**Vizva Smart Contracts Initial Report**

[**Scope of Audit**](#_ypywdf1ix69g)2

[**Check Vulnerabilities**](#_fk11hakqdckb)2

[**Techniques and Methods**](#_4f674zh1p4ou)3

[Issue Categories](#_6qxggtor8n0m) 4

[Number of security issues per severity.](#_2idbboidjlj7) 5

[**Introduction**](#_aqhg5kqr9l1u)6

[**Issues Found – Code Review / Manual Testing**](#_jnyy7e310pds) **7**

[High Severity Issues](#_tif7pylvu8h3) 7

[H.1 Cannot verify signature](#_rqevufs9vhax) 7

[Low Severity Issues](#_svf7kq8t0gvg) 8

[L.1 Unused Imports](#_xg8p62iavmvd) 8

[L.2 Unused ReentrancyGuard](#_unuvo0w5fbua) 9

[L.3 Misleading error message](#_svhlzd9kf3kz) 9

[L.4 Unnecessary require check](#_heo6fy626leu) 9

[L.5 Unnecessary return variable declaration](#_1n6kn1pxbhkz) 10

[L.6 Missing Events](#_o5l5wj5agvv9) 11

[L.7 Incorrect Event emit](#_2pcf6vl93qb1) 11

[Informational Issues](#_zays838lwmyq) 11

[I.1 Constants can be initialized with hardcoded values 1](#_nhb4vqaltlj9)1

[I.2 Repeated read from state](#_unuvo0w5fbua) 12

[I.3 Similar code between two methods](#_vkcliqxbe0rm) 12

[I.4 Contract name in error message](#_ia6jm4dnennx) 13

[I.5 uint256 instead of uint](#_6civuqcqtm6l) 13

[I.6 Missing netspec comments](#_1enr6l318a00) 13

[I.7 Event naming convention](#_x7u68gwznmvz) 13

[I.8 Refactor buyItem method](#_8y374hc7ev5e) 14

[I.9 Virtual methods](#_5x4ojyo4enc2) 14

**Automated Tests 16**

[**Results**](#_nnxarl1y1et8) **24**

[**Closing Summary**](#_lfygtoo97ebl) **24**

## **Scope of Audit**

The scope of this audit was to analyze and document the Vizva smart contracts codebase for quality, security, and correctness.

## **Check Vulnerabilities**

* Re-entrancy
* Timestamp Dependence
* Gas Limit and Loops
* DoS with Block Gas Limit
* Transaction-Ordering Dependence
* Use of tx.origin
* Exception disorder
* Gasless send
* Balance equality
* Byte array
* Transfer forwards all gas
* ERC20 API violation
* Malicious libraries
* Compiler version not fixed
* Redundant fallback function
* Send instead of transfer
* Style guide violation
* Unchecked external call
* Unchecked math
* Unsafe type inference
* Implicit visibility level

# **Techniques and Methods**

Throughout the audit of smart contracts, care was taken to ensure:

* The overall quality of code.
* Use of best practices.
* Code documentation and comments match logic and expected behavior.
* Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
* Efficient use of gas.
* Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

**Structural Analysis**

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

**Static Analysis**

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

**Code Review / Manual Analysis**

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

**Gas Consumption**

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

**Tools and Platforms used for Audit**

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

## 

## 

## **Issue Categories**

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

**High Severity Issues**

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract’s performance or functionality, and we recommend these issues be fixed before moving to a live environment.

**Medium Severity Issues**

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

**Low Severity Issues**

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

**Informational** **Issues**

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

## 

## **Number of security issues per severity.**

# 

| **TYPE** | **HIGH** | **MEDIUM** | **LOW** | **INFORMATIONAL** |
| --- | --- | --- | --- | --- |
| **Open** | **1** | **0** | **7** | **9** |
| **Acknowledged** | **0** | **0** | **0** | **0** |
| **Closed** | **0** | **0** | **0** | **0** |

# 

# **Introduction**

During the period of **March 8th, 2021 to March 20th, 2021**.

Manmeet performed a security audit for **Vizva** smart contracts.

## 

| **Version Number** | **Date** |
| --- | --- |
| 01 | Mar 8 - Mar 13 |

# **Issues Found – Code Review / Manual Testing**

## **High Severity Issues**

### H.1 Cannot verify signature

**Contract**: VizvaMarket\_V1.sol

**Description:**

The verification of voucher did not return the right address for any signature that was created during the testing

| 799 | function \_verifyBid(BidVoucher calldata voucher)  internal  view  virtual  returns (address)  {  bytes32 digest = \_hash(  abi.encode(  keccak256(  "BidVoucher(address asset,address tokenAddress,uint256 tokenId,uint256 marketId,uint256 bid)"  ),  voucher.asset,  voucher.tokenAddress,  voucher.tokenId,  voucher.marketId,  voucher.bid  )  );  return ECDSAUpgradeable.recover(digest, voucher.signature);  } |
| --- | --- |
| 823 | function \_verifyNFTVoucher(NFTVoucher calldata voucher)  internal  view  returns (address)  {  bytes32 digest = \_hash(  abi.encode(  keccak256(  "NFTVoucher(uint256 tokenId,uint256 minPrice,uint16 royalty,string uri,address tokenAddress)"  ),  voucher.tokenId,  voucher.minPrice,  voucher.royalty,  keccak256(bytes(voucher.uri)),  voucher.tokenAddress  )  );  return ECDSAUpgradeable.recover(digest, voucher.signature);  } |

**Recommendation**:

Test and modify the function or provide documentation on how to generate a valid signature.

**Status: Open**

## **Low Severity Issues**

### L.1 Unused Imports

**Contract**: Bool\_Token.sol, Vizva721.sol, VizvaMarket\_V1.sol and VizvaMarketPlace.sol

**Description**

The contracts *Bool\_Token.sol, Vizva721.sol and VizvaMarket\_V1.sol* import openzeppelins *Initializable.sol* to make the contract upgradeable but the contracts are already initializable as the import other Upgradeable contracts which higher up in the hierarchy imports *Initializable.sol.*

The contract VizvaMarketContract imports the ReentrancyGuard and does not use it

**Recommendation**:

We should remove the unnecessary imports.

**Status: Open**

### 

### L.2 Unused ReentrancyGuard

**Contract**: VizvaLazyNFT\_V1.sol

**Description**

The contracts imports and uses ReentrancyGuard in the definition but never uses the nonReentrant modifier in the contained code*.*

**Recommendation**:

We should remove the unnecessary import.

**Status: Open**

### L.3 Misleading error messsage

**Contract**: Vizva\_MarketPlace.sol

**Description**

The *OnlyItemOwner* modifier in the contract message states this as the require check message “*Only Item owner alowed to list in market*” which is specific to a use case and can be misleading if modifier is used somewhere else

**Recommendation**:

Use a generic error message specific to the modifier e.g. “NOT\_OWNER” or “only owner can call this function”

**Status: Open**

### L.4 Unnecessary require check

**Contract**: Vizva\_MarketPlace.sol

**Description**

The method *addItemToMarket* has this require check within the code  
*require(itemsForSale[newItemId].id == newItemId, "Item id mismatch")*

But this should always be true because it was immediate assigned in the function.

**Recommendation**:

Remove the require check as it will always be true but consume gas in execution

**Status: Open**

### L.5 Unnecessary return variable declaration

**Contract**: VizvaMarketPlace.sol

**Description**

There are methods in this contract which declare a return variable and don't use it to return values. The values are returned explicitly.

| **Line** | **Code/Function** |
| --- | --- |
| 255 | function getAllItemForSale()  public  view  returns (SaleOrder[] memory saleOrder) |
| 310 | function updateCommission(uint16 \_newValue)  public  virtual  onlyOwner  returns (uint16 \_commission) |
| 327 | function updateWalletAddress(address \_address)  public  virtual  onlyOwner  returns (address \_wallet) |

**Recommendation**:

Remove the declared return variables as they are unused

**Status: Open**

### L.6 Missing events

**Contract**: VizvaMarketPlace.sol

**Description**

The methods *updateCommission* and *updateWalletAddress* do not emit any event on update

**Recommendation**:

Add an update event for both methods and return bool value after emitting the update event

**Status: Open**

### L.7 Incorrect event emit

**Contract**: VizvaMarketPlace.sol

**Description**

The method *batchCancelSale* emits *batchSaleCancelled* event from withing the for loop which is incorrect

**Recommendation**:

The event *batchSaleCancelled*  should be emitted only once after the batch operation

**Status: Open**

## **Informational Issues**

### I.1 Constant initialization can be changed to hardcoded values

**Contract:** BoolToken.sol

| **Line** | **Code/Function** |
| --- | --- |
| 56 | bytes32 public constant MINTER\_ROLE = keccak256("MINTER\_ROLE");  bytes32 public constant BURNER\_ROLE = keccak256("BURNER\_ROLE");  bytes32 public constant MANAGER\_ROLE =keccak256("MANAGER\_ROLE"); |

**Description:**

The MINTER\_ROLE, BURNER\_ROLE and MANAGER\_ROLE values are calculated during deployment which increases the deployment cost

**Recommendation:**

Since the calculation results will always be the same we precalculate the byte32 value and directly assign to them to save cost during deployment and for readability purpose we can use comments e.g.

*// keccak256("MINTER\_ROLE")*

*bytes32 public constant MINTER\_ROLE = 9f2df0fed2c77648de5860a4cc508cd0818c85b8b8a1ab4ceeef8d981c8956a6;*

**Status: Open**

### I.2 Repeated read from state

**Contract:** Vizva\_MarketPlace.sol

**Description:**

The contract within the methods *finalizeBid* and *buyItem* reads same values like *tokenAddress* and *tokenId* from the state variable itemsForSale several times, which is a costly operation

**Recommendation:**

Store values that are required several times in a local variable and reuse it from there instead of reading from state everytime

**Status: Open**

### I.3 Similar code between two methods Contract: Vizva\_MarketPlace.sol

**Description:** The methods *finalizeBid* and *buyItem* share similar code while executing the sale

**Recommendation:**

We recommend making an internal method called *\_executeSale* and make a call two it from both methods to avoid writing repeated code

**Status: Open**

### I.4 Contract name in error message

**Recommendation:**

It is recommended to add contract name in the error message, since it makes it easier to traceback when the error is through e.g.

*require(success, "Transfer failed.")*

should be written as

*require(success, "VizvaMarketContract: Transfer failed.");*

**Status: Open**

### I.5 uint256 instead of uint

**Recommendation:**

When we use *uint* to initialize a variable in solidity it defaults to *uint256,* therefore it is recommended to use *uint256* itself to initialize the variable for readability.

**Status: Open**

### I.6 Missing netspec comments

**Recommendation:**

We recommend adding netspec comments for each method and variables for better readability and understanding of code.

**Status: Open**

### I.7 Event naming convention

**Description**

The event names throughout the contract e.g. *itemAdded, saleCancelled, batchSaleCancelled, salePriceUpdated* are named in camel-case, which is not a standard practice for naming events

**Recommendation**

The event names should start with a capital letter.

**Status: Open**

### I.8 When buying item we can just take ID as parameter

**Recommendation:**

When buying item we can just take the sale Id as a parameter and fetch token address and token id from the mapping,

It be less complex and will remove a couple of more require checks to save gas cost

**Status: Open**

### I.9 Virtual methods

**Description**

Almost all method in *Bool\_Token\_V1* and *VizvaMarket\_V1* are marked as virtual, which allows them to be overridden but they are never overridden

**Recommendation**

Remove unnecessary declaration of methods as virtual

**Status: Open**

# **Goerli Test Contracts**

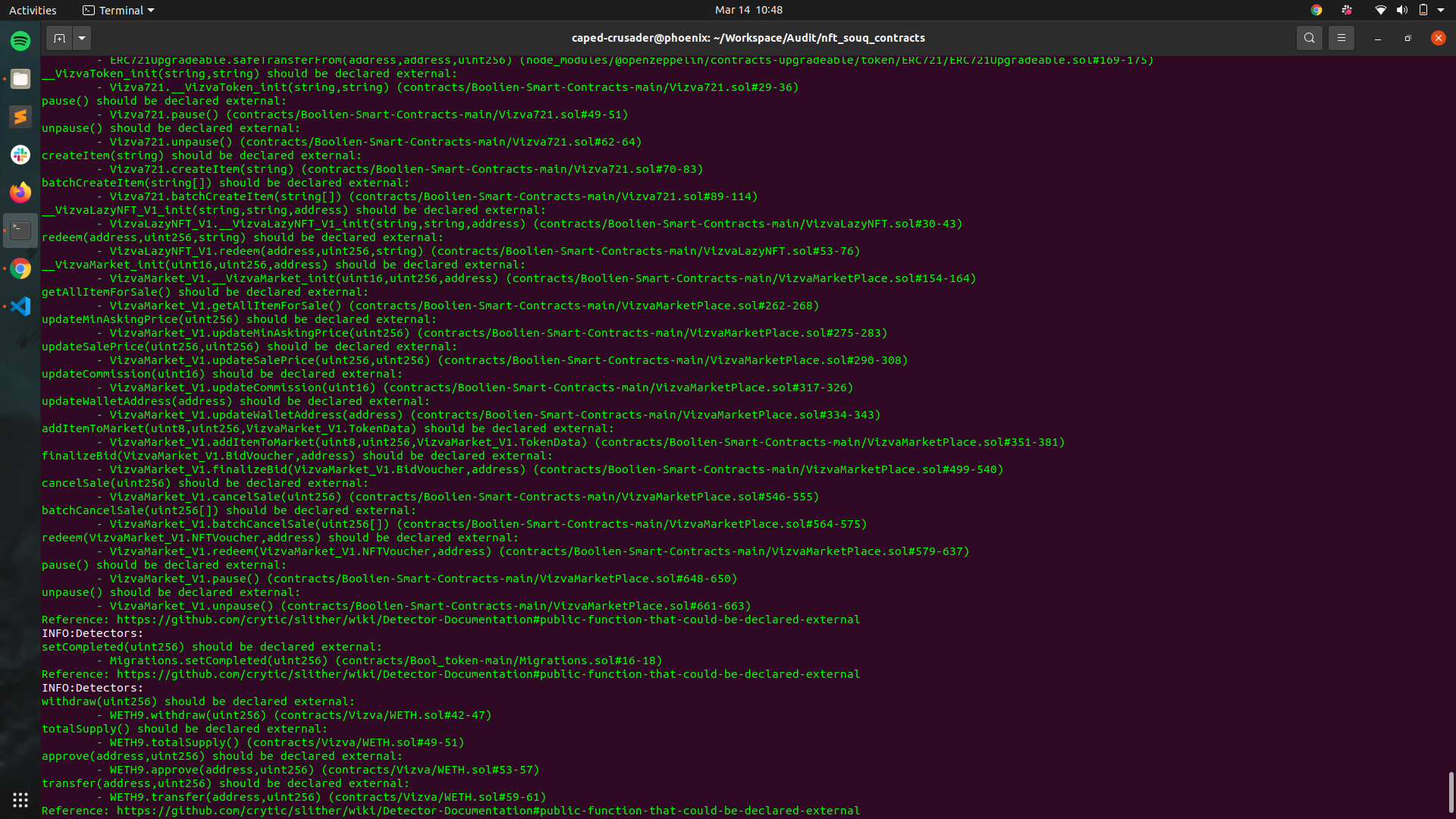
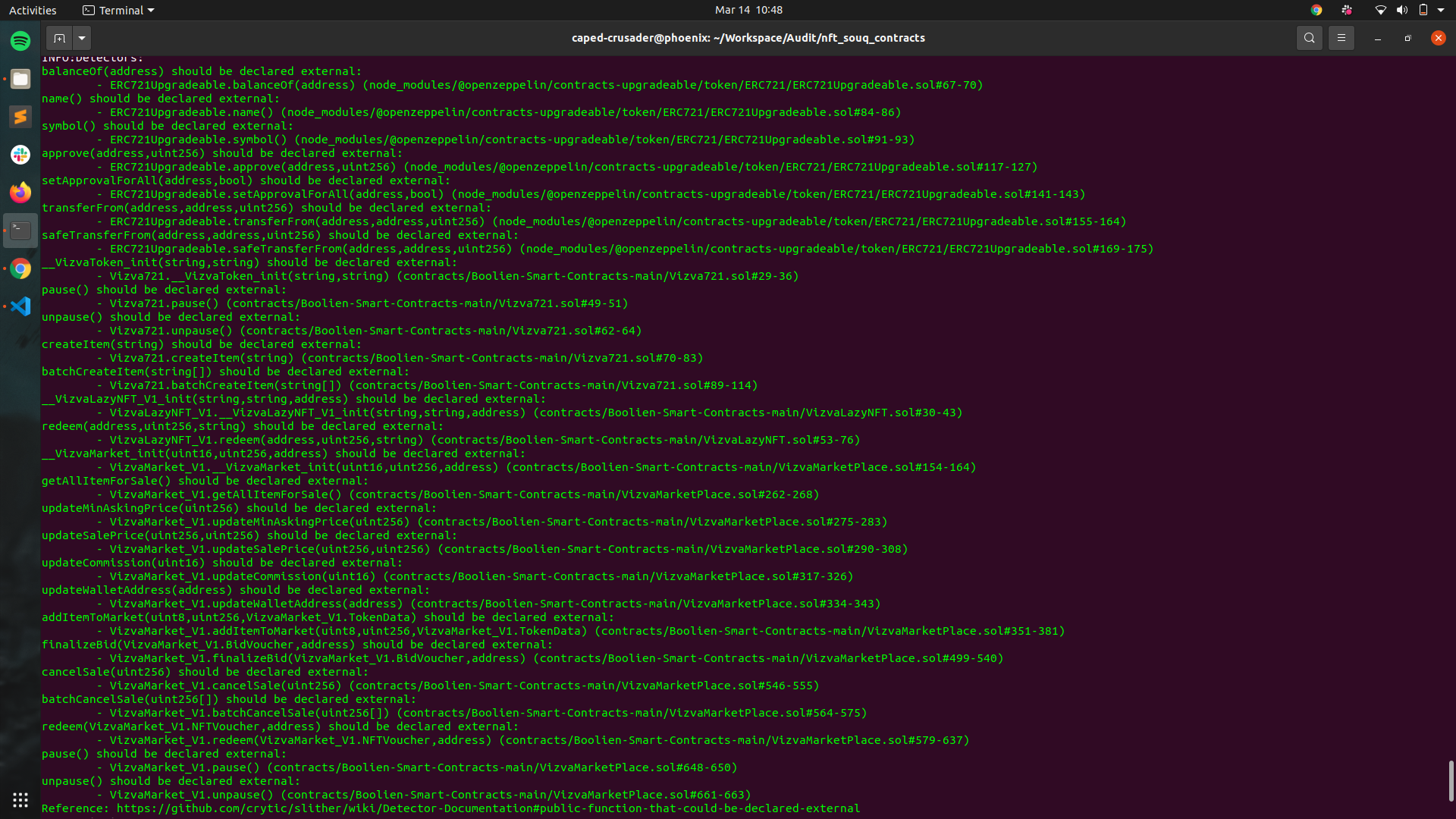
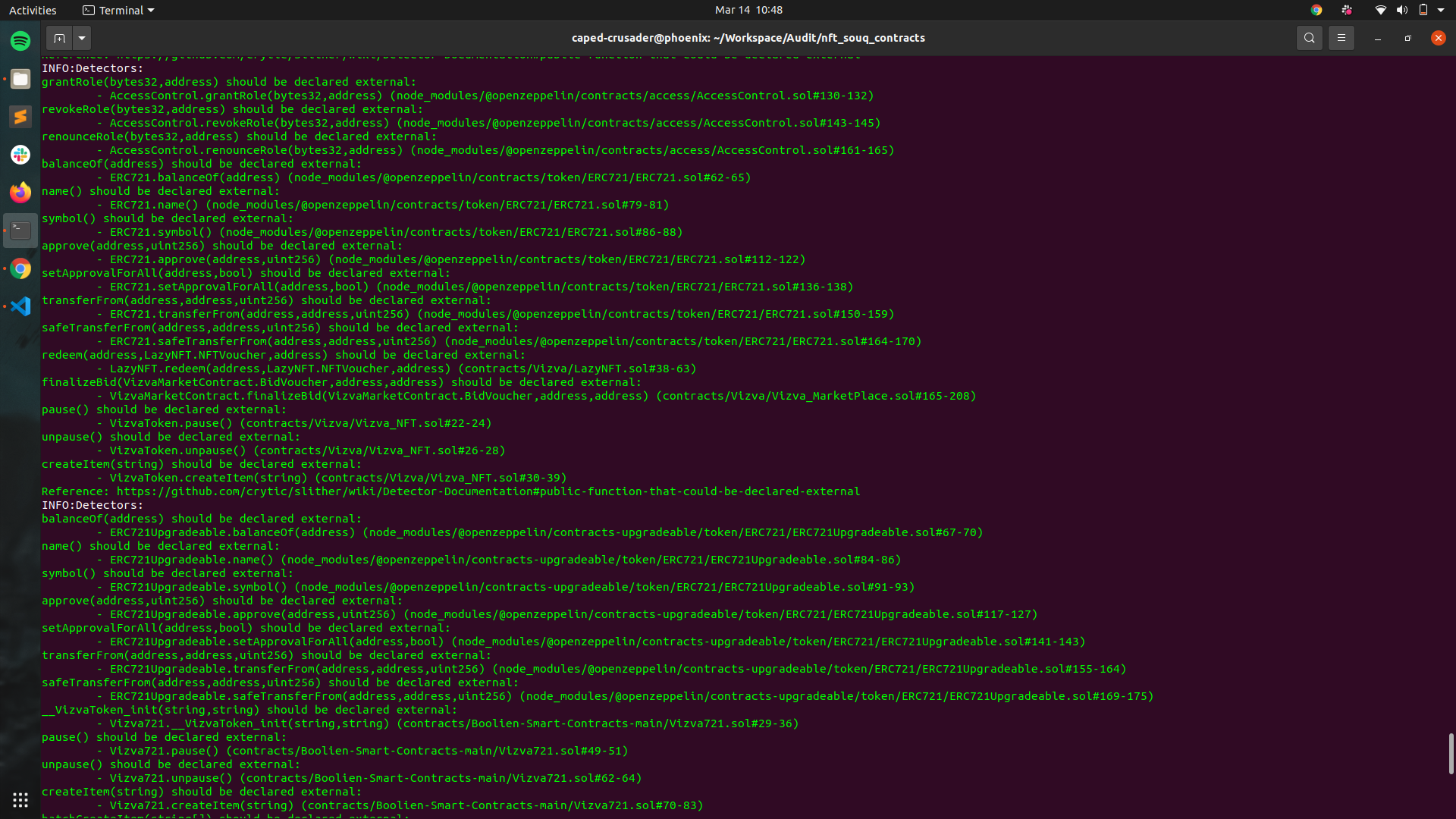
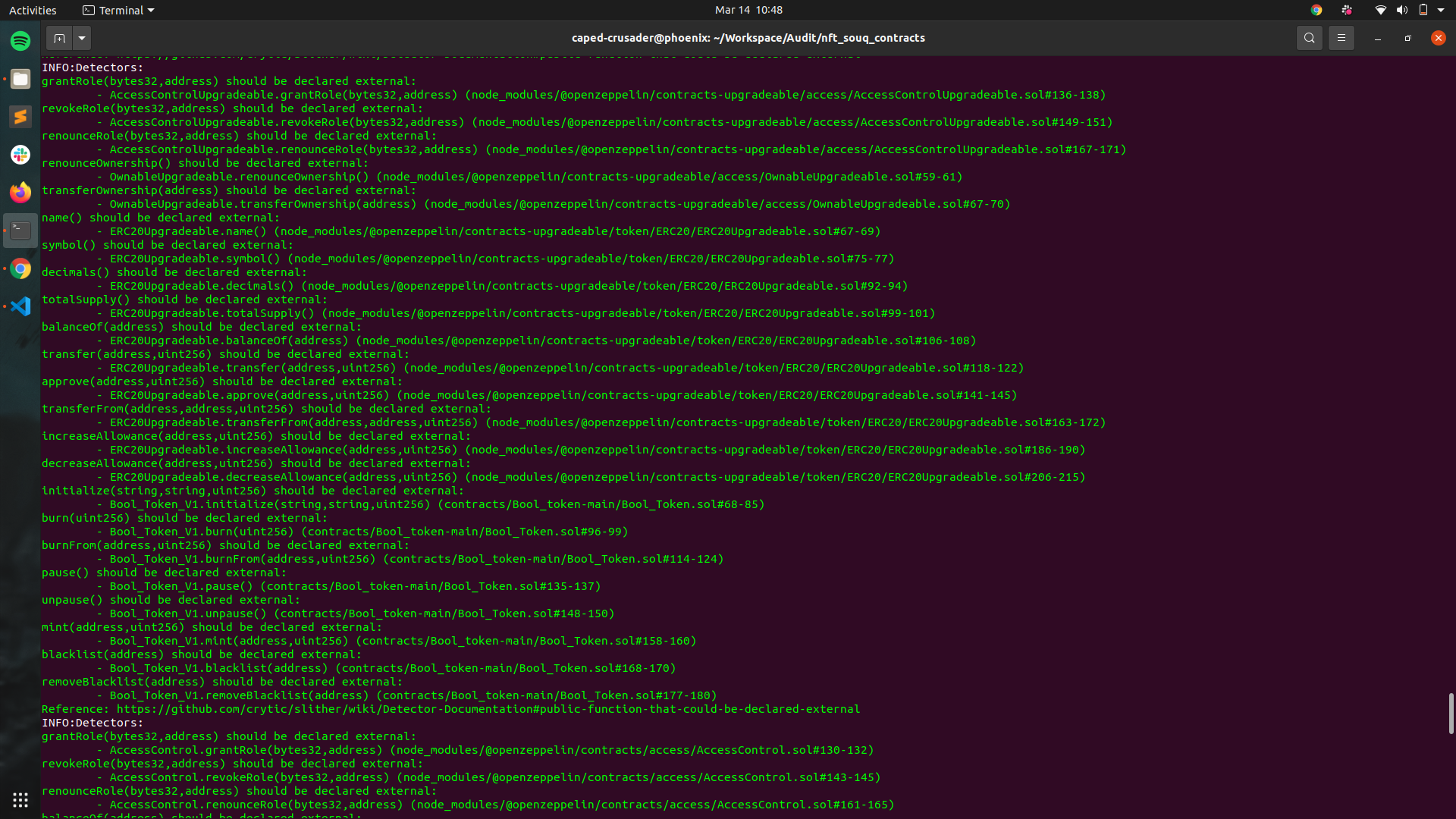
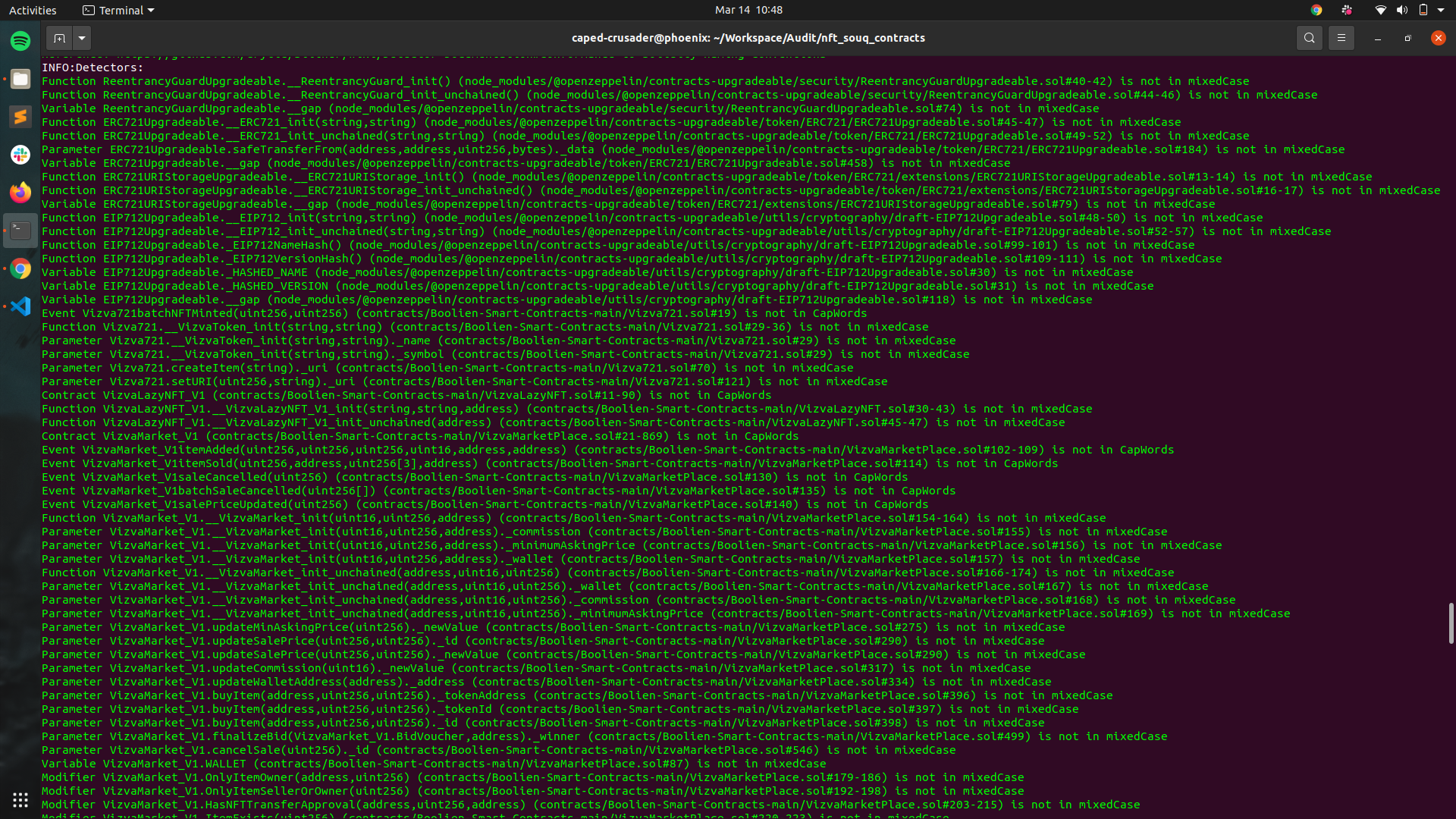
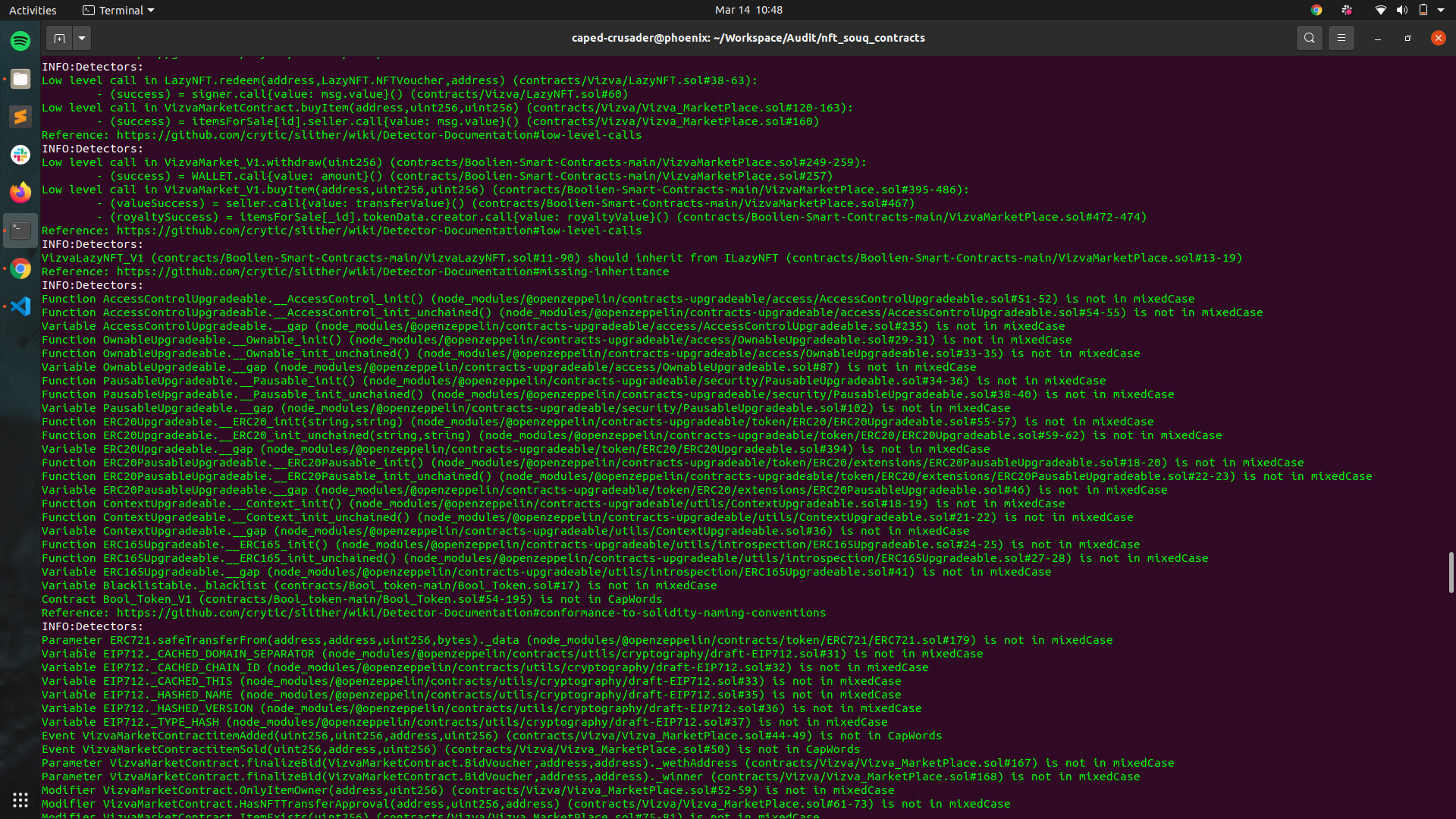
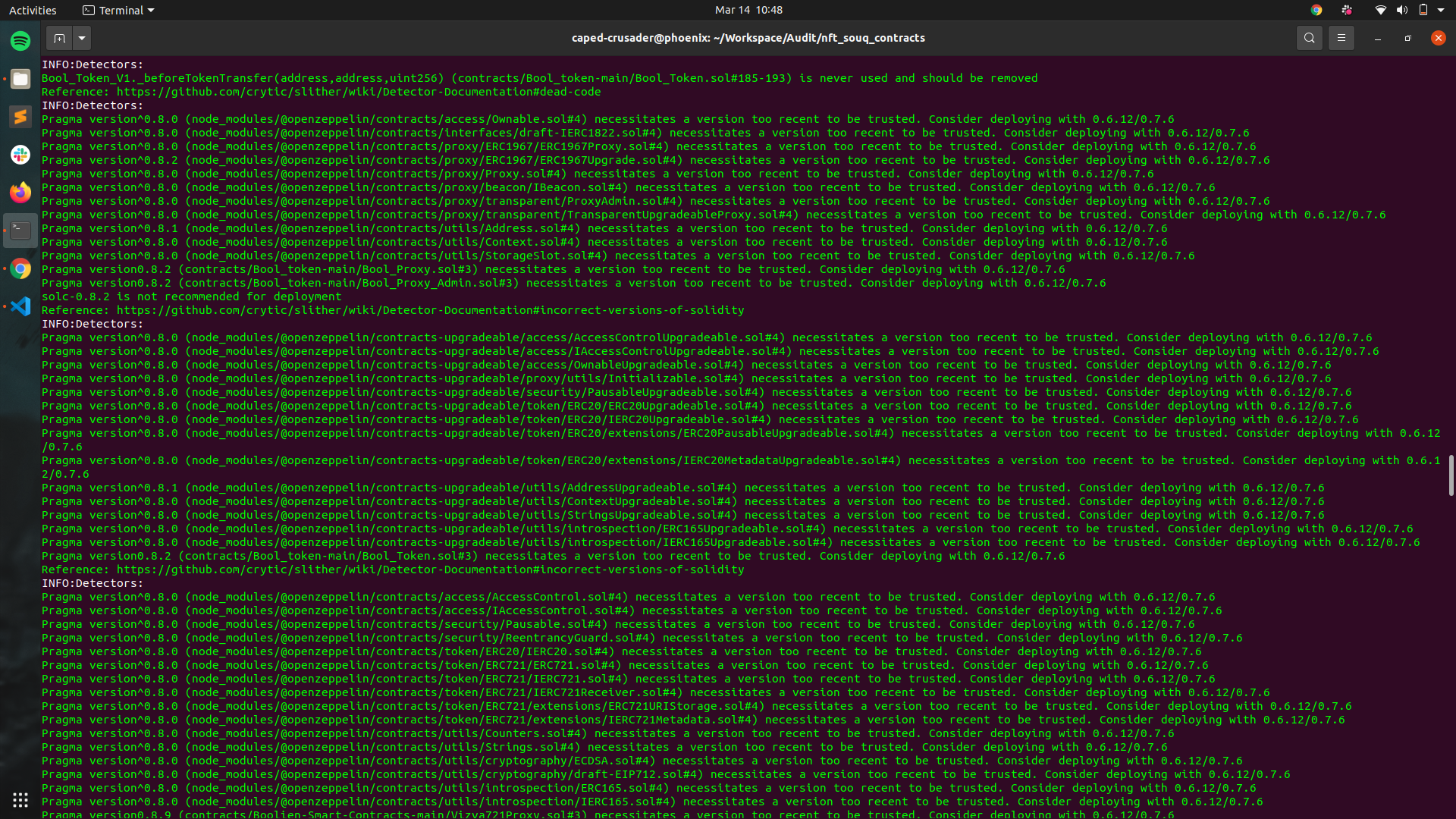
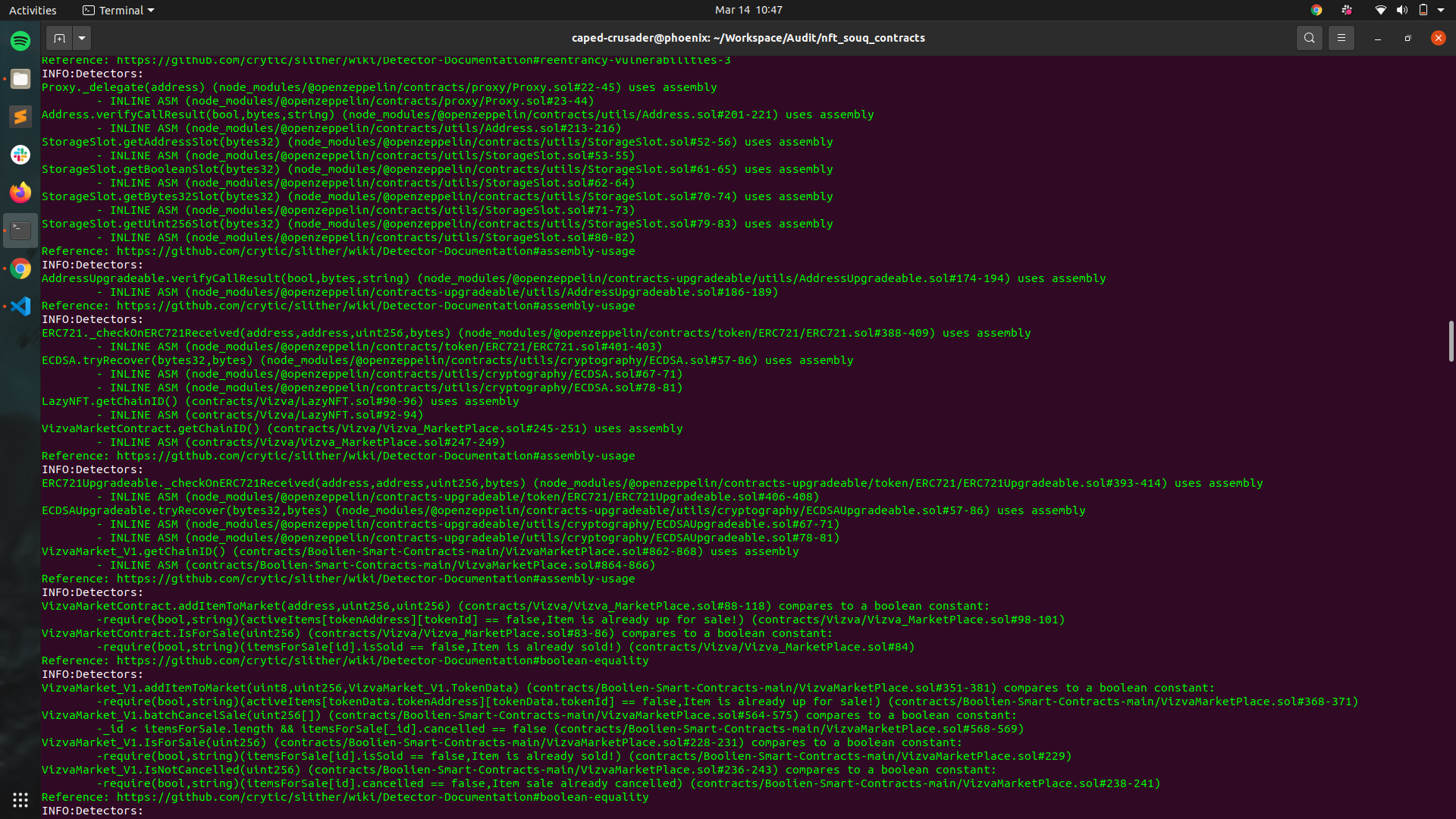
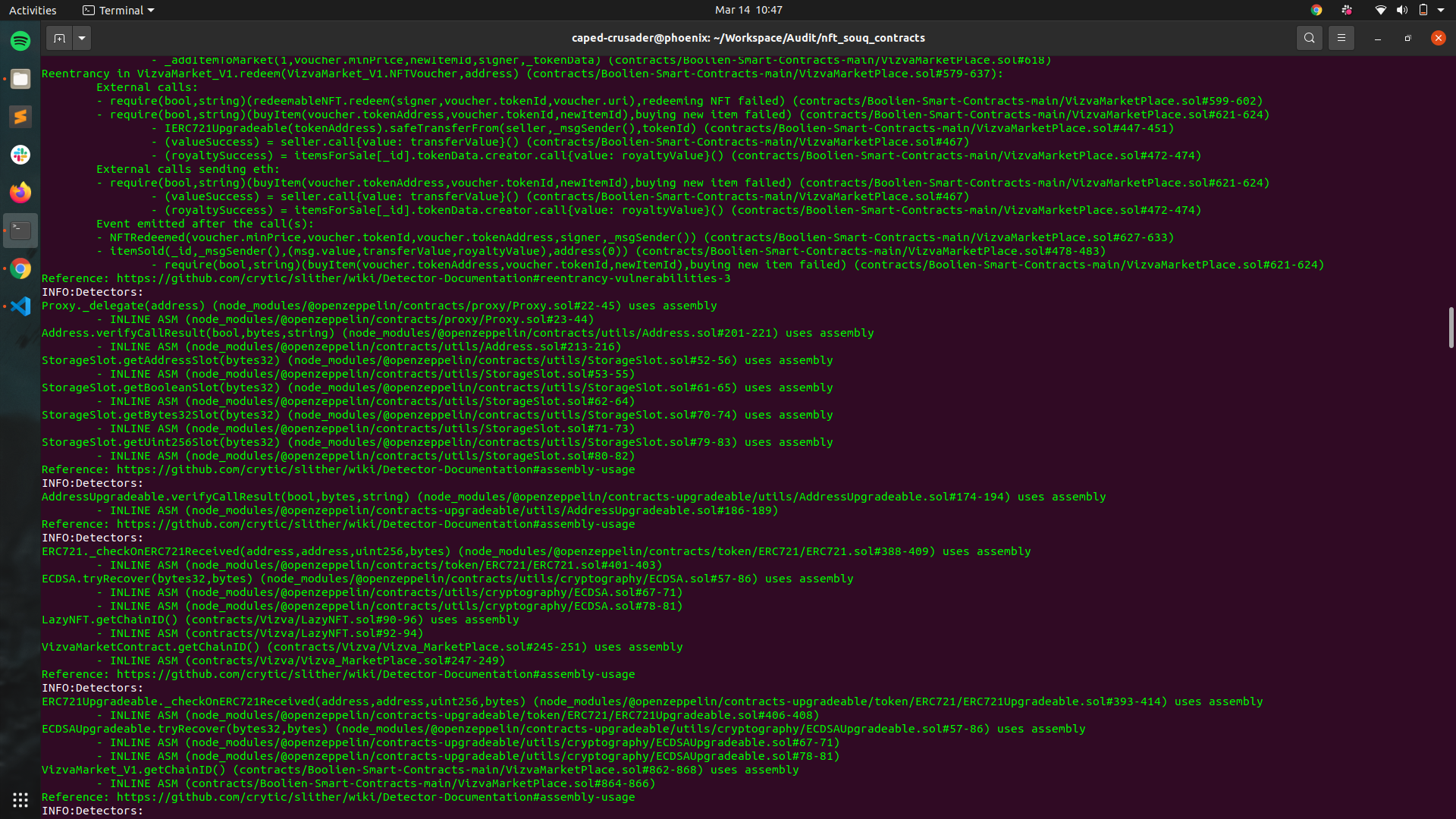
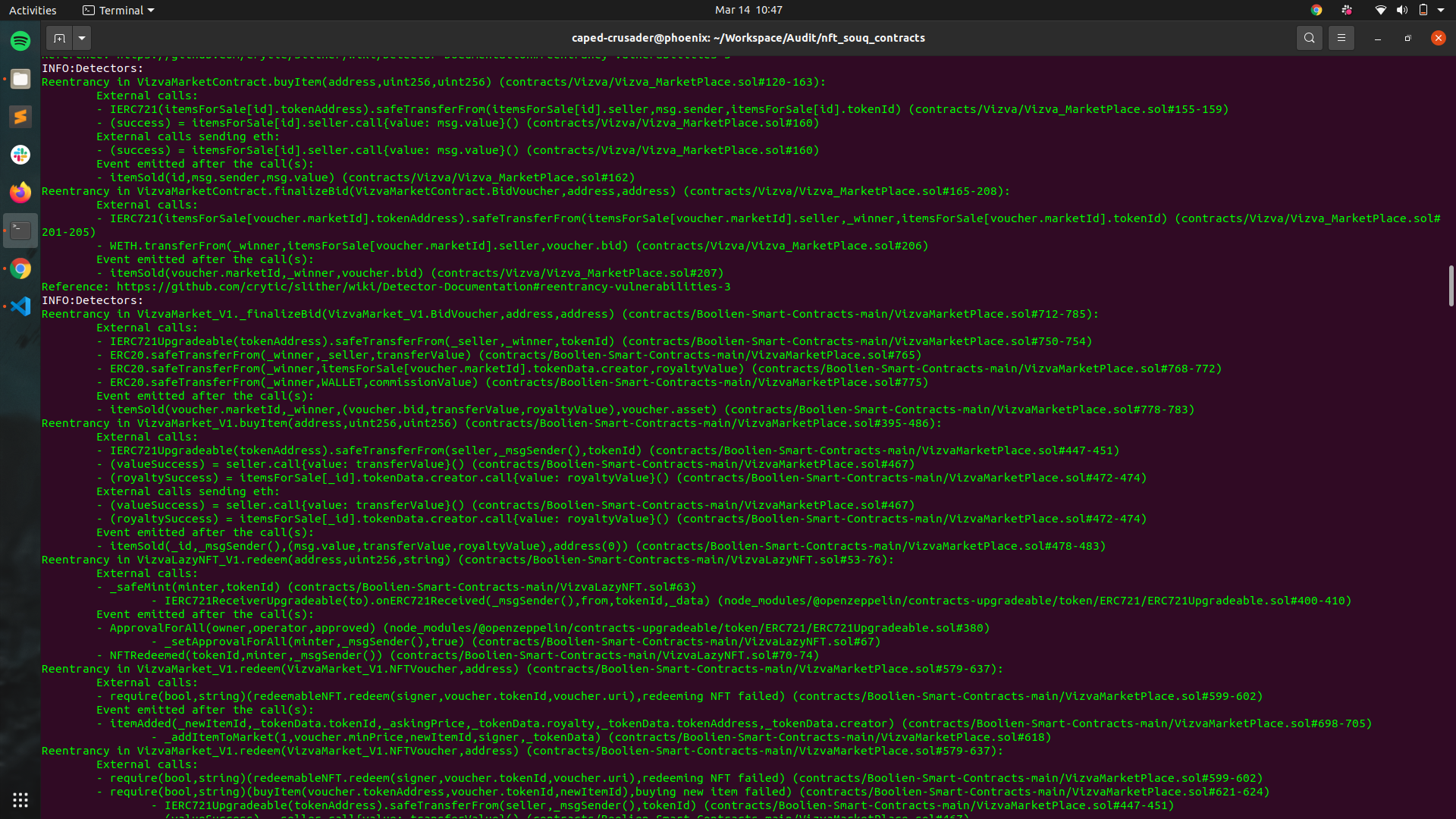
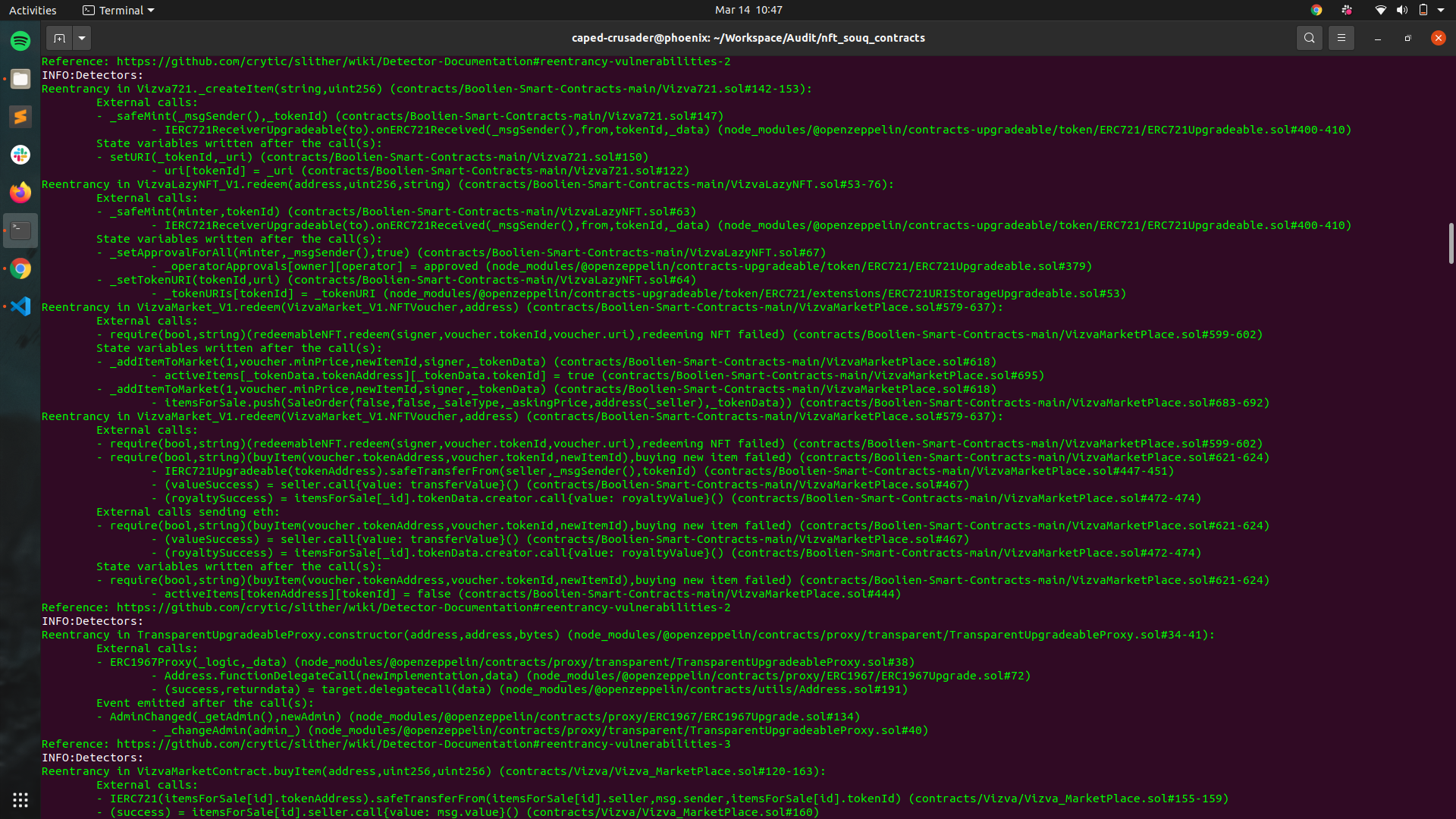
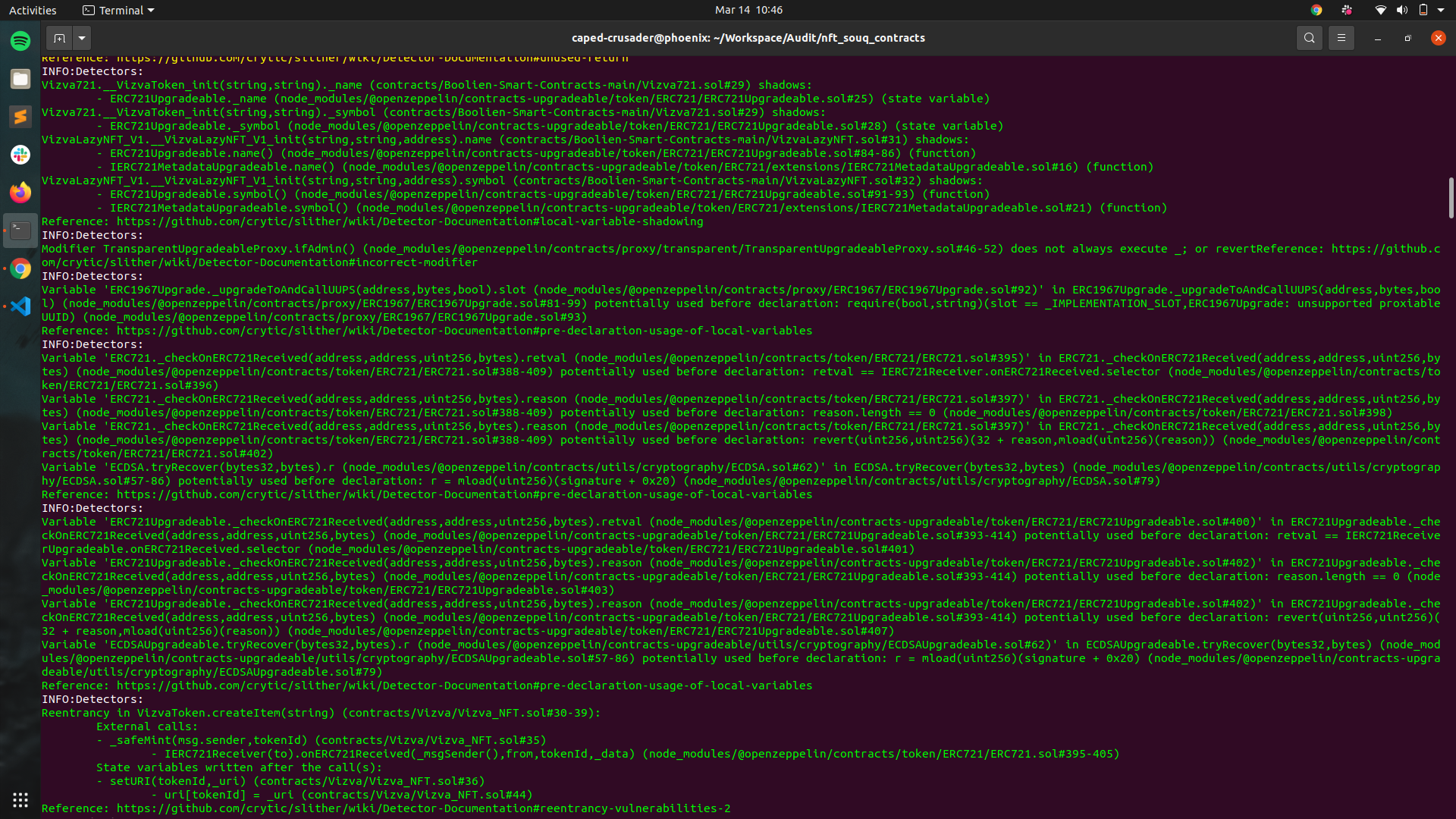
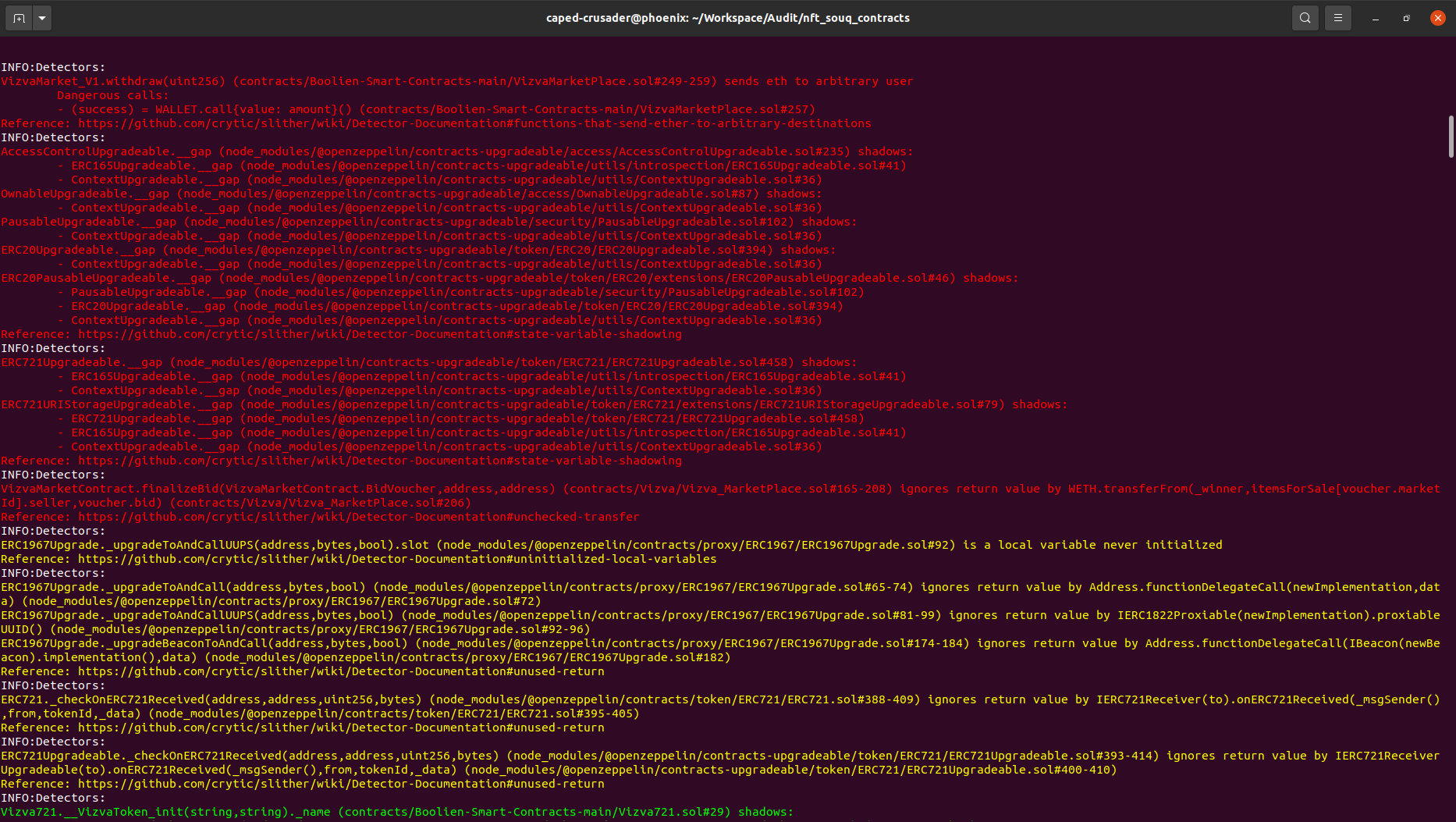
**Bool\_Token\_V1: 0xce8dac2407453747d0A4C3823dEFaC4bBBe673A7**

**VizvaLazyNFT:**  0x8934E84711ae593aA062c994aB3Ce2dF120b672C

**VizvaMarket\_V1:**  0xCAE9c959765BFcdF9A5c831186Aa971CbA7067D6

| addItemToMarket | 0x901164752cfc8b0557f2173aa4f927a24ccade4dfbf6aff5fda242a2331df150 | Pass |
| --- | --- | --- |
| Failed to add items not owned | 0x109ecd9cb6903e7057e96c48b5ffab5c1b8371d21658b92050e7f5e2ac09ece7 | Pass |
| BuyItem | 0xeb5ac335056ca86d3163cf21984b1d3740cf83580b0b8c07bbf2cbe56d357213 | Pass |
| Cancel Sale | 0x2da3e1a7ca1aa038859903efc48bd93c90e92b7b6408a8a38014afb2e940c833 | Pass |
| Unauthorized Redeem | 0xcd51512c80129cdd880bf8b2f8e34625d2b72faea7969eaa51f33e3016b71b10 | Pass |
| addItem for Bid | 0xfa06fd6415c2bb3df5f10cd837b36933c5f5aaacc3b8968181ac5276d1786251 | Pass |

# Automated Tests **Slither:**

****

# Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

# **Closing Summary**

Overall, smart contracts are very well written and adhere to guidelines.

Some minor and informational issues were discovered in the initial audit. It is recommended to kindly go through the above-mentioned details and fix the code accordingly.

# Disclaimer

Indiviual audit is not a security warranty, investment advice, or an endorsement of the **Vizva platform**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Marhaba Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.