

TÀI LIỆU CÔNG KHAI

# BÁO CÁO KIỂM THỬ AN NINH ỨNG DỤNG

## Phiên bản v1.3

Tài liệu này trình bày chi tiết quá trình thực hiện và kết quả của dự án kiểm thử an ninh hợp đồng thông minh được thực hiện bởi CyStack từ ngày 30/03/2022 đến 14/04/2022.

Dành cho

**Nền tảng NFT5**

Thực hiện bởi

**Công ty Cổ phần CyStack Việt Nam**

© 2022 CyStack. Bản quyền thuộc về CyStack.

Toàn bộ văn bản và các văn mẫu sử dụng trong quá trình tạo ra văn bản là sản phẩm bản quyền của CyStack và không được sử dụng (toàn bộ hay một phần) bên ngoài khi chưa có sự đồng ý từ CyStack.

CyStack và (các) tác giả văn bản không chịu trách nhiệm trước bất cứ lỗi, thiếu sót hay tổn hại xảy ra trong quá trình sử dụng các thông tin tại văn bản này. Việc sử dụng các dịch vụ của CyStack không đảm bảo việc hệ thống luôn đạt trạng thái an toàn bảo mật, hay các cuộc tấn công mạng sẽ không xảy ra.

# Mục lục

<b>1 Tổng quan</b>	<b>4</b>
1.1 Chi tiết dự án kiểm thử an ninh . . . . .	4
1.2 Mục tiêu kiểm thử an ninh . . . . .	7
1.3 Phương pháp kiểm thử an ninh . . . . .	7
1.4 Phạm vi kiểm thử an ninh . . . . .	10
<b>2 Tóm tắt báo cáo</b>	<b>11</b>
<b>3 Chi tiết lỗ hổng bảo mật</b>	<b>13</b>
<b>4 Kết luận</b>	<b>16</b>
<b>5 Danh mục phụ lục</b>	<b>17</b>
Phụ lục A – Định nghĩa về các tình trạng lỗ hổng . . . . .	17
Phụ lục B – Mức độ nghiêm trọng của lỗ hổng . . . . .	18
Phụ lục C – Phân loại lỗ hổng theo Smart Contract Weakness Classification Registry (SWC Registry) . . . . .	19
Phụ lục D – Phân loại lỗ hổng theo Related Common Weakness Enumeration (CWE) . . . . .	24

## Tuyên bố miễn trừ trách nhiệm

Dự án Kiểm thử an ninh Hợp đồng thông minh chỉ cung cấp các phát hiện và khuyến cáo cho mã nguồn các hợp đồng đã được xác định nội dung duy nhất. Vì vậy, các kết quả có thể không đảm bảo tính chính xác đối với các đoạn mã nguồn tương ứng nhưng ngoài phạm vi đã xác định, hay đối với các đoạn mã nguồn có sự thay đổi nhất định hoặc cập nhật so với phiên bản đã tiến hành kiểm thử. Kết quả kiểm thử cũng không khẳng định việc hợp đồng thông minh hoàn toàn không chứa lỗ hổng, và cũng không loại trừ khả năng phát hiện mới các lỗ hổng trên hệ thống hợp đồng thông minh.

Những tương tác của quá trình kiểm thử an ninh bị giới hạn trong một khoảng thời gian, vì vậy kết quả không đảm bảo việc phát hiện tất cả lỗ hổng cho hợp đồng thông minh. CyStack ưu tiên kiểm tra những lỗ hổng nguy hiểm nhất có thể bị kẻ tấn công khai thác. CyStack khuyến nghị NFT5 thực hiện bài kiểm tra này định kỳ, sử dụng đội ngũ kiểm thử viên nội bộ hoặc bên thứ ba để đảm bảo kiểm soát bảo mật liên tục và tốt nhất.

Kết quả đánh giá an ninh hợp đồng thông minh không dùng cho các mục đích tư vấn đầu tư.

## Lịch sử thay đổi

Phiên bản	Ngày hoàn thiện	Ghi chú
1.0	05/04/2022	Hoàn thành bài đánh giá an toàn cho hợp đồng AssetHolder.sol. Không có phát hiện lỗ hổng.
1.1	07/04/2022	NFT5 yêu cầu xuất báo cáo công khai cho đợt kiểm thử an ninh thứ nhất.
1.2	14/04/2022	Hoàn thành việc đánh giá an ninh cho tất cả các hợp đồng thông minh phía NFT yêu cầu. Một phát hiện được khắc phục ngay lập tức, phát hiện còn lại chưa khắc phục.
1.3	15/04/2022	Toàn bộ các lỗ hổng đã được khắc phục. NFT5 cho phép truy cập công khai đối với báo cáo này.

## Thông tin liên hệ

Tổ chức	Đại diện	Chức vụ	Địa chỉ email
NFT5	Nguyễn Hoàng Triều	Product Owner	trieunguyen@nft5.io
CyStack	Võ Huyền Nhi	Sales Manager	nhivh@cystack.net
CyStack	Nguyễn Ngọc Anh	Sales Executive	anhntn@cystack.net

## Danh sách kiểm thử viên

Họ tên	Chức vụ	Địa chỉ email
Nguyễn Hữu Trung	Head of Security	trungnh@cystack.net
Hà Minh Châu	Auditor	
Vũ Hải Đăng	Auditor	
Nguyễn Văn Huy	Auditor	
Nguyễn Trung Huy Sơn	Auditor	
Nguyễn Bá Anh Tuấn	Auditor	

# Tổng quan

Từ 30/03/2022 đến 14/04/2022, CyStack đã thực hiện bài đánh giá an ninh hợp đồng thông minh cho Nền tảng NFT5 theo yêu cầu của NFT5. Các phát hiện và khuyến cáo từ CyStack đều được chỉ rõ trong báo cáo này.

## 1.1 Chi tiết dự án kiểm thử an ninh

### Đối tượng kiểm thử

NFT5 là một nền tảng phát hành NFT thế hệ mới. NFT phát hành trên NFT5 sẽ đại diện cho một phần quyền sở hữu và thu nhập của sản phẩm sở hữu trí tuệ. NFT5 ra đời nhằm cung cấp giải pháp kết nối nhà sáng tạo nội dung và nhà đầu tư một cách đơn giản và thuận tiện nhất.

NFT5 được xây dựng và phát triển bởi đội ngũ của Remitano, thuộc NFT5.

Thông tin cơ bản về dự án được liệt kê trong bảng dưới đây:

Đề mục	Thông tin
Tên dự án	Nền tảng NFT5 (GitHub: remitano/nft5-core)
Nhà phát hành	NFT5
Website	<a href="https://nft5.io/">https://nft5.io/</a>
Nền tảng	Binance Smart Contract
Ngôn ngữ	Solidity
Mã nguồn	<a href="https://github.com/remitano/nft5-core/tree/409747c7758d4a2b52e39a700a1f57fcd377dc57/contracts">https://github.com/remitano/nft5-core/tree/409747c7758d4a2b52e39a700a1f57fcd377dc57/contracts</a>
Số commit	409747c7758d4a2b52e39a700a1f57fcd377dc57
Phương pháp kiểm thử	Whitebox

ERC-1155 là chuẩn token được đội ngũ phát triển nền tảng NFT5 lựa chọn để vận hành xuyên suốt hệ thống NFT5. ERC-1155 là một chuẩn giúp hợp đồng thông minh quản lý nhiều loại token. Một smart contract áp dụng chuẩn ERC-1155 có thể xử lý nhiều loại token khác nhau như fungible, non-fungible hay semi-fungible.

Mô hình hoạt động của hệ sinh thái NFT5 được minh họa ở hình dưới đây:



Danh sách các tính năng sẽ ra mắt tại NFT5:

- **Phát hành NFT (Đã hoàn tất)**: Đây là chức năng giúp các nhà sáng tạo nội dung phát hành các NFT của mình thông qua NFT5;
- **Sàn giao dịch (Đã hoàn tất)**: Đây là nơi lưu thông toàn bộ tài sản mã hóa độc nhất (NFT) bằng các hợp đồng thông minh chất lượng, an toàn và bảo mật;
- **Staking NFT (Đang phát triển)**: Khi nhà đầu tư khóa NFT của mình vào Staking pool của NFT5 sẽ nhận được phần thưởng từ doanh thu của nền tảng;
- **Đấu giá (Đang phát triển)**: Các NFT được đấu giá sẽ tăng tính cạnh tranh, các nhà đầu tư sẽ thích thú hơn và tạo động lực cho thị trường mua bán.

Trong phạm vi của dự án kiểm thử an ninh này CyStack chỉ tiến hành đánh giá mức độ an toàn của hai tính năng: **Phát hành NFT** và **Sàn giao dịch**. Có bốn hợp đồng thông minh được tạo ra cho việc vận hành các tính năng này trên nền tảng NFT5:

- AssetHolder.sol;
- FeeCollector.sol;
- MarketPlace.sol;
- MintNFT.sol.

Trong giai đoạn đầu tiên của dự án kiểm thử, CyStack tập trung phân tích hợp đồng **AssetHolder.sol** và tìm kiếm các lỗ hổng bảo mật có thể tồn tại ở hợp đồng này. Hợp đồng AssetHolder.sol hoạt động như một kho lưu trữ BUSD nạp vào từ người dùng trên nền tảng NFT5. Đồng thời, hợp đồng này cũng được người dùng cấp quyền (approve) cho phép thực hiện các giao dịch liên quan đến NFT, phục vụ tính năng MarketPlace.

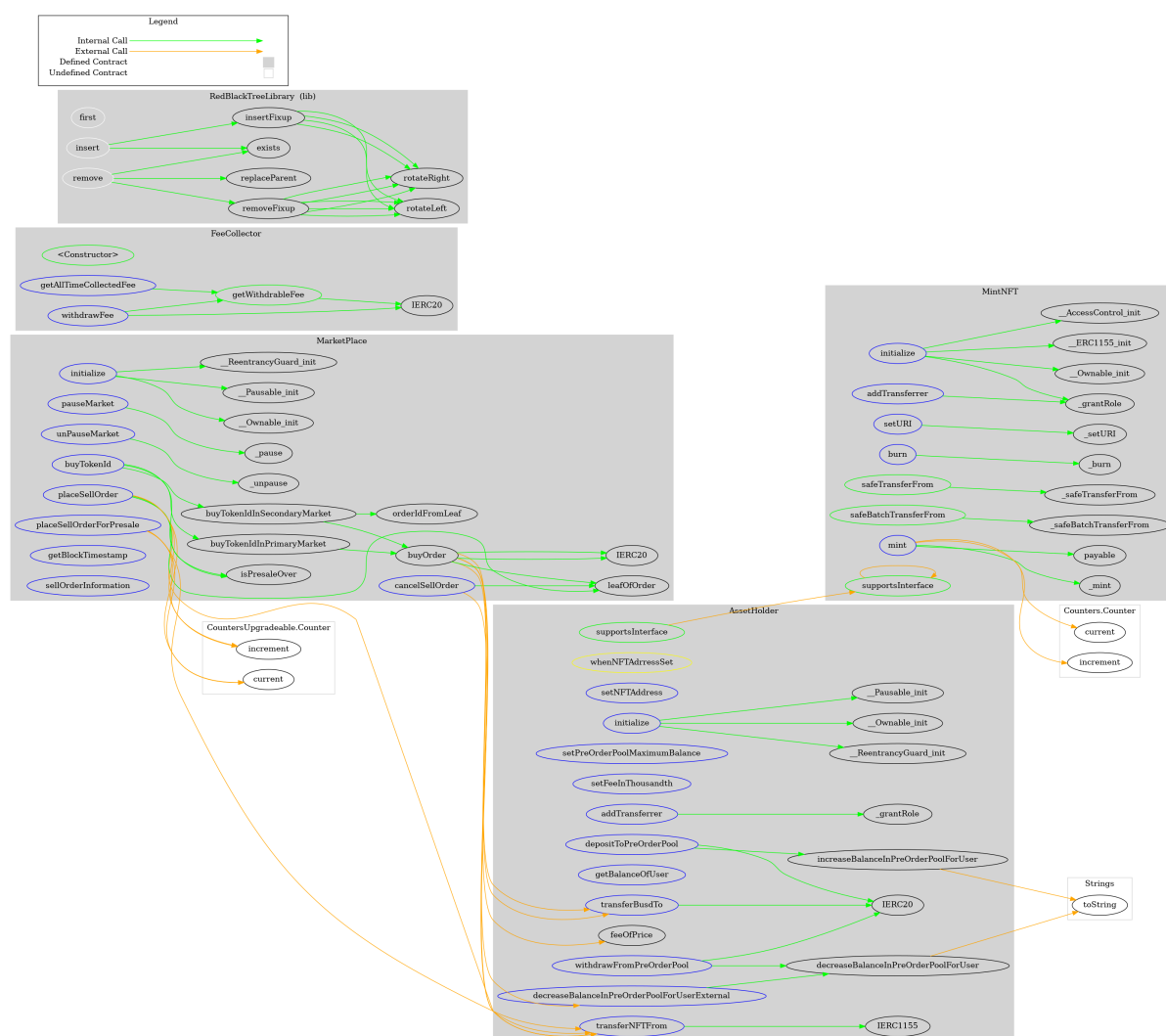
Trong giai đoạn thứ hai của quá trình kiểm thử, CyStack tiến hành các bài đánh giá an ninh cho ba hợp đồng thông minh **FeeCollector.sol**, **MarketPlace.sol** và **MintNFT.sol**.

Hợp đồng MarketPlace.sol cho phép người dùng thực hiện mua và bán NFT. Người bán có thể tạo một lệnh bán NFT của mình với số lượng và giá tùy ý. Khi người mua thực hiện mua NFT, người mua sẽ được ưu tiên giá từ thấp đến cao. Người mua cũng có thể chọn sử dụng tài khoản trong pool (AssetHolder) để thực hiện mua NFT. Ngoài ra, owner của hợp đồng có thể tạo các private sale NFT. Owner có thể pause/unpause đối với hợp đồng.

Hợp đồng MintNFT.sol thực hiện các chức năng liên quan đến việc khởi tạo, đốt và giao dịch NFT trên hệ thống NFT5. Hợp đồng được thiết kế với hai quyền là admin và transferer. Chỉ có admin mới có quyền khởi tạo và đốt các NFT.

Hợp đồng FeeCollector.sol thực hiện thu phí cho các giao dịch tại MarketPlace.sol. Lượng phí thu được này sẽ được lưu trữ trực tiếp trên contract FeeCollector.sol. Chỉ có owner của contract mới có quyền tương tác với contract này, và owner có thể thực hiện xem các thông tin về lượng phí thu được, cũng như thực hiện rút số tiền phí thu được từ contract này và về ví của owner.

Các tương tác gọi hàm trong toàn bộ dự án NFT5 được thể hiện ở lược đồ dưới đây:



## Đơn vị cung cấp giải pháp kiểm thử an ninh

CyStack là công ty hàng đầu trong lĩnh vực an ninh mạng, an toàn thông tin tại Việt Nam. Chúng tôi xây dựng sản phẩm và giải pháp bảo mật cho tổ chức, doanh nghiệp để chống lại rủi ro an ninh trong thời đại số. CyStack hiện là thành viên của Hiệp hội an toàn an ninh mạng Việt Nam và Liên minh Phát triển hệ sinh thái an toàn và an ninh mạng Việt Nam.

Các chuyên gia của CyStack thường xuyên có các bài tham luận tại các diễn đàn bảo mật lớn nhất thế giới và khu vực như BlackHat USA, BlackHat Asia, Xcon, T2FI, v.v., đồng thời đã phát hiện nhiều lỗi bảo mật nghiêm trọng trong các sản phẩm và có tên trong Bảng vàng danh vọng - Hall of Fame của các tập đoàn công nghệ quốc tế.

## 1.2 Mục tiêu kiểm thử an ninh

Trọng tâm của việc kiểm thử an ninh nhằm khẳng định hệ thống hợp đồng thông minh được triển khai an toàn, có sức chống chịu và hoạt động tuân theo tài liệu mô tả kỹ thuật đã đề ra. Hoạt động kiểm thử an ninh hợp đồng mạng tập trung vào các vấn đề sau:

1. **Tính bảo mật:** Phát hiện các vấn đề bảo mật trong từng hợp đồng và trong hệ thống các hợp đồng thông minh.
2. **Kiến trúc an toàn:** Kiến trúc của hệ thống các hợp đồng thông minh được đánh giá dựa trên tập các quy chuẩn thiết kế an toàn hợp đồng thông minh, cũng như phần mềm nói chung.
3. **Tính chuẩn xác và chất lượng mã nguồn:** Kiểm tra đánh giá toàn bộ mã nguồn hợp đồng thông minh. Những tiêu chí cần thực hiện đánh giá bao gồm:
  - Tính chuẩn xác
  - Tính dễ đọc
  - Bố cục hợp lý cho bộ mã nguồn có độ phức tạp cao
  - Nâng cao tính mở rộng
  - Số lượng và chất lượng test coverage

## 1.3 Phương pháp kiểm thử an ninh

Để chuẩn hóa quy trình đánh giá, chúng tôi xây dựng các định nghĩa sau đây dựa trên phương pháp đánh giá rủi ro của OWASP (OWASP Risk Rating Methodology):

- **Khả năng (Likelihood)** thể hiện xác suất mà lỗi hỏng được bí mật sử dụng khai thác;
- **Mức độ ảnh hưởng (Impact)** đo lường mất mát kỹ thuật và thiệt hại kinh tế của một vụ tấn công thành công;
- **Mức độ nghiêm trọng (Severity)** thể hiện mức độ nghiêm trọng khi rủi ro xảy ra.



Khả năng và mức độ ảnh hưởng được chia ra làm 3 mức độ: High, Medium và Low, hay tương ứng, H, M và L. Mức độ nghiêm trọng được đánh giá dựa vào khả năng và mức độ ảnh hưởng, từ đó phân thành năm mức độ tương ứng là Critical, Major, Medium, Minor và Thông tin (Info), được thể hiện ở bảng dưới đây:

Impact	High	Critical	Major	Medium
	Medium	Major	Medium	Minor
	Low	Medium	Minor	Info
		High	Medium	Low
		Likelihood		

CyStack bước đầu phân tích hợp đồng thông minh với các bộ công cụ đánh giá bảo mật mã nguồn mở và các công cụ do CyStack tự phát triển để tìm kiếm các lỗi liên quan đến hợp đồng thông minh nói chung. Các công cụ này bao gồm Slither, securify, Mythril, Sūrya, Solgraph, Truffle, Geth, Ganache, Mist, Metamask, solhint, mythx, v.v. Sau đó, các chuyên gia bảo mật sẽ kiểm tra thủ công kết quả từ các công cụ quét, ghi lại các mô tả về lỗ hổng và quyết định mức độ nghiêm trọng của chúng.

Bước tiếp theo, CyStack rà soát các lỗ hổng có thể xảy ra mà các công cụ quét tự động không thể phát hiện dựa trên một checklist tự xây dựng, dựng các test case cho từng loại lỗ hổng và chỉ ra mức độ an toàn trước các lỗ hổng từ kết quả phân tích. Nếu không có lỗ hổng được phát hiện sau khi tiến hành phân tích thủ công, hợp đồng thông minh có thể được coi là an toàn trong phạm vi các lỗ hổng thực hiện test case. Trong trường hợp có phát hiện lỗ hổng, CyStack sẽ tiếp tục triển khai các hợp đồng thông minh trên testnet riêng và chạy thử để khẳng định các phát hiện lỗ hổng. Thêm vào đó, chúng tôi có thể đưa ra PoC chứng minh khả năng tấn công bằng lỗ hổng được phát hiện, nếu được yêu cầu hoặc là cần thiết.

Checklist tiêu chuẩn sử dụng cho mọi quá trình kiểm thử an ninh hợp đồng thông minh luôn chặt chẽ tuân thủ theo danh mục Smart Contract Weakness Classification Registry (SWC Registry). Danh mục SWC Registry được xây dựng dựa trên mô hình phân loại các lỗ hổng được đề xuất trong dự án The Ethereum Improvement Proposal với mã tra cứu EIP-1470. Checklist các bài kiểm tra tuân theo danh mục SWC Registry được gửi kèm báo cáo trong Phụ lục C.

Nói chung, quá trình kiểm thử an ninh tập trung tìm kiếm và kiểm tra sự tồn tại của các vấn đề dưới đây:

- **Các vấn đề liên quan đến quy chuẩn lập trình:** Tập trung tìm kiếm các lỗi lập trình dựa trên các quy chuẩn và phương pháp lập trình hợp đồng thông minh nói chung.
- **Các vấn đề ở thiết kế tổng quan:** Kiểm tra thiết kế kiến trúc hợp đồng thông minh và thực hiện các test case, như tấn công self-DoS, thực hiện thừa kế không đúng, v.v.

- **Các vấn đề liên quan đến lập trình an toàn:** Tìm kiếm các lỗ hổng bảo mật phổ biến ở hợp đồng thông minh như integer overflows, cơ chế xác thực thiếu an toàn, sử dụng chữ ký mật mã chưa phù hợp, v.v.
- **Các vấn đề về thiết kế lập trình:** Kiểm tra tính logic của code và cách xử lý lỗi trong hợp đồng thông minh, có thể kể đến việc khởi tạo biến trong hợp đồng, kiểm soát số dư và luồng giao dịch token, kiểm tra các hàm ngẫu nhiên, v.v.
- **Các nguy cơ tiềm ẩn khác trong mã nguồn:** Làm rõ các vấn đề đặc biệt khác, như tính bảo mật dữ liệu, độ tin cậy dữ liệu, tối ưu gas tiêu hao, một số các test case về xác thực và quyền chủ (owner) hợp đồng, các hàm fallback, v.v.

Để có cái nhìn toàn vẹn về chi tiết và mức độ nghiêm trọng của các vấn đề được phát hiện, mỗi mã SWC đều được trỏ về một mã CWE tương đồng nhất của danh mục Common Weakness Enumeration. CWE là một hệ thống phân loại dành cho các thiếu sót và lỗ hổng trong phần mềm nói chung, hỗ trợ định dạng các lỗi phát hiện ở một phần mềm nhất định. Danh sách ở phụ lục D chỉ ra những lỗ hổng trong phần mềm thông dụng cũng xảy ra trong lập trình hợp đồng thông minh.

Báo cáo tổng hợp kết quả cuối cùng sẽ được gửi tới nhà phát hành hợp đồng thông minh với phần tóm tắt và kết quả chi tiết, để đưa ra được các giải pháp khắc phục phù hợp.

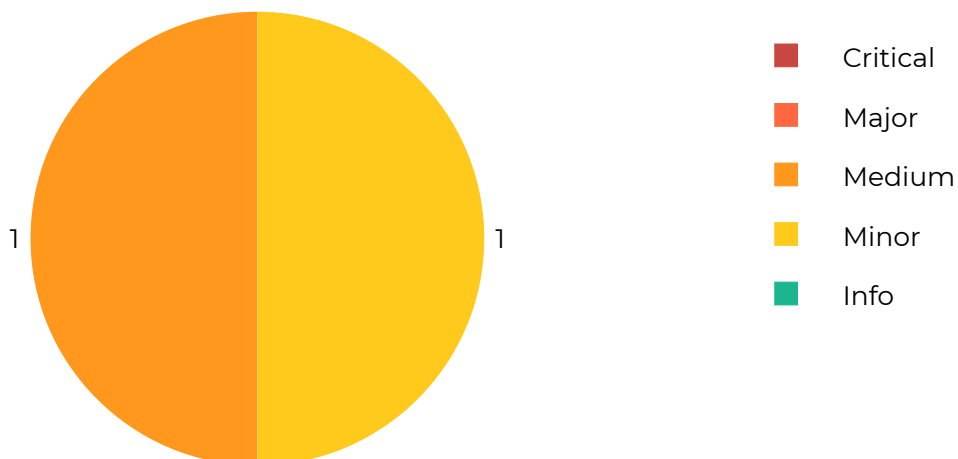
## 1.4 Phạm vi kiểm thử an ninh

Phương pháp	Đối tượng	Loại hình
<b>Các đối tượng ban đầu</b>		
White-box testing	<a href="#">AssetHolder.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">FeeCollector.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">MarketPlace.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">MintNFT.sol</a>	Tệp mã nguồn Solidity
<b>Các đối tượng lần vá lỗi thứ nhất</b>		
White-box testing	<a href="#">AssetHolder.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">FeeCollector.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">MarketPlace.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">MintNFT.sol</a>	Tệp mã nguồn Solidity
<b>Các đối tượng lần vá lỗi thứ hai</b>		
White-box testing	<a href="#">AssetHolder.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">FeeCollector.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">MarketPlace.sol</a>	Tệp mã nguồn Solidity
White-box testing	<a href="#">MintNFT.sol</a>	Tệp mã nguồn Solidity

# Tóm tắt báo cáo

## Số lượng lỗi hỏng theo mức độ nguy hiểm

### Legend



## Số lượng lỗi hỏng theo phân loại SWC

Integer Overflow and Underflow (SWC-101)

2



## Số lượng lỗi hỏng theo phân loại CWE

Incorrect Calculation (CWE-682)

2



## Danh sách lỗi hỏng

ID	Status	Vulnerability	Severity
#nft5-001	Đã khắc phục	Giá trị order.price không được kiểm soát dẫn tới việc hàm orderIdFromLeaf hoạt động sai	MEDIUM
#nft5-002	Đã khắc phục	Giá trị order.id không được kiểm soát dẫn tới việc hàm orderIdFromLeaf hoạt động sai	MINOR

## Khuyến cáo

Dựa vào kết quả của bài kiểm tra an ninh hợp đồng thông minh, CyStack đưa ra những khuyến nghị chính sau đây:

Một số khuyến cáo	
Vấn đề	<p>CyStack đã triển khai xong giai đoạn thứ nhất của dự án kiểm thử hợp đồng thông minh dành cho Nền tảng NFT5 thuộc NFT5. Không có lỗ hổng được phát hiện đối với AssetHolder.sol.</p> <p>Trong giai đoạn thứ hai, CyStack tiến hành các bài đánh giá an ninh đối với ba hợp đồng thông minh FeeCollector.sol, MarketPlace.sol và MintNFT.sol. Có tất cả hai vấn đề được phát hiện: một vấn đề nghiêm trọng với mức độ Medium đã được chấp nhận và được khắc phục ngay lập tức bởi NFT5, một lỗ hổng còn lại mang mức độ nghiêm trọng Low và chưa thể khắc phục do bản chất thiết kế MarketPlace.sol.</p>
Khuyến nghị	<p>CyStack khuyến nghị NFT5 đánh giá kết quả kiểm thử đối chiếu với nhiều bên thứ ba cung cấp giải pháp an ninh để có kết luận chính xác nhất.</p>
Tham khảo	<ul style="list-style-type: none"><li>• <a href="https://consensys.github.io/smart-contract-best-practices/known_attacks">https://consensys.github.io/smart-contract-best-practices/known_attacks</a></li><li>• <a href="https://consensys.github.io/smart-contract-best-practices/recommendations/">https://consensys.github.io/smart-contract-best-practices/recommendations/</a></li><li>• <a href="https://medium.com/@knownsec404team/ethereum-smart-contract-audit-checklist-ba9d1159b901">https://medium.com/@knownsec404team/ethereum-smart-contract-audit-checklist-ba9d1159b901</a></li></ul>

# Chi tiết lỗ hổng bảo mật

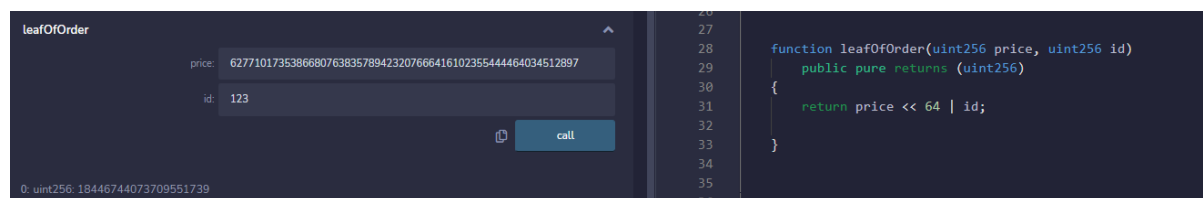
## 1. Giá trị *order.price* không được kiểm soát dẫn tới việc hàm *orderIdFromLeaf* hoạt động sai

Mã số	#nft5-001
Danh mục	SWC-101 - Integer Overflow and Underflow
Mô tả	Hai hàm <i>leafOfOrder</i> và <i>orderIdFromLeaf</i> thiết kế xử lý các phép tính bitwise bao gồm SHIFT LEFT («), OR ( ) và AND &. Tuy nhiên, các hàm này không thực hiện kiểm tra giá trị của <i>order.price</i> . Khi giá trị <i>order.price</i> vượt quá $2^{192}$ , xảy ra các sai lệch tính toán trong các hàm <i>leafOfOrder</i> và <i>orderIdFromLeaf</i> .
Mức độ nghiêm trọng	MEDIUM
Vị trí lỗ hổng	MarketPlace.sol: 333-343
Trạng thái lỗ hổng	Đã khắc phục
Mã tham chiếu	CWE-682 - Incorrect Calculation
Phương pháp khắc phục	Đặt câu lệnh require để kiểm soát giá trị của <i>order.price</i> . Khi giá trị này vượt quá $2^{192}$ , trả về thông báo lỗi.

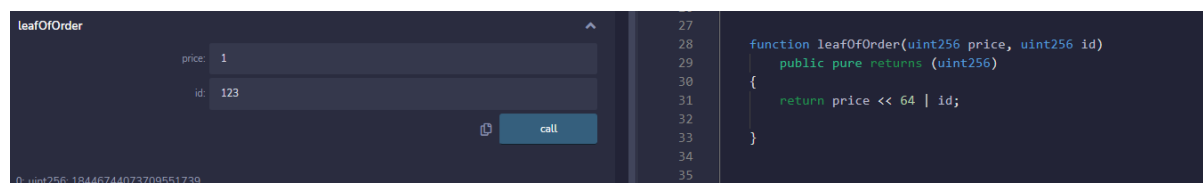
### Mô tả

Khi giá thực của price  **$order.price = 2^{192} + 1 =$**

**6277101735386680763835789423207666416102355444464034512897** và  **$order.id = 123$** , giá trị trả về bởi hàm *leafOfOrder* là **18446744073709551739**:



Tuy nhiên,  **$order.price = 1$**  và  **$order.id = 123$** , giá trị trả về bởi hàm *leafOfOrder* cũng là **18446744073709551739**:



Trong hai trường hợp, với giá trị **order.price** khác nhau, giá trị trả về bởi hàm **leafOfOrder** là như nhau. Khi đó, **order.price** bằng  $2^{192} + 1$  sẽ được sắp xếp trước các **order.price** khác lớn hơn 1, dẫn đến chương trình hoạt động sai.

Đoạn code xảy ra lỗi:

```
...
333     function leafOfOrder(Order memory order)
334         internal pure returns (uint256)
335     {
336         return order.price << 64 | order.id;
337     }
338
339     function orderIdFromLeaf(uint256 leaf)
340         internal pure returns (uint256)
341     {
342         return leaf & ((uint256(1) << 64) - 1);
343     }
...
```

Có thể khắc phục lỗi như sau:

```
...
333     function leafOfOrder(Order memory order)
334         internal pure returns (uint256)
335     {
336         require(order.price < (uint256(1)) << 192, "Value exceeded.");
337         return order.price << 64 | order.id;
338     }
339
340     function orderIdFromLeaf(uint256 leaf)
341         internal pure returns (uint256)
342     {
343         return leaf & ((uint256(1) << 64) - 1);
344     }
...
```

## 2. Giá trị *order.id* không được kiểm soát dẫn tới việc hàm *orderIdFromLeaf* hoạt động sai

Mã số	#nft5-002
Danh mục	SWC-101 - Integer Overflow and Underflow
Mô tả	Hai hàm <i>leafOfOrder</i> và <i>orderIdFromLeaf</i> thiết kế xử lý các phép tính bitwise bao gồm SHIFT LEFT («), OR ( ) và AND &. Tuy nhiên, các hàm này không thực hiện kiểm tra giá trị của <i>order.id</i> . Khi giá trị <i>order.id</i> vượt quá $2^{64}$ , xảy ra các sai lệch tính toán trong các hàm <i>leafOfOrder</i> và <i>orderIdFromLeaf</i> .
Mức độ nghiêm trọng	MINOR
Vị trí lỗ hổng	MarketPlace.sol: 333-343
Trạng thái lỗ hổng	Đã khắc phục
Mã tham chiếu	CWE-682 - Incorrect Calculation
Phương pháp khắc phục	Do bản chất giá trị <i>order.id</i> sinh từ hợp đồng thông minh và không thể thiết đặt thủ công, NFT5 nên có phương án thiết kế khác cho các hàm trên để tránh các vấn đề tính toán sai lệch này.

### Description

Khi giá trị thực của id  **$order.id = 2^{64} + 1 = 1844674425817699235309551617$**  và ***order.price* = 123**, giá trị trả về từ hàm ***leafOfOrder*** là **2268949521066274848769**. Tuy nhiên, khi thực hiện hàm ***orderIdFromLeaf*** với giá trị này, giá trị ***order.id*** trả về lại bằng **1**:

leafOfOrder

price: 123

id: 18446744073709551617

call

0: uint256: 2268949521066274848769

orderIdFromLeaf

leaf: 2268949521066274848769

call

0: uint256: 1

```

26
27
28 function leafOfOrder(uint256 price, uint256 id)
29     public pure returns (uint256)
30 {
31     return price << 64 | id;
32 }
33
34
35
36
37
38 function orderIdFromLeaf(uint256 leaf)
39     public pure returns (uint256)
40 {
41     return leaf & ((uint256(1) << 64) - 1);
42 }
43
44

```



# Kết luận

CyStack đã thực hiện đánh giá an ninh cho các hợp đồng thông minh của NFT5. Có tất cả hai phát hiện, và những phát hiện này không phải các lỗ hổng hay lỗi ở mức độ nghiêm trọng. Sau khi tiến hành re-test cho các hợp đồng thông minh của NFT5, CyStack khẳng định tất cả các lỗ hổng đã được khắc phục hoàn toàn. Không có phát hiện mới cho các phần thay đổi bổ sung trong các hợp đồng thông minh. Về tổng thể, các hợp đồng được kiểm thử đã được triển khai tuân thủ các nguyên tắc thiết kế hợp đồng thông minh an toàn và vượt qua bài đánh giá an ninh cho hợp đồng thông minh từ phía CyStack.

Để nâng cao chất lượng của báo cáo này, cũng như chất lượng của các báo cáo kiểm thử an ninh cho hợp đồng thông minh của CyStack nói chung, chúng tôi rất sẵn lòng với bất kỳ các góp ý hoặc đề xuất đối với phương pháp thực hiện, các phát hiện trong quá trình kiểm thử, hoặc thiếu sót trong phạm vi kiểm thử.

# Danh mục phụ lục

## Phụ lục A – Định nghĩa về các tình trạng lỗ hổng

Tên tình trạng	Định nghĩa
Chưa ghi nhận	Vấn đề bảo mật đã được báo cáo và hiện thời đang được đánh giá bởi các lập trình viên hoặc nhà phát hành hợp đồng thông minh.
Chưa khắc phục	Vấn đề bảo mật được công nhận là lỗ hổng và được lên kế hoạch khắc phục trong thời gian tới. Ở thời điểm của báo cáo hiện hành, lỗ hổng chưa được khắc phục.
Đã khắc phục	Vấn đề bảo mật được công nhận là lỗ hổng và đã được khắc phục hoàn toàn bởi các lập trình viên hoặc nhà phát hành hợp đồng thông minh.
Từ chối	Vấn đề chỉ ra không phải là một lỗ hổng bảo mật hoặc không có hay rất ít ảnh hưởng tới tính an ninh của hợp đồng thông minh, vì vậy không được ghi nhận và không cần khắc phục.

## Phụ lục B – Mức độ nghiêm trọng của lỗ hổng

Mức độ	Định nghĩa
<b>CRITICAL</b>	Những vấn đề được ghi nhận critical thường khi chúng là các lỗi hoặc lỗ hổng bảo mật có thể khai thác trực tiếp. Những vấn đề này được khuyến cáo khắc phục ngay lập tức để phòng tránh các hậu quả đặc biệt nghiêm trọng và việc hệ thống hợp đồng thông minh ngưng hoạt động.
<b>MAJOR</b>	Những vấn đề đánh dấu major thường là các lỗi và lỗ hổng mà không thể khai thác trực tiếp khi thiếu một số điều kiện nhất định. Việc tiến hành vá những vấn đề bảo mật này nên thực hiện càng sớm càng tốt, do có khả năng rất cao chúng sẽ gây ra các vấn đề bảo mật nghiêm trọng và có thể gây ra các vấn đề liên quan đến vận hành hệ thống.
<b>MEDIUM</b>	Những vấn đề với mức độ nghiêm trọng medium thường là các vấn đề, lỗi và lỗ hổng tồn tại nhưng không thể khai thác khi bỏ qua một số thao tác, ví dụ social engineering. Đối với những vấn đề này, chúng tôi khuyến nghị lập kế hoạch ứng phó và tiến hành vá theo thứ tự ưu tiên, sau khi các lỗ hổng nghiêm trọng hơn đã được khắc phục.
<b>MINOR</b>	Những vấn đề minor thường liên quan đến bản chất lập trình (ngôn ngữ, các quy tắc lập trình, v.v.) và thường không thực sự là lỗi hay lỗ hổng bảo mật. Những vấn đề này vẫn được khuyến nghị khắc phục, trừ khi có những lý do chính đáng để không tiến hành xử lý.
<b>INFO</b>	Những vấn đề chỉ mang tính chất thông tin (info) thường liên quan đến những phương pháp lập trình tối ưu hoặc tính dễ đọc của mã nguồn. Thông thường, những vấn đề này không phải là lỗi hay lỗ hổng bảo mật. Nên khắc phục những vấn đề này, nếu chúng thực sự nâng cao tính hiệu quả cũng như bảo mật cho hợp đồng thông minh.

## Phụ lục C – Phân loại lỗ hổng theo Smart Contract Weakness Classification Registry (SWC Registry)

Mã định danh	Tên	Mô tả
	<b>Các vấn đề liên quan đến quy chuẩn lập trình</b>	
SWC-100	Function Default Visibility	Nên có lựa chọn chính xác cho giới hạn truy cập đối với một hàm, có thể là <i>public</i> , <i>external</i> , <i>internal</i> hoặc <i>private</i> . Mặc định thì hàm cho phép truy cập <i>public</i> .
SWC-102	Outdated Compiler Version	Nên sử dụng phiên bản trình biên dịch Solidity gần đây nhất để tránh các lỗi hay lỗ hổng gặp phải ở các phiên bản cũ.
SWC-103	Floating Pragma	Nên khóa pragma để đảm bảo các hợp đồng thông minh không triển khai trên các phiên bản Solidity cũ mang lỗ hổng.
SWC-108	State Variable Default Visibility	Các biến có thể truy cập ở dạng <i>public</i> , <i>internal</i> hoặc <i>private</i> . Cần định nghĩa định dạng truy cập cho các biến hợp lý.
SWC-111	Use of Deprecated Solidity Functions	Solidity cung cấp các giải pháp khác thay vì sử dụng các hàm constructor không còn được hỗ trợ. Các phương án đều tương đồng, và việc thay thế cấu trúc cũ không làm thay đổi hành vi hợp đồng.
SWC-118	Incorrect Constructor Name	Nên sử dụng trình biên dịch Solidity gần nhất cho contract và chuyển sang định nghĩa constructor mới (với từ khóa <i>constructor</i> ).
	<b>Các vấn đề ở thiết kế tổng quan</b>	

SWC-113	DoS with Failed Call	Các external call có thể không thực hiện được do khách quan hoặc chủ quan, điều này gây ra tình trạng DoS trong hợp đồng thông minh. Nên tách biệt các external call ở các giao dịch riêng biệt và thiết kế các hàm xử lý khi call không thể thực hiện.
SWC-119	Shadowing State Variables	Kiểm tra các biến được lưu trữ cho hệ thống hợp đồng thông minh và loại bỏ tối đa những yếu tố không minh bạch. Kiểm tra các cảnh báo từ trình biên dịch về việc sử dụng biến trong một hợp đồng.
SWC-125	Incorrect Inheritance Order	Khi kế thừa nhiều hợp đồng, nhất là khi chúng có các hàm như nhau, lập trình viên nên thận trọng ghi đúng thứ tự thừa kế (từ những cái chung nhất đến cái cụ thể hơn).
SWC-128	DoS With Block Gas Limit	Thay đổi mảng với kích cỡ không xác định, có thể khiến cho kích thước của mảng tăng theo thời gian, dẫn tới tình trạng Từ chối dịch vụ. Nên tránh các xử lý đòi hỏi việc looping xuyên suốt một cấu trúc dữ liệu.
	<b>Các vấn đề liên quan đến lập trình an toàn</b>	
SWC-101	Integer Overflow and Underflow	Nên sử dụng các thư viện safe math cho các phép tính toán học xuyên suốt toàn bộ hệ thống hợp đồng thông minh, nhằm tránh integer overflow hoặc underflow.
SWC-107	Reentrancy	Đảm bảo các sự thay đổi các trạng thái ẩn bên trong được thực hiện trước khi tiến hành một call, hoặc sử dụng khóa reentrancy.
SWC-112	Delegatecall to Untrusted Callee	Sử dụng <i>delegatecall</i> thận trọng và đảm bảo không bao giờ gọi tới các hợp đồng không đáng tin cậy. Nếu địa chỉ gọi đến là dữ liệu nhập vào từ người dùng, đảm bảo dữ liệu này được kiểm tra thông qua một whitelist các hợp đồng đáng tin cậy.

SWC-117	Signature Malleability	Một chữ ký không nên được đính kèm vào hash của tin nhắn đã được ký nếu các tin nhắn được xử lý bởi các hợp đồng thông minh.
SWC-121	Missing Protection against Signature Replay Attacks	Để bảo vệ khỏi các cuộc tấn công dùng lại chữ ký, cần lưu trữ các hash tin nhắn được xử lý bởi hợp đồng thông minh, trong đó bao gồm cả địa chỉ của hợp đồng thông minh đã thực hiện xử lý tin nhắn, và không bao giờ tạo hash một tin nhắn chứa cả chữ ký.
SWC-122	Lack of Proper Signature Verification	Khuyến nghị không sử dụng các mô hình kiểm tra xác minh khác mà không đòi hỏi xác thực chữ ký thông qua <code>ecrecover()</code> .
SWC-130	Right-To-Left-Override control character (U+202E)	Ký tự <code>U+202E</code> không nên xuất hiện trong mã nguồn của hợp đồng thông minh.
	<b>Các vấn đề về thiết kế lập trình</b>	
SWC-104	Unchecked Call Return Value	Nếu sử dụng các method low-level call, cần kiểm soát khả năng các call không thực hiện được bằng cách kiểm tra giá trị trả về.
SWC-105	Unprotected Ether Withdrawal	Thiết đặt các cơ chế kiểm soát sao cho các giao dịch rút tiền chỉ có thể được thực hiện bởi các bên được ủy quyền hoặc tuân thủ chặt chẽ theo tài liệu mô tả kỹ thuật của hệ thống hợp đồng thông minh.
SWC-106	Unprotected SELFDESTRUCT Instruction	Cân nhắc việc loại bỏ tính năng self-destruct. Nếu thực sự cần thiết, nên thiết đặt một mô hình đa chữ ký (multisig) để đảm bảo thao tác self-destruct chỉ được thực hiện khi có sự chấp nhận của nhiều bên.
SWC-110	Assert Violation	Cân nhắc xem các điều kiện được kiểm tra trong <code>assert()</code> có thực sự bất biến hay không. Nếu không, thay thế <code>assert()</code> bằng <code>require()</code> .

SWC-116	Block values as a proxy for time	Cần lưu ý khi sử dụng các giá trị khối (block values) khi viết các hợp đồng thông minh, do việc sử dụng chúng có thể dẫn tới các kết quả không tính toán được. Hoặc sử dụng các oracle.
SWC-120	Weak Sources of Randomness from Chain Attributes	Nhằm tránh các việc sử dụng yếu tố ngẫu nhiên không đủ mạnh, có thể sử dụng các mô hình có sẵn như RANDAO, hoặc các mô hình ngẫu nhiên bên ngoài, thông qua oracle như Oraclize, hay các hash của khối Bitcoin block.
SWC-123	Requirement Violation	Nếu đòi hỏi một yêu cầu logic quá chặt chẽ, cần giảm bớt điều này để cố thể chấp nhận các giá trị nhập vào hợp lệ từ bên ngoài. Nếu không, cần đảm bảo không có giá trị nhập vào không hợp lệ được nhập vào được cung cấp.
SWC-124	Write to Arbitrary Storage Location	Tất cả các cấu trúc dữ liệu nên sử dụng chung một vùng nhớ (địa chỉ), và cần đảm bảo việc ghi lại dữ liệu cho một cấu trúc dữ liệu không ghi đè lên cấu trúc dữ liệu khác.
SWC-132	Unexpected Ether balance	Hạn chế kiểm tra bằng điều kiện "bằng" quá chặt đối với số dư Ether có trong hợp đồng.
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Khi sử dụng <i>abi.encodePacked()</i> , cần lưu ý kiểm tra chữ ký hợp lệ không thể tạo ra từ nhiều các tham số khác nhau. Để đơn giản, có thể sử dụng <i>abi.encode()</i> thay thế. Hàm này cũng được khuyên dùng để tránh các cuộc tấn công sử dụng lại.
	<b>Các nguy cơ tiềm ẩn khác trong mã nguồn</b>	
SWC-109	Uninitialized Storage Pointer	Các biến được lưu local mà chưa được khởi tạo có thể trở và các vùng nhớ không dự đoán được trong hợp đồng. Nếu việc sử dụng biến local là cần thiết, đánh dấu nó với từ khóa <i>memory</i> , hoặc <i>storage</i> phụ thuộc vào định nghĩa. Trình biên dịch từ các phiên bản 0.5.0 hoặc mới hơn thì vấn đề này đã được giải quyết trên toàn hệ thống.

SWC-114	Transaction Order Dependence	Để giải quyết tình trạng race condition khi đưa lên các thông tin để trao đổi hỏi thưởng có thể sử dụng mô hình hiện hữu hash commit. Để giải quyết được race condition cho ERC20 cần thêm các cơ chế approve an toàn để giao dịch có thể thực hiện hoặc hoàn trả (revert).
SWC-115	Authorization through tx.origin	<i>tx.origin</i> không nên được sử dụng để xác định phân quyền hay ủy quyền. Nên sử dụng <i>msg.sender</i> thay thế.
SWC-126	Insufficient Gas Griefing	Tấn công insufficient gas griefing có thể được thực hiện với các hợp đồng nhận dữ liệu và sử dụng dữ liệu này trong một sub-call tới hợp đồng khác. Để tránh kiểu tấn công này, chỉ cho phép những người dùng đáng tin cậy được thực hiện giao dịch có trung gian và đòi hỏi phía trung gian (forwarder) phải cung cấp đủ phí gas.
SWC-127	Arbitrary Jump with Function Type Variable	Việc sử dụng assembly nên được hạn chế tối đa. Lập trình viên không nên để người dùng nhập giá trị bất kỳ vào các biến kiểu hàm.
SWC-129	Typographical Error	Khiếm khuyết này có thể hạn chế bằng cách thực hiện các kiểm tra điều kiện tiên quyết trước các phép tính toán học, hoặc sử dụng thư viện có sẵn cho các tính toán số học như SafeMath phát triển bởi OpenZeppelin.
SWC-131	Presence of unused variables	Loại bỏ toàn bộ các biến không được sử dụng khỏi mã nguồn.
SWC-134	Message call with hardcoded gas amount	Hạn chế sử dụng <i>transfer()</i> và <i>send()</i> và cũng không nên cố định giá trị phí gas cho việc thực hiện các hàm này. Sử dụng <i>.call.value(...)(<i>"</i>)</i> thay thế.
SWC-135	Code With No Effects	Cần đảm bảo hợp đồng thông minh vận hành tuân thủ tài liệu mô tả kỹ thuật. Viết các test unit để đảm bảo mã nguồn vận hành chính xác.
SWC-136	Unencrypted Private Data On-Chain	Tất cả những dữ liệu mật không nên được lưu trữ off-chain, hoặc cần được mã hóa cẩn thận.



## Phụ lục D – Phân loại lỗ hổng theo Related Common Weakness Enumeration (CWE)

Phân loại SWC Registry dựa trên các định nghĩa và có cấu trúc tương đồng với phân loại CWE, nhưng đồng thời cũng bao quát toàn diện hơn các lỗ hổng đặc biệt liên quan tới các hợp đồng thông minh.

Những lỗ hổng CWE \* mà những lỗ hổng trong danh mục SWC Registry có liên quan, được liệt kê ở bảng dưới đây:

Mã CWE	Tên	Các mã SWC tương ứng
<b>CWE-284</b>	<b>Improper Access Control</b>	SWC-105, SWC-106
CWE-294	Authentication Bypass by Capture-replay	SWC-133
<b>CWE-664</b>	<b>Improper Control of a Resource Through its Lifetime</b>	SWC-103
CWE-123	Write-what-where Condition	SWC-124
CWE-400	Uncontrolled Resource Consumption	SWC-128
CWE-451	User Interface (UI) Misrepresentation of Critical Information	SWC-130
CWE-665	Improper Initialization	SWC-118, SWC-134
CWE-767	Access to Critical Private Variable via Public Method	SWC-136
CWE-824	Access of Uninitialized Pointer	SWC-109
CWE-829	Inclusion of Functionality from Untrusted Control Sphere	SWC-112, SWC-116
<b>CWE-682</b>	<b>Incorrect Calculation</b>	SWC-101
<b>CWE-691</b>	<b>Insufficient Control Flow Management</b>	SWC-126
CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ("Race Condition")	SWC-114
CWE-480	Use of Incorrect Operator	SWC-129
CWE-667	Improper Locking	SWC-132
CWE-670	Always-Incorrect Control Flow Implementation	SWC-110
CWE-696	Incorrect Behavior Order	SWC-125

CWE-841	Improper Enforcement of Behavioral Workflow	SWC-107
<b>CWE-693</b>	<b>Protection Mechanism Failure</b>	
CWE-330	Use of Insufficiently Random Values	SWC-120
CWE-345	Insufficient Verification of Data Authenticity	SWC-122
CWE-347	Improper Verification of Cryptographic Signature	SWC-117, SWC-121
<b>CWE-703</b>	<b>Improper Check or Handling of Exceptional Conditions</b>	SWC-113
CWE-252	Unchecked Return Value	SWC-104
<b>CWE-710</b>	<b>Improper Adherence to Coding Standards</b>	SWC-100, SWC-108, SWC-119
CWE-477	Use of Obsolete Function	SWC-111, SWC-115
CWE-573	Improper Following of Specification by Caller	SWC-123
CWE-695	Use of Low-Level Functionality	SWC-127
CWE-1164	Irrelevant Code	SWC-131, SWC-135
<b>CWE-937</b>	<b>Using Components with Known Vulnerabilities</b>	SWC-102

\* Những mã CVE in đậm là danh mục cha của các mã CWE theo sau nó.

Tất cả các mã lỗi hổng trong danh mục CWE trên đây đều nằm trong danh mục “Research Concepts” (CWE-1000), ngoại trừ mã CWE-937, liên quan đến phân loại ở danh mục “Weaknesses in OWASP Top Ten (2013)” (CWE-928).