# Security Audit Report

26/04/2022

GhostMarket

## Content

## Introduction

GhostMarket aspires to be a powerful cross-chain Non-Fungible Tokens (NFTs) worldwide marketplace, aiming to offer a comprehensive selection of features wrapped in an intuitive user interface. GhostMarket intents to feature a fully trustless NFT trading facilitated by smart contracts on each integrated blockchain. One of its most innovative features is its self-minting platform.

As requested by **GhostMarket** and as part of the vulnerability review and management process, Red4Sec has been asked to perform a security code audit in order to evaluate the security of the **GhostMarket** project.

The report includes specifics of the audit for the existing vulnerabilities of GhostMarket. The performed analysis shows that the **GhostMarket** project does contain critical vulnerabilities. Additionally, Red4Sec has found issues that need to be addressed and fixed. This document outlines the results of the security audit.

## Disclaimer

This document only represents the results of the code audit conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered neither "endorsement" nor "disapproval" of the guarantee of the correct business model of the analyzed project.

## Scope

**GhostMarket** has asked Red4Sec to perform an analysis of the project's **source code and Smart Contracts.**

Red4Sec has made a thorough audit of the smart contract security level against attacks, identifying possible errors in the design, configuration or programming; therefore, guaranteeing the confidentiality, integrity and availability of the information accessed, treated, and stored.

The scope of this evaluation includes the following projects enclosed in the repository:

- https://github.com/OnBlockIO/GhostMarketContractN3
  - branch: main
  - commit: `b4429e2a4203935ecedb953aae5cb50fa1287baf`
    - `/contracts/NEP17/GhostMarketToken.py`
    - `/contracts/NEP11/GhostMarket.NFT.py`

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange
  - branch: main
  - commit: `fa7296a84eb57f261d691520e89aec7cedb1d9e8`
    - `/manifest-proxy/*.go`
    - `/market-contract`

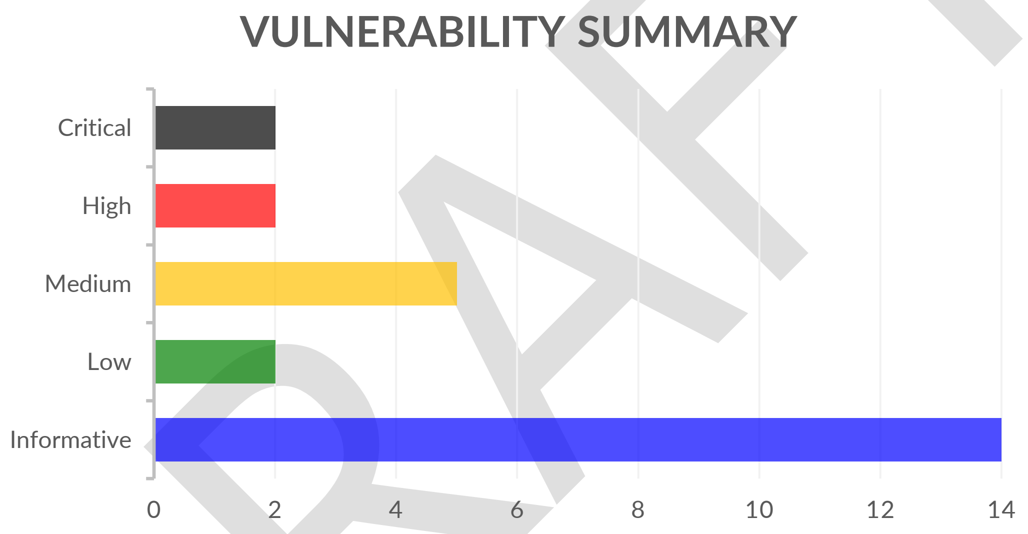
## Executive Summary

As requested by **GhostMarket**, the company Red4Sec Cybersecurity has been asked to perform a security audit against the **GhostMarket**, in order to assess the security of the source code and in addition to the vulnerability review and the management process.

This security audit has been conducted between the following dates: **04/05/2022** and **04/26/2022**.

Once the analysis of the technical aspects of the environment has been completed, the performed analysis shows that the audited source code contains 2 critical vulnerabilities that should be mitigated as soon as possible.

During the analysis, a total of **25 vulnerabilities** were detected. These vulnerabilities have been classified in the following levels of risk according to the impact level defined by CVSS (Common Vulnerability Scoring System):



## Conclusions

To this date, **26/04/2022**, the general conclusion resulting from the conducted audit, is that the **GhostMarket** project **is not completely secure** and does present some vulnerabilities that could compromise the security of the users and their information.
The general conclusions of the performed audit are:

- Critical and high-risk vulnerabilities were detected during the security audit. These vulnerabilities pose a great risk for **GhostMarket** project and must be fixed as soon as possible.

- Certain methods **do not make the necessary input checks** in order to guarantee the integrity and expected arguments format.

- The overall impression about code quality and organization is not optimal. The developed code does not comply with code standards and lacks essential security measures. Red4Sec has given some additional recommendations on how to continue improving and how to apply good practices.

- A few **low impact issues** were detected and classified only as informative, but they will continue to help GhostMarket improve the security and quality of its developments.

- The **GhostMarket** Project has been **developed using compiler outdated version.** This is an absolutely discouraged practice and should be solved prior to the deployment.

- Apply all the proposed recommendations considered necessary by Red4Sec in order to improve the security of the project.

- In order to deal with the detected vulnerabilities, an action plan must be elaborated to guarantee its resolution, prioritizing those vulnerabilities of greater risk and trying not to exceed the maximum recommended resolution times.

## Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered upon the security audit.

### List of vulnerabilities

Below, we have gathered a complete list of the vulnerabilities detected by Red4Sec, presented and summarized in a way that can be used for risk management and mitigation.

All these vulnerabilities have been classified in the following levels of risk according to the impact level defined by CVSS v3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST):

| Table of vulnerabilities | | | |
|---|---|---|---|
| **ID** | **Vulnerability** | **Risk** | **State** |
| **GSM-01** | Owner Contract Takeover | **Critical** | **Open** |
| **GSM-02** | Wrong transfer logic | **Critical** | **Open** |
| **GSM-03** | Unsecure Owner Verification | **High** | **Open** |
| **GSM-04** | Use of Assert instead of Throw | **High** | **Open** |
| **GSM-05** | Improve Storage Design of Locked Content | **Medium** | **Open** |
| **GSM-06** | Wrong NEP-11 Standard | **Medium** | **Open** |
| **GSM-07** | Lack of Inputs Validation in NEP Contracts | **Medium** | **Open** |
| **GSM-08** | Lack of Inputs Validation | **Medium** | **Open** |
| **GSM-09** | Limit Call Rights | **Medium** | **Open** |
| **GSM-10** | Use of Transaction Verify as Access Control | **Low** | **Open** |
| **GSM-11** | Denial of Service in the Query Methods | **Low** | **Open** |
| **GSM-12** | Lack of StorageMap | **Informative** | **Open** |
| **GSM-13** | Lack of Safe Method Attribute | **Informative** | **Open** |
| **GSM-14** | Modify storage before call external contracts | **Informative** | **Open** |
| **GSM-15** | Outdated Compiler Version | **Informative** | **Open** |
| **GSM-16** | Safe Storage Access | **Informative** | **Open** |
| **GSM-17** | Missing Manifest Information | **Informative** | **Open** |
| **GSM-18** | Bad Coding Practices | **Informative** | **Open** |
| **GSM-19** | GAS Optimization NEO | **Informative** | **Open** |
| **GSM-20** | Safe Contract Update/Destroy | **Informative** | **Open** |

| | | | |
|---|---|---|---|
| **GSM-21** | WhiteList Logic optimizacion | **Informative** | **Open** |
| **GSM-22** | NFT Royalty Standard | **Informative** | **Open** |
| **GSM-23** | Wrong IsContractSafe Design | **Informative** | **Open** |
| **GSM-24** | Unify and Validate ID format | **Informative** | **Open** |
| **GSM-25** | [Out of Scope] Steal Burned Tokens | **Informative** | **Open** |

## Vulnerability details

In this section, we provide the details of each of the detected vulnerabilities indicating the following aspects:

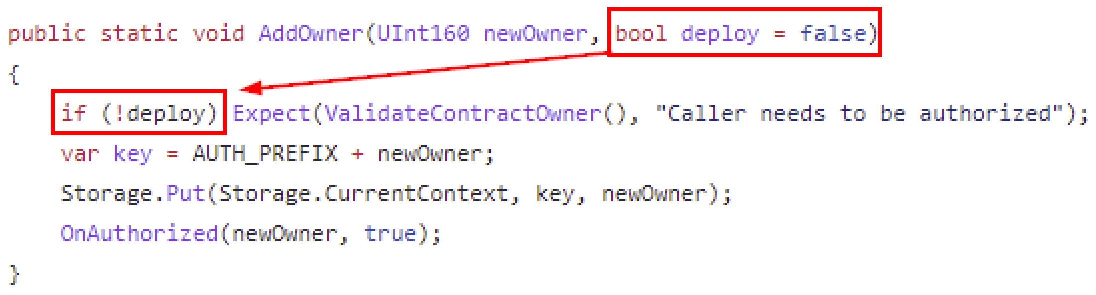
- Category
- Active
- Risk
- Description
- Recommendations

## Owner Contract Takeover

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-01** | Unprotected Privileged Function | **Critical** | **Open** |

The `GhostMarket` contract is vulnerable to arbitrary typing by the owners in the `AddOwner` method, this method allows any malicious user to add an owner to the contract without requiring any authentication. This will allow to take complete control of the contract.

In the following image we can see that the `AddOwner` method has the `deploy` argument to define whether or not you wish to verify if the owner calls to this method. Therefore, if an attacker uses `false`, no type of verification will be carried out.



```
public static void AddOwner(UInt160 newOwner, bool deploy = false)
{
    if (!deploy) Expect(ValidateContractOwner(), "Caller needs to be authorized");
    var key = AUTH_PREFIX + newOwner;
    Storage.Put(Storage.CurrentContext, key, newOwner);
    OnAuthorized(newOwner, true);
}
```

### Recommendations

It is essential to add adequate protections in order to prevent this behaviour, it would be convenient to eliminate the `deploy` variable and use a `AddOwnerInternal` method to be the one that executes the operation, to be able to call from the deploy, without going through the validation of the owner, which should never be an option.

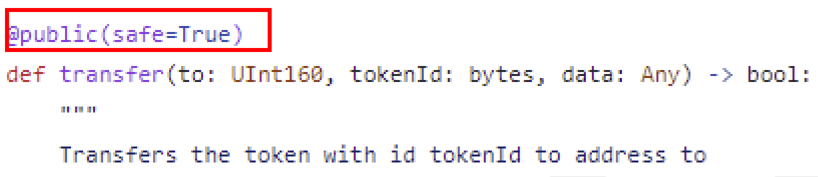
### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L679

## Wrong transfer logic

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-02** | DoS with Failed Call | **Critical** | **Open** |

The `transfer` method of the `GhostMarket.NFT` contract, it is currently defined as `safe`, which prevents any possible change in the storage. Considering the fact that during the logic of said method there are changes in the storage, the service will be denied.



```
@public(safe=True)
def transfer(to: UInt160, tokenId: bytes, data: Any) -> bool:
    """
    Transfers the token with id tokenId to address to
```

It has been verified that with the `neo3-boa 0.10.1` version, the `safe` attribute is omitted, and the manifest does not mark as `safe` the methods that ought to. However, by the time the compiler corrects this issue, the contract will be compiled in a way that denies service to any transfer of an NFT.

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L224

## Unsecure Owner Verification

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-03** | Missing Authorization | **High** | **Open** |

The `Transaction.Sender` global variable refers to the original account that started the transaction while `Runtime.CallingScriptHash` refers to the immediate account or contract that invokes the function. `Transaction.Sender` and `Runtime.CallingScriptHash` can be a contract or an external account but if there are multiple function invocations on multiple contracts, `Transaction.Sender` will always refer to the account that started the transaction, regardless of the stack of contracts invoked.

Never use `Transaction.Sender` for authorization, another contract can have a method which will call your contract and your contract will authorize that transaction as your address is in `Transaction.Sender`.

`Runtime.CallingScriptHash` should be used for authorization, so if another contract calls your `Runtime.CallingScriptHash` contract, it will be the address of the contract and not the address of the user who called the contract.

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1283-L1286

## Use of Assert instead of Throw

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-04** | Bad Coding Practices | **High** | **Open** |

The native `ExecutionEngine.Assert` must be used instead of `THROW`. The `ASSERT` and `ABORT` opcodes are approximately 9 times less expensive than the `THROW` opcode, and in the case of the `ExecutionEngine.Assert` it also allows to specify an error message.

`THROW` does not guarantee a revert of the transaction in every possible case, since it may be captured by the `try/catch` instructions; for this reason, its use must be limited only to some initial verifications in the methods when none of the values have been updated and even in these cases it is more recommended to use `ExecutionEngine.Assert`.

### Recommendations

**`THROW` it is only recommended to be used within a `try` sentence in the same contract,** otherwise it can end up being captured by an external contract and subsequently make unwanted modifications on the states of the contract.

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L303
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L712
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L719
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L726
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L733
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L816-L818
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L817
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L818
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L836
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L932-L934
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L942
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L953
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1011
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1089

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1093
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1205
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1209
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1266

### Out of Scope

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L54
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L91
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L98
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L131
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L161-L162
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L175-L184
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L200

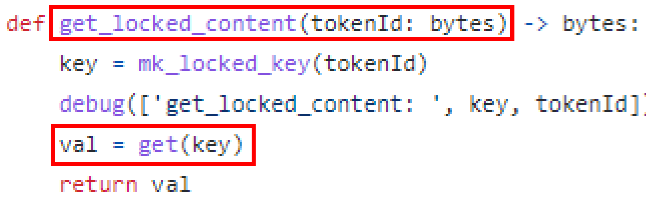
## Improve Storage Design of Locked Content

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-05** | Business Logic Errors | **Medium** | **Open** |

The logic implemented in the `GhostMarket.NFT` contract allows private information associated with a specific NFT to be stored. This information suggests that it will only be accessible by the current owner of the NFT, in fact, it is indicated on the web interface itself (https://ghostmarket.io/mint/n3).

```
"You can opt-in to have locked content. This is a URL which will only be accessible by t
he current NFT owner."
```

However, during the audit it was detected that the private content is stored in plain text, thus it does not have any encryption mechanism with the public key of the current owner. This mechanism is certainly not adequate and could be considered a weak procedure to manage secrecy.



```
def get_locked_content(tokenId: bytes) -> bytes:
    key = mk_locked_key(tokenId)
    debug(['get_locked_content: ', key, tokenId])
    val = get(key)
    return val
```

It is important to highlight that the storages of the blockchain are already "public", so storing sensitive information in the blockchain is not ideal.

### Recommendations

Inform the user that this content can be read by third parties that have access to the storage of the blockchain.

### References

- https://ghostmarket.io/mint/n3/

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L625

## Wrong NEP-11 Standard

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-06** | Bad Coding Practices | **Medium** | **Open** |

The `NEP11` standard establishes the methods to be implemented by the project, however the following discrepancies have been found in the audited token.

The standard defines the `properties` and `transfer` methods as follows:

```
{
  "name": "properties",
  "safe": true,
  "parameters": [
    {
      "name": "tokenId",
      "type": "ByteString"
    }
  ],
  "returntype": "Map"
}
{
  "name": "transfer",
  "safe": false,
  "parameters": [
    {
      "name": "to",
      "type": "Hash160"
    },
    {
      "name": "tokenId",
      "type": "ByteString"
    },
    {
      "name": "data",
      "type": "Any"
    }
  ],
  "returntype": "Boolean"
}
```

Nevertheless, these methods are defined as follows:

- `def properties(tokenId: bytes) -> Dict[str, str]:` instead of `def properties(tokenId: ByteString) -> Dict[Any, Any]:`.
- `def transfer(to: UInt160, tokenId: bytes, data: Any) -> bool:` instead of `def transfer(to: UInt160, tokenId: ByteString, data: Any) -> bool:`

Additionally, the `properties` and `ownerOf` methods must be defined as indicated in the following issue **Lack of Safe Method Attribute**.

This produces exceptions with the last version of the `neo3-boa 0.11.2` compiler, by integrating a standards detection during the compilation.

```
ERROR: 'NEP-11': Missing 'transfer' method definition 'public (UInt160, ByteString,
 any) -> bool'

ERROR: 'NEP-11': Missing 'ownerOf' method definition 'public (ByteString) -> UInt16
0'

ERROR: 'NEP-11': Missing 'Transfer' event definition 'public (Union[none, UInt160],
 Union[none, UInt160], int, ByteString)'

ERROR: 'NEP-11': Missing 'properties' method definition 'public (str) -> dict[any,
any]'
```

### Recommendations

It is essential that the methods of the `NEP11` standard are rigorously defined as specified in the standard.

### References

- https://github.com/neo-project/proposals/blob/master/nep-11.mediawiki

## Lack of Inputs Validation in NEP Contracts

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-07** | Bad Coding Practices | **Medium** | **Open** |

It has been verified that various methods of the different contracts in the GhostMarket `NEP-11` and `NEP-17` contracts do not properly check the arguments, which can lead to major errors.

It is advisable to always check the format and type of the arguments before using their value, otherwise, a user could send unexpected values through these arguments, being able to make injections or arbitrary reads from the storage, either intentionally or not.

It is important to know that the type of the arguments from an invocation are not natively verified by NEO VM and that neo3-boa does not add any type of verification during its compilation, therefore, all the arguments must be carefully verified.

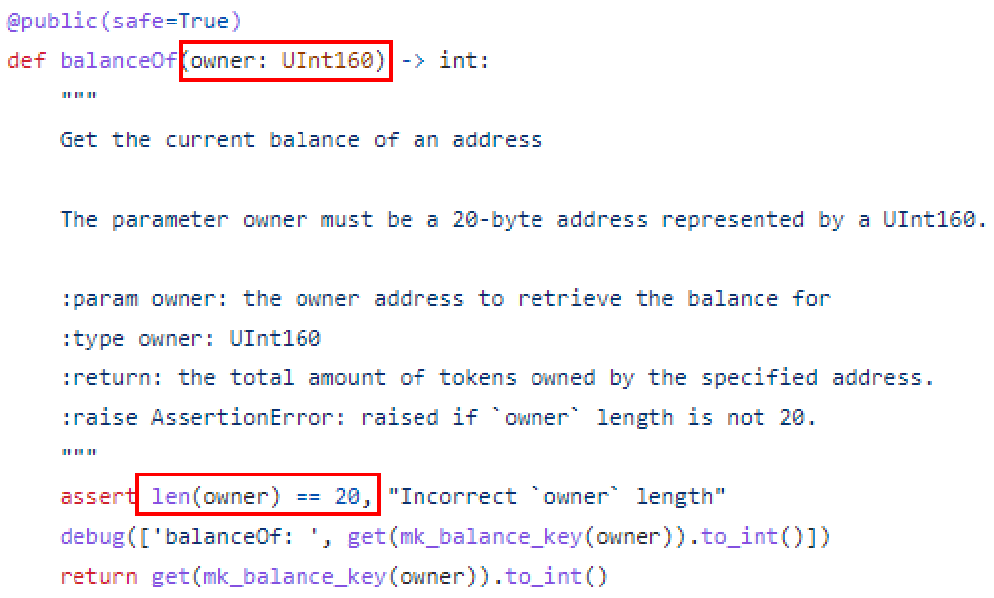
The integrity of the `UInt160` types is only verified by its size (20 bytes) however, it is more convenient to use the `isinstance()` method since it makes verifications by type and size.

Additionally, in some methods it is convenient to check that the address value is not address `zero`, which could lead to unwanted behaviours in certain occasions.

It is convenient to add a method that unifies the format verification of the addresses, as suggested below:

```
def validateAddress(address: UInt160) -> bool:
    if not isinstance(address, UInt160):
        return False
    if address == 0:
        return False
    return True
```

An example of not correctly verifying the `UInt160` types is found in the `balanceOf` method of the `GhostMarket.NFT` contract.



```
@public(safe=True)
def balanceOf(owner: UInt160) -> int:
    """
    Get the current balance of an address

    The parameter owner must be a 20-byte address represented by a UInt160.

    :param owner: the owner address to retrieve the balance for
    :type owner: UInt160
    :return: the total amount of tokens owned by the specified address.
    :raise AssertionError: raised if `owner` length is not 20.
    """
    assert len(owner) == 20, "Incorrect `owner` length"
    debug(['balanceOf: ', get(mk_balance_key(owner)).to_int()])
    return get(mk_balance_key(owner)).to_int()
```

In addition to the verifications of types for `UInt160` arguments, it must be mentioned that there are methods where the integer arguments are not checked to be greater than zero.

### Recommendations

It is advisable to always check the format of the arguments before using their value, otherwise, a user could send unexpected values through these arguments, being able to make injections or arbitrary reads from the storage, either intentionally or not.

### Source Code References

Check `UInt160` with `validateAddress`

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L203
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L219
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L249
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L558
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L142
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L168

Verify the type of `status` in order to avoid any value in `PAUSED` to be registered.

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L742

Check `amount` value to avoid negative numbers, in order to prevent possible problems.

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L953

Check `feeBps` in order to avoid that the value is negative, since the `mint` method requires that the `fee` is not negative and in case the `fee` is in fact negative, it will trigger a denial of service.

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L599
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L482

## Lack of Inputs Validation

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-08** | Bad Coding Practices | **Medium** | **Open** |

It has been detected that certain methods of the different contracts in the `GhostMarket` project do not properly check the arguments, which can lead to major errors.

The general execution of the `GhostMarket` contract is not checking the inputs nor the integrity of most the `UInt160` types, and in various cases, integers with negative value. This bad practice together with the absence of the use of `StorageMap`, considerably increases the risk of suffering from injections in the storage.

Additionally, in certain methods it is convenient to check that the value is not `IsZero`, leaving the verification as: `hash.IsValid && !hash.IsZero`.

### Recommendations

It is advisable to always check the format of the arguments before using their value, otherwise, a user could send unexpected values through these arguments, being able to make injections or arbitrary reads from the storage, either intentionally or not.

### Source Code References

Check `UInt160` format.

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L146
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L156
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L174
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L199
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L207
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L243
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L279
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L679
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L687
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L724
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L855
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L880
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L929

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L939
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L951

Check `UInt160` is not zero.

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L679
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L929
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L939

Check number value to avoid negative numbers, in order to prevent possible problems.

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L146
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L279
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L394
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L710
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L717
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L731
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L748
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L778

Check `feeBps` in order to avoid that the value is negative, since the `mint` method requires that the `fee` is not negative and in case the `fee` is in fact negative, it will trigger a denial of service.

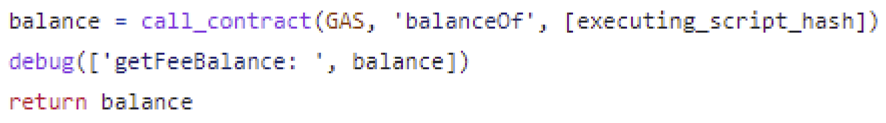
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L146

## Limit Call Rights

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-09** | Reentrancy | **Medium** | **Open** |

It is important to highlight that in certain cases, the witnesses scope extends beyond the invoked contracts and that there is a possibility that the invoked contract makes a reentrancy; so, it is advisable to use the principle of least privilege (PoLP) during all the external processes or the calls to the contracts.

Therefore, when making the call to any contract it is expected to be read-only; as it is the case of obtaining the user's balance, this call should always be made with the `ReadOnly` flag instead of `CallFlags.All.`



```
balance = call_contract(GAS, 'balanceOf', [executing_script_hash])
debug(['getFeeBalance: ', balance])
return balance
```

### References

- https://en.wikipedia.org/wiki/Principle_of_least_privilege

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L569
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L583
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L896
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L1100
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L1116
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L1121
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L1126
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L1254
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L1261
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L253

### Out of scope

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/contract-call/ContractCall.cs#L21

## Use of Transaction Verify as Access Control

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-10** | Improper Privilege Management | **Low** | **Open** |

The `GhostMarketToken` and `GhostMarket.NFT` contracts make use of the `Verify` method to control access to the administrative functions of the contract. The `Verify` method executes a special functionality of validating the transactions or actions issued by the smart contract on the N3 blockchain.

The use of the `Verify` method as access control, such as `onlyOwner`, or from roles to functions of the contract is an absolutely discouraged practice. Any user/owner/admin with permissions to successfully invoke `Verify` can update, destroy or withdraw from the contract even if there are no methods with this functionality or existing and they are limited for the user.

### Recommendations

We must distinguish if inside Neo's blockchain there are different access controls to each profile, between the adminsitrative actions over the implemented logic in the smart contract and the actions/transactions of the contract.

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L267
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L343
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L387
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L568
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L608
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L822
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L833

## Denial of Service in the Query Methods

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-11** | DoS With Block Gas Limit | **Low** | **Open** |

Different query methods of the contract return a list of characteristics without considering that neo's virtual machine has a limitation of *2048* elements in the return of the operations and a denial of service could occur with higher values.

Taking into consideration a scenario where these query methods fail at some point by exceeding the limits of the virtual machine and returning `FAULT` in its execution, a denial of service would be produced, this limits the affected methods to fewer than 600 entries.

### Source Code References

Affected methods:

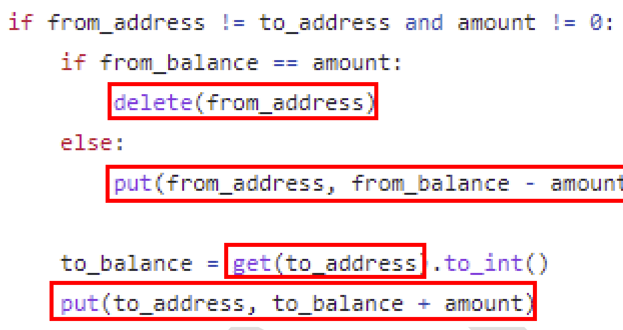
- `GetCustomFees`
  - https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L189
- `GetRegistered`
  - https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L221
- `GetExcessiveBalance`
  - https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L863
- `GetAllTokensNotInOrders`
  - https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L899
- `FixTotalAmount`
  - https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1215

## Lack of StorageMap

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-12** | Bad Coding Practices | **Informative** | **Open** |

It has been detected that the storage of Smart Contract values is not performed in a secure manner. It is recommended to use `StorageMap`, instead of `Storage`, since this class adds the prefix and avoids possible errors.

In the following image you can see how the `StorageMap` class is not used:



```
if from_address != to_address and amount != 0:
    if from_balance == amount:
        delete(from_address)
    else:
        put(from_address, from_balance - amount)

    to_balance = get(to_address).to_int()
    put(to_address, to_balance + amount)
```

*This deficiency can be found throughout all the audited contracts.*

### Recommendations

By adding prefixes with `StorageMap` to the storages, a large part of the vulnerabilities related to arbitrary writing in storage contracts would be avoided, mitigating in some way the appearance of new vulnerabilities.

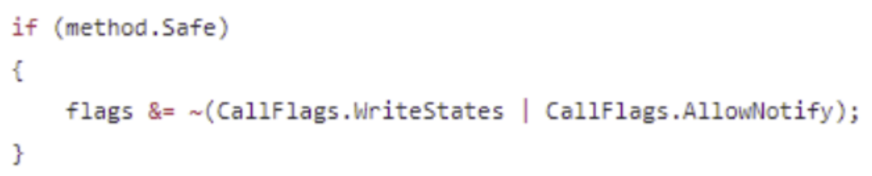
### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs

## Lack of Safe Method Attribute

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-13** | Bad Coding Practices | **Informative** | **Open** |

In N3 there is a `Safe` attribute which defines that the call to the contract will create an execution context where the storage will not be modifiable or able to produce notifications. This characteristic turns the `Safe` methods into secure query methods.



```
if (method.Safe)
{
    flags &= ~(CallFlags.WriteStates | CallFlags.AllowNotify);
}
```

Additionally, it will provide the wallets and dApps with the necessary information to identify it as a query method and to make a reading invocation with no GAS costs. So it is convenient to establish our query methods as `Safe` to keep the principle of least privilege.

Furthermore, in the case of the `GhostMarket.NFT` and `GhostMarketToken` tokens, exceptions occur with the latest version of the `neo3-boa 0.11.2` compiler as it integrates standards detection during the compilation and lacks these attributes.

### References

- https://en.wikipedia.org/wiki/Principle_of_least_privilege

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L401
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L416
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L756
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L92
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L106
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L119
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L132
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L271
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L285
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L306
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L363

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L392
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L214
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L243
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L279
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L855
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L880
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1155
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1251
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1258

## Modify storage before call external contracts

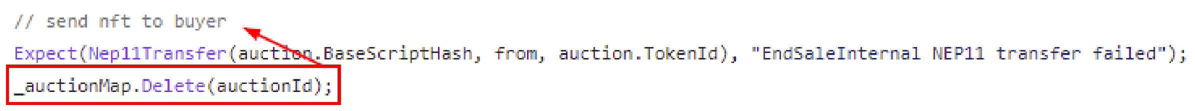
| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-14** | Reentrancy | **Informative** | **Open** |

Making adequate changes to the storage before external calls will prevent reentrance-type attacks.

The Reentrancy attack is a vulnerability that occurs when external contract calls can make new calls to the calling contract, before the initial execution is completed. For a function, this means that the state of the contract could change in the middle of its execution as a result of a call to an untrusted contract or the use of a low-level function with an external address.

This attack is possible in N3 because the NEP17 and NEP11 standards establish that after sending tokens to a contract, the `onNEPXXPayment()` method must be invoked. Therefore, transfers to contracts will invoke the execution of the payment method of the recipient and the recipient may redirect the execution to himself or to another contract.

In the case of the `EndSaleInternal` method of the `GhostMarket` contract, the values are updated after the transfers are made so the call can be redirected to any method before updating the values. In this situation, no problem has been detected since in the case of wanting to transfer the NFT again, the transfer will fail, but undoubtedly it is a completely unadvised practice that can open new vectors of attacks.



```
// send nft to buyer
Expect(Nep11Transfer(auction.BaseScriptHash, from, auction.TokenId), "EndSaleInternal NEP11 transfer failed");
_auctionMap.Delete(auctionId);
```

*The risk of this vulnerability has been reduced from critical to informative, because it has not been possible to satisfactorily exploit it.*

### Recommendations

It is **essential to always make the state changes in the storage before making transfers or calls to external contracts**, in addition to implementing the necessary measures to avoid duplicated calls or chain calls to the methods.

### References

- https://github.com/neo-project/proposals/blob/master/nep-17.mediawiki
- https://github.com/neo-project/proposals/blob/master/nep-11.mediawiki

### Source Code References

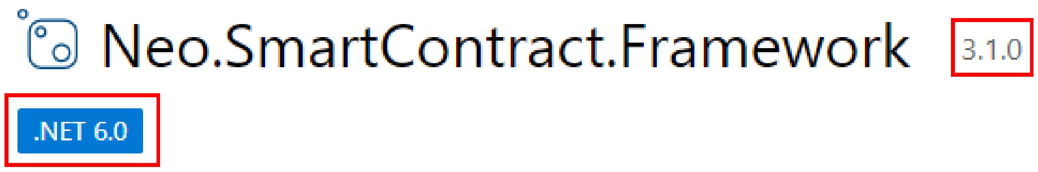
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L1080

## Outdated Compiler Version

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-15** | Outdated Software | **Informative** | **Open** |

The neo's C# compiler frequently launches new versions of the compiler. Using an outdated version of the compiler can be problematic, especially if there are errors that have been made public or known vulnerabilities that affect this version.

It has been verified that the project is compiled for the version `5` of .net framework, and it uses Neo's framework `3.0.2`. Both versions can be updated and contain major improvements, as can be seen in the following image.


Neo.SmartContract.Framework 3.1.0

.NET 6.0

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.csproj#L8

### Out of Scope

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.csproj#L8
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/contract-call/ContractCall.csproj#L8

## Safe Storage Access

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-16** | Bad Coding Practices | **Informative** | **Open** |

N3 contains different types of storage access, being `CurrentReadOnlyContext` the most appropriate one for the read-only methods; using a read-only context prevents any malicious change to the states. As in the rest of the cases, it is important to follow the principle of least privilege (PoLP) in order to avoid future problems.

### References

- https://en.wikipedia.org/wiki/Principle_of_least_privilege

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L222
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L312
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L686
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L176
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L186
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L218
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L248
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L699
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L752
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L782
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L858
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L882
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L913
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1151
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1158
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1212
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1273

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1285

Out of scope

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L48
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L55
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L62
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L83
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L92
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L178
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L245

## Missing Manifest Information

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-17** | Bad Coding Practices | **Informative** | **Open** |

The Red4Sec team has detected that the audited contracts do not properly specify the information related to the Smart Contract in its manifest.

All the contact information identified in the contract's manifest (*author, email, description, source*) belongs to the developer of the project. Therefore, it is recommended to customize said content by adding your own information, which will also provide the users with relevant information about the project, increase the trust and improve the SEO of the contract.

### References

- https://github.com/neo-project/proposals/blob/master/nep-15.mediawiki

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L11-L13
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L35
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L29

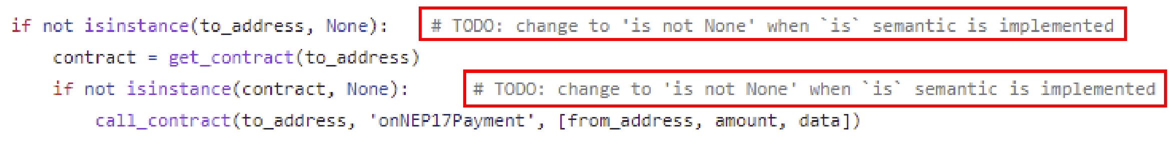
## Bad Coding Practices

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-18** | Bad Coding Practices | **Informative** | **Open** |

During the audit of the Smart Contract certain bad practices have been detected throughout the code that should be improved, it is always recommended to apply various coding style and good practices.

### Open TODO

For example, certain to-do comments have been detected that reflect that the code is unfinished. Thus, many unaudited changes could introduce new vulnerabilities in the future.



```
if not isinstance(to_address, None):   # TODO: change to 'is not None' when `is` semantic is implemented
    contract = get_contract(to_address)
    if not isinstance(contract, None):    # TODO: change to 'is not None' when `is` semantic is implemented
        call_contract(to_address, 'onNEP17Payment', [from_address, amount, data])
```

#### Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L217-L219
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L540
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L486

### Hardcode values

Establishing hardcoded addresses in the contract is a discouraged practice that instigates human errors, which is why it is recommended to use the `data` argument of the `_deploy` method to establish the necessary variables of the network on which the contract is deployed, so that there are no more variables than necessary and that does not require modifying the script to adapt it to the desired network.

```
[InitialValue("NeuknLsWKNj64zUDyvgdJFFfUqgkCvx5oJ", ContractParameterType.Hash160)]
private static readonly UInt160 ManifestProxyMainnet = default;

[InitialValue("NiQ8iNrbJnto142AW3EwGpuAjZMZrEjjdK", ContractParameterType.Hash160)]
private static readonly UInt160 ManifestProxyTestnet = default;

[InitialValue("NYp7pX59WACC2WqThXFFMh4EFgRWZJ4PBu", ContractParameterType.Hash160)]
private static readonly UInt160 ManifestProxyNeoXP = default;

private static UInt160 GetContractForNetwork(uint network)
{
    switch(network)
    {
        case 860833102: // Mainnet
            return ManifestProxyMainnet;

        case 877933390: // Testnet
            return ManifestProxyTestnet;

        case 768299814: // neo-express
            return ManifestProxyNeoXP;
    }

    return null;
}
```

Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/master/market-contract/GhostMarket.cs#L1186

## Homogenize storage prefixes

It has been identified that different *storage* prefixes, for the same functionality, are used throughout the contracts. It is convenient to use the same prefixes in all the contracts to avoid possible development errors.

Additionally, it is important to define a standard for the prefixes so the dApps that requires to query the storage can have a better unified access to said values; for example, in the case of the `GhostMarket.NFT`, the `LOCKED_VIEW_COUNT_PREFIX` key is storedwith a prefix size of `4` digits, while other keys use `3` digits.

Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L62-L70

## Homogenize precision

It has been possible to verify that the percentages of the fee are based on 10_000, while the percentages of increase in the bids are based on 100. This can facilitate the occurrence of human errors. Since it is necessary to know the different powers on which each of the percentages are computed in order to define them.
It is recommended to unify these percentages to have the same precision in all of the operations.

### Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L526
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L846
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L967

## GAS Optimization NEO

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-19** | Bad Coding Practices | **Informative** | **Open** |

Software optimization is the process of modifying a software system to make an aspect of it work more efficiently or use less resources. This premise must be applied to smart contracts as well, so that they execute faster or in order to save GAS.

On the N3 blockchain, GAS is an execution fee which is used to compensate the network for the computational resources required to power smart contracts. If the network usage is increasing, so will the value of GAS optimization.

These are some of the requirements that must be met to reduce GAS consumption:

- Short-circuiting.
- Remove redundant or dead code.
- Delete unnecessary libraries.
- Use of proper data types.
- Use hard-coded CONSTANT instead of state variables.
- Avoid expensive operations in a loop.
- Pay special attention to arithmetical operations and comparisons.

### Static Variables

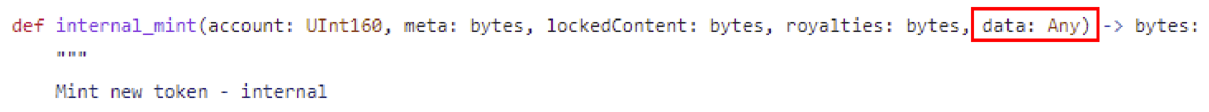
The static variables in Neo are processed at the beginning of the execution of the smart contract, so they must be carefully used. Although these static variables are not used for the call that will be made, their initialization will be processed anyway, and they will occupy elements in the stack with the corresponding cost that this entails.

It is a good practice to reduce the static variables, either through constants or through methods that return the desired value.

### Dead Code

In programming, a part of the source code that is never used is known as dead code. The execution of this type of code consumes more GAS during deployment in something that is not necessary.

The `data` argument is not used in the `internal_mint` method of the `GhostMarket.NFT` contract, so it would be convenient to either remove it or use it.



```
def internal_mint(account: UInt160, meta: bytes, lockedContent: bytes, royalties: bytes, data: Any) -> bytes:
    """
    Mint new token - internal
```

The methods `onNEP11Payment` and `onNEP17Payment` of the `GhostMarket.NFT` contract and `onNEP11Payment` of the `GhostMarketToken` contract execute an `abort` with the intention of not being able to receive any `NEP11/NEP17` token, however if it does not contain these methods, the logic of a transfer to said contract will automatically fail, but it will be more low-cost for the user who executes it.

Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L272-L299
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L401-L413
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L862

The `AUCTION_IDS_PREFIX`, `MIN_BID_INCREASE_PREFIX`, `TRADE_FEE_PREFIX`, `TRADE_FEE_ADDRESS_PREFIX` and `ROYALTY_CACHE_PREFIX` prefixes of the `GhostMarket` contract, are not used and they can be eliminated.

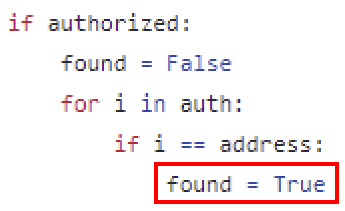
Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L57
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L61
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L63
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L65
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L67
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L88

## Logic Optimization

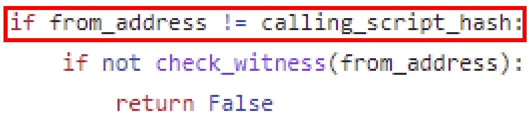
The logic related to the detection of `authorized` addresses contains a loop that can be optimized.

The logic related to these parts of code is intended to know if the address is authorized, for this reason, once it has been detected as `found=true`, it is not necessary to continue iterating the loop, and the `break` statement can save unnecessary costs of gas.



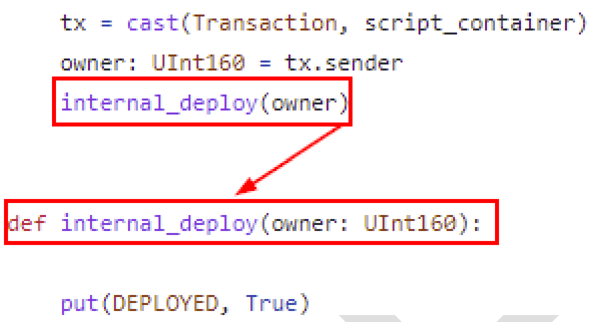
```
if authorized:
    found = False
    for i in auth:
        if i == address:
            found = True
```

The `GhostMarketToken` contract checks the calling script before calling `check_witness`, however this check is already performed in the `check_witness` syscall, so executing it in the smart contract will only increase the costs.



```
if from_address != calling_script_hash:
    if not check_witness(from_address):
        return False
```

The `internal_deploy` method is only called in the `_deploy` method, so removing the `internal_deploy` method will improve readability as well as reduce execution costs.



```
tx = cast(Transaction, script_container)
owner: UInt160 = tx.sender
internal_deploy(owner)
```

```
def internal_deploy(owner: UInt160):

    put(DEPLOYED, True)
```

Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L351
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L693
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L729
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L182
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L241

## Unnecessary logic

During the execution of the `GhostMarket.NFT` and `GhostMarketToken` contracts, a `Debug` event is emitted that increases the user's costs in order to help the developer in their debugging tasks. However, these instructions should be avoided in production and use compiler directives or other systems to avoid increasing costs to users once the contract is on mainnet.

Source reference

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L74-L85
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L134-L145
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L365-L379

Out of Scope

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L29

## Safe Contract Update/Destroy

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-20** | Business Logic Errors | **Informative** | **Open** |

It is important to mention that the owner of the contract has the possibility of updating the contract, which implies a possible change in the logic and in the functionalities of the contract, reducing part of the concept of decentralized trust.

Although this is a recommended practice in these early phases of the N3 blockchain where significant changes can still take place, it would be convenient to include certain protections to increase transparency for the users so they can act accordingly.

### Recommendations

There are certain good practices that help mitigate this problem, such as; add a `timelock` to start the `Update` operations, emit events when the `Update` operation is requested, temporarily disable contract functionalities and finally issue a last notification and execute the `Update` operation after the `timelock` is completed.

### Source Code References

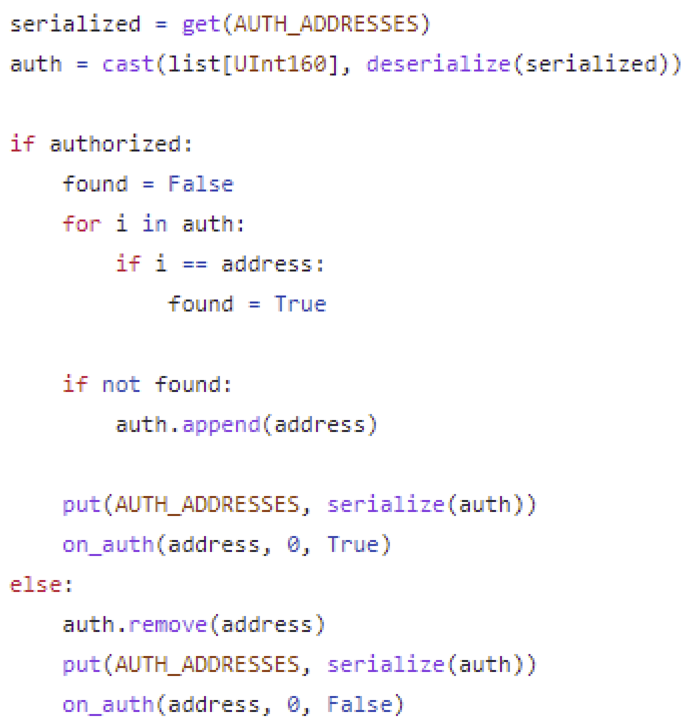
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L257
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L812
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L827

## WhiteList Logic optimizacion

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-21** | Code Optimization | **Informative** | **Open** |

In the `GhostMarketToken` and `GhostMarket.NFT` contracts, when checking if an address is authorized or in the whitelist, a serialized array is used. This automatically limits the number of possible entries, in addition to assuming unnecessary iteration costs to check if said array exists.

It is convenient to use a `StorageMap` so that it can be iterated with the `Find` method. Subsequently making the process of checking if an address exists almost automatic, since by verifying that the key exists in the storage, the loop that goes through all the entries can be avoided.



```
serialized = get(AUTH_ADDRESSES)
auth = cast(list[UInt160], deserialize(serialized))

if authorized:
    found = False
    for i in auth:
        if i == address:
            found = True

    if not found:
        auth.append(address)

    put(AUTH_ADDRESSES, serialize(auth))
    on_auth(address, 0, True)
else:
    auth.remove(address)
    put(AUTH_ADDRESSES, serialize(auth))
    on_auth(address, 0, False)
```

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L787-L792
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L345-L351
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP17/GhostMarketToken.py#L403-L409

## NFT Royalty Standard

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-22** | Improper Adherence to Coding Standards | **Informative** | **Open** |

The payment and the distribution of royalties of NFT's can be complicated tasks, so different standards have emerged over time to cover this need. Although it is not mandatory, it is always advisable to be based on a widespread and tested standard.

N3 does not currently have any standard or NEP destined to describe this feature, so it may be convenient to study the developments of the rest of the blockchains and attempt to replicate the most widespread standards in the sector, such as the Ethereum Royalties standard defined in `EIP2981`.

### Recommendations

It is recommended to define a standard and create the according NEP, or to study the standards of the industry and implement them.

### References

- https://eips.ethereum.org/EIPS/eip-2981

### Source Code References

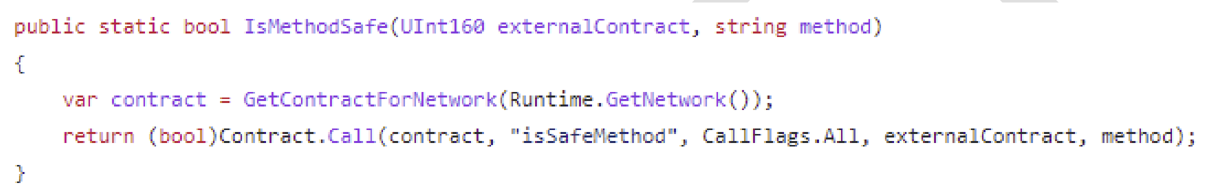
- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L544

## Wrong IsContractSafe Design

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-23** | Bad Coding Practices | **Informative** | **Open** |

During the audit of the Smart Contract certain bad practices have been detected throughout the code that should be improved, such as the logic related to the `IsMethodSafe` method of the `GhostMarket` contract which is over-engineered.

This function intents to check if a method of a contract is `Safe`, so it calls an external contract in Go that reads the manifest of the contract and returns the result. This logic only guarantees that at the time of this verification the contract method is `Safe`, but it does not prevent said contract from being updated in the future and modifying the method, which has direct implications for the security of the smart contract.



```
public static bool IsMethodSafe(UInt160 externalContract, string method)
{
    var contract = GetContractForNetwork(Runtime.GetNetwork());
    return (bool)Contract.Call(contract, "isSafeMethod", CallFlags.All, externalContract, method);
}
```

The safest way to trust that the call to a third-party contract is safe is to use the `CallFlag` of the `ReadOnly`, in this manner the call will have limited permissions even if the code changes in the future.

### Recommendations

It is recommended to limit the permissions of the calls to external contracts with the `CallFlags` of the `ReadOnly`.

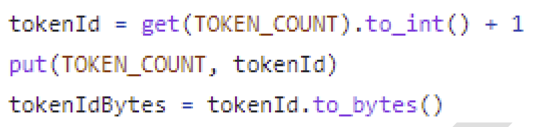
### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/market-contract/GhostMarket.cs#L1258

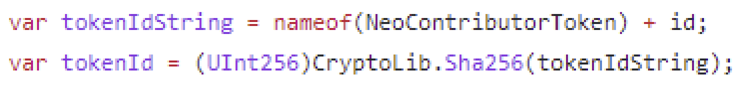
## Unify and Validate ID format

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-24** | Bad Coding Practices | **Informative** | **Open** |

The `GhostMarket.NFT` contract uses a `tokenId` format that is incremental and numeric, on the contrary, the `NeoContributorToken` contract uses a 32-byte format resulting from `SHA256` for the same `tokenId`.



```
tokenId = get(TOKEN_COUNT).to_int() + 1
put(TOKEN_COUNT, tokenId)
tokenIdBytes = tokenId.to_bytes()
```



```
var tokenIdString = nameof(NeoContributorToken) + id;
var tokenId = (UInt256)CryptoLib.Sha256(tokenIdString);
```

In both contracts the user's inputs are not cleared as valid inputs in the case of a `tokenId`, there must be a uniform implementation between the contracts, which allows validating the user's input and not allowing arbitrary values for `tokenId`.

### Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3/blob/b4429e2a4203935ecedb953aae5cb50fa1287baf/contracts/NEP11/GhostMarket.NFT.py#L881

### Out of scope

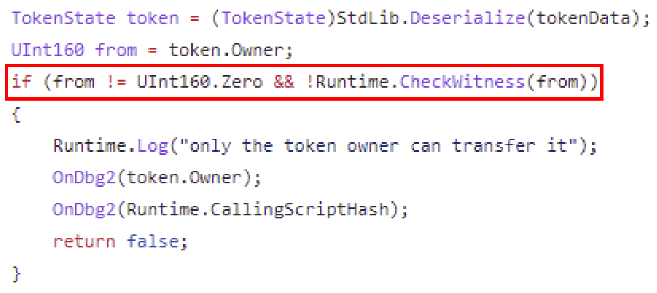
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L140

## [Out of Scope] Steal Burned Tokens

| Identifier | Category | Risk | State |
|---|---|---|---|
| **GSM-25** | Business Logic Errors | **Informative** | **Open** |

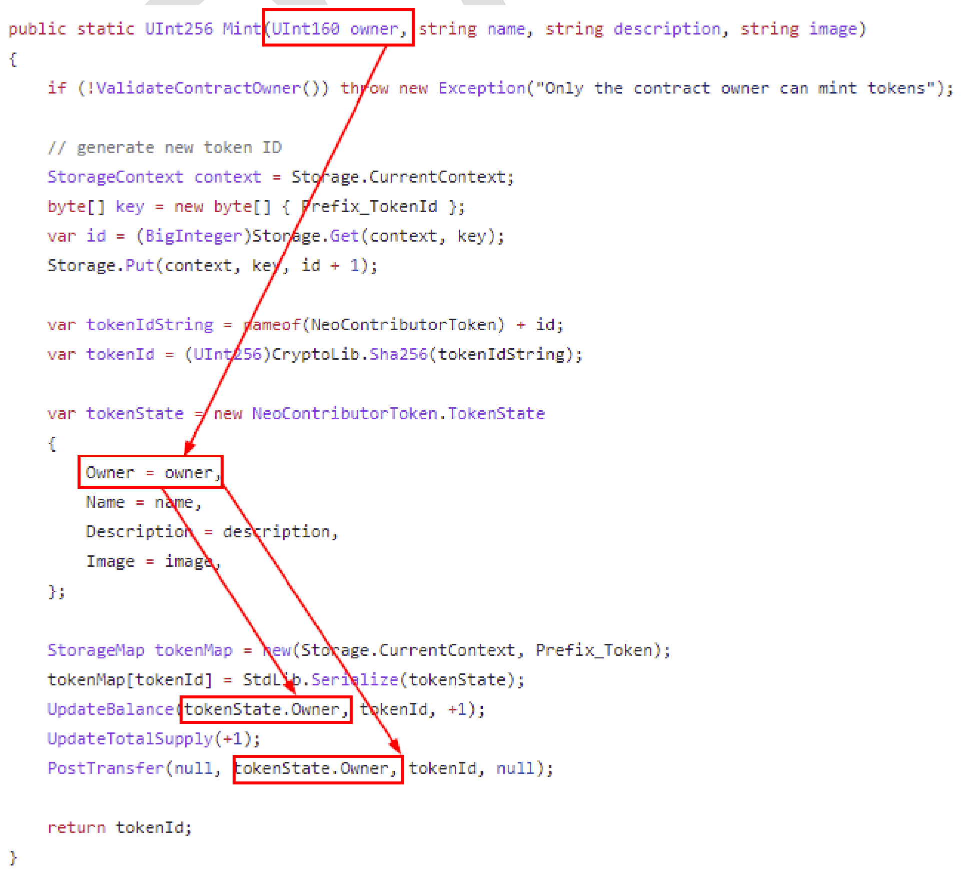
The `Transfer` method of the `NeoContributorToken` contract contains conditions that allow any user to retrieve all the funds that have been sent to the `UInt160.Zero` address.

The validation of the token's property during the `transfer` method is performed only if the owner is different from `UInt160.Zero`. Therefore, any transfer with this origin is accepted without verification. In the case of burning tokens sent to the `UInt160.Zero` address, they can be recovered by any user who makes a transfer from the `UInt160.Zero` address to their wallet.



```
TokenState token = (TokenState)StdLib.Deserialize(tokenData);
UInt160 from = token.Owner;
if (from != UInt160.Zero && !Runtime.CheckWitness(from))
{
    Runtime.Log("only the token owner can transfer it");
    OnDbg2(token.Owner);
    OnDbg2(Runtime.CallingScriptHash);
    return false;
}
```

The risk is increased if the `Owner` mints a token without an owner, since no validation is performed on this argument during the `mint`. In the case of establishing an empty owner, any user could claim said NFT as theirs simply by making the transfer.



```
public static UInt256 Mint(UInt160 owner, string name, string description, string image)
{
    if (!ValidateContractOwner()) throw new Exception("Only the contract owner can mint tokens");

    // generate new token ID
    StorageContext context = Storage.CurrentContext;
    byte[] key = new byte[] { Prefix_TokenId };
    var id = (BigInteger)Storage.Get(context, key);
    Storage.Put(context, key, id + 1);

    var tokenIdString = nameof(NeoContributorToken) + id;
    var tokenId = (UInt256)CryptoLib.Sha256(tokenIdString);

    var tokenState = new NeoContributorToken.TokenState
    {
        Owner = owner,
        Name = name,
        Description = description,
        Image = image,
    };

    StorageMap tokenMap = new(Storage.CurrentContext, Prefix_Token);
    tokenMap[tokenId] = StdLib.Serialize(tokenState);
    UpdateBalance(tokenState.Owner, tokenId, +1);
    UpdateTotalSupply(+1);
    PostTransfer(null, tokenState.Owner, tokenId, null);

    return tokenId;
}
```

## Source Code References

- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L110-L116
- https://github.com/OnBlockIO/GhostMarketContractN3Exchange/blob/fa7296a84eb57f261d691520e89aec7cedb1d9e8/token-contract/NeoContributorToken.cs#L144

# Annexes

## Annex A – Vulnerabilities Severity

Red4Sec determines the vulnerabilities severity in the following levels of risk according to the impact level defined by CVSS v3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST):

### Vulnerability Severity

The risk classification has been made on the following 5 value scale:

| Severity | Description |
|---|---|
| **Critical** | Vulnerabilities that possess the highest impact over the systems, services and/or sensitive information. The existence of these vulnerabilities is dangerous and should be fixed as soon as possible. |
| **High** | Vulnerabilities that could compromise severely compromise the service or the information it manages even if the vulnerability requires expertise to be exploited. |
| **Medium** | Vulnerabilities that on their own can have a limited impact and/or that combined with other vulnerabilities could have a greater impact. |
| **Low** | These vulnerabilities do not suppose a real risk for the systems. Also includes vulnerabilities which are extremely hard to exploit or whose impact on the service is low. |
| **Informative** | It covers various characteristics, information or behaviours that can be considered as inappropriate, without being considered as vulnerabilities by themselves. |

RED4SEC

*Invest in Security, invest in your future*