2: Outputs of three instances of integer underflows

```
_add(0x8000000000000000000000000000000000000000000000000000000000000000,
0x8000000000000000000000000000000000000000000000000000000000000000)
_mul(0x8000000000000000000000000000000000000000000000000000000000000000,
0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)
_div(0xffffffffffffffffffffffffffffffff00000000000000000000000000000000,
0xfffffffffffffffffffffffffffffff
```

In addition to the checks listed above in Table 2, another custom check for a more specific property was created based on an issue discovered by the ConsenSys Diligence audit team. The check verifies the complex state invariants for the MixinStorage contract. After using Harvey, it was determined that the code responsible for the issue was not part of the contracts that were deployed to the mainnet. It will be possible to automatically check the property once this contract is part of the migrated contracts. This would be an example of checking for violations during the course of the development life cycle.

Conclusion

The MythX team was able to detect 37 potential issues using MythX Pro, 149 potential issues using MythX's extended greybox fuzzing campaign, and verified 5 custom contract properties through custom checks. To prevent regression and newly introduced security bugs, continuously verifying the code using MythX, including the custom checks built in this project, was recommended.

TECHNICAL MILESTONES

The completion of the 0x project performed by both MythX and ConsenSys Diligence not only resulted in high confidence that the upcoming 0x v3.0 release will be secure, it has also produced a significant milestone for Ethereum security.

- MythX Pro is the first automated security analysis tool that is able to perform both symbolic execution and fuzzing on a major project
- Comprehensive coverage and realistic analysis and testing was performed by deploying smart contracts to a local testnet and extracting information to create the initial state (deployed state) for the greybox fuzzing campaign
- The estimated residual risk on the likelihood that a new path or behavior would appear was calculated on smart contracts analyzed by our Harvey greybox fuzzer

TECHNOLOGY PARTNERSHIP

MythX

About MythX

MythX, a ConsenSys product, scans for security vulnerabilities in Ethereum smart contracts. Its comprehensive range of analysis techniques which include static analysis, dynamic analysis, and symbolic execution, accurately detects security vulnerabilities and provides an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, MythX can be utilized in all phases of the smart contract development lifecycle.

For more information, visit mythx.io

CONSENSYS
Diligence

About ConsenSys Diligence

Our smart contract auditing and blockchain security services are delivered by a highly experienced team, driven by their passion for enabling more secure platforms and ecosystems. ConsenSys Security auditors and researchers are distributed all over the world and focused on creating tools that are truly helpful to auditors and smart contract developers.

For more information, visit diligence.consensys.net

MythX

"Working with the MythX team solidified our perspective on the effectiveness of fuzz testing, and strengthened the trust in the audit report ConsenSys led on our v3.0 release."

0x Team