Security Audit

Report

12/05/2022

GrantShares

RED4SEC

Content

Introduction

GrantShares is a decentralized and transparent grants program for the Neo ecosystem developed by AxLabs, a Swiss tech company specialized in software and infrastructure engineering for blockchain space, open protocols, and decentralized solutions.

GrantShares aims to be a more agile and diverse extension in comparison to other Neo grant programs by involving key community members in the decision process.

As requested by **AxLabs** 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 **GrantShares** project.

The report includes specifics of the audit for all the existing vulnerabilities of GrantShares. The performed analysis shows that the Smart Contract does contain high risk vulnerabilities. Also, 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, nor as guarantee on the operation or viability of the implemented financial product.

Red4Sec makes full effort and applies every resource available for each audit, however it does not warrant the function, nor the safety of the project and it cannot be deemed a sufficient assessment of the code's utility and safety, bug-free status, or any other declarations of the project. Additionally, Red4Sec makes no security assessments or judgments about the underlying business strategy, or the individuals involved in the project.

Blockchain technology and cryptographic assets come with its own new risks and challenges, where the ecosystem, platform, its programming language, and other software related to said technology can have vulnerabilities that could lead to exploits. As a result, the audit cannot guarantee the explicit security of the audited projects.

The audit reports can be used to improve the code quality of smart contracts, to help limit the vectors of attack and to lower the high level of risks associated with utilizing new and continually changing technologies such as cryptographic tokens and blockchain, but they are unable to detect any future security concerns with the related technologies.

Scope

AxLabs 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/AxLabs/grantshares-contracts
  - branch: `master`
  - commit: `d7c20c971bb686e17a1da13cab9250436327b627`
- Last Review on 12/05/2022.
  - branch: `master`
  - commit: `860f011ea9be1793acc3aaf252cd374260512cc4`

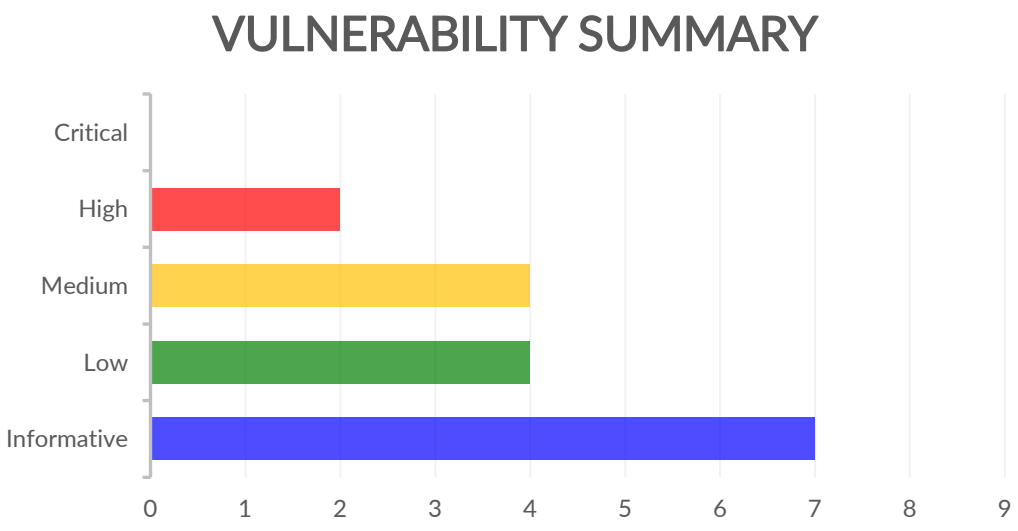
Executive Summary

As requested by **AxLabs**, the company Red4Sec Cybersecurity has been asked to perform a security audit against the **GrantShares**, 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: **03/14/2022** and **04/04/2022** and the final reviews were performed between the following dates: **27/04/2022** and **12/05/2022.**

Once the analysis of the technical aspects of the environment was completed, the performed analysis showed that the audited source code contained important issues that were mitigated as soon as possible.

During the analysis, a total of **17 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, **12/05/2022**, the general conclusion resulting from the conducted audit, is that the **Grantshares is secure** and does not present known vulnerabilities that could compromise the security of the project.

An elaborated action plan to guarantee the resolution of the detected vulnerabilities was promptly and effectively implemented by the AxLabs team. Prioritizing the vulnerabilities of greater risk and they did not exceed the maximum recommended resolution times.

The general conclusions of the performed audit are:

- High-risk vulnerabilities were detected during the security audit. These vulnerabilities were of great risk for the AxLabs project and **have been properly corrected**.
- **Low impact issues** were detected and classified only as informative, but they will continue to help improve the security and quality of its developments. AxLab has decided to correct them as well in order to improve the security and quality of the development.
- Certain methods **do not make the necessary input checks** in order to guarantee the integrity and expected arguments format.
- **Most of the proposed recommendations**, considered necessary by Red4Sec in order to improve the security of the project **were applied**.
- It is important to highlight that both, the new N3 blockchain and the smart contract compiler `neow3j` are new technologies, which are in constant development and have less than a year functioning. For this reason, they are prone to new bugs and breaking changes. Therefore, this audit is unable to detect any future security concerns with neo smart contracts.

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**
GS-01	Denial of Service in Neo Candidate selection	**High**	**Fixed**
GS-02	Use of ASSERT instead of THROW	**High**	**Fixed**
GS-03	Vulnerable to Proposal Flooding	**Medium**	**Assumed**
GS-04	Lack of Inputs Validation in Treasury Contract	**Medium**	**Fixed**
GS-05	Lack of Inputs Validation in Governor Contract	**Medium**	**Fixed**
GS-06	Incomplete OnNEP17Payment logic	**Medium**	**Fixed**
GS-07	Insecure Multisig Threshold	**Low**	**Fixed**
GS-08	Wrong Pausable Logic	**Low**	**Fixed**
GS-09	Denial of Service in the Query Methods	**Low**	**Assumed**
GS-10	Improvable Intents	**Low**	**Fixed**
GS-11	Abuse of discussionUrl argument	**Informative**	**Assumed**
GS-12	Safe Storage Access	**Informative**	**Fixed**
GS-13	Missing NEF Information	**Informative**	**Fixed**
GS-14	Improper Call Rights	**Informative**	**Fixed**
GS-15	GAS Optimization	**Informative**	**Partially Fixed**
GS-16	Unify storage prefixes	**Informative**	**Assumed**
GS-17	Open to-do	**Informative**	**Fixed**

Vulnerability details

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

- Category
- Active
- Risk
- Description
- Recommendations

Denial of Service in Neo Candidate selection

Identifier	Category	Risk	State
GS-01	DoS with Failed Call	**High**	**Fixed**

The candidate selection method for the voting of Neo's governance is susceptible to a denial of service.

The contract includes the logic necessary to vote in the governance of the neo blockchain, thus receive higher rewards for the balance accumulated in the contract.

The `getCommitteeMemberWithLeastVotes` method queries the native contract of `NeoToken.getCandidates()` without considering that neo's virtual machine has a limitation of *2048* elements in the return of the operations, this limitation implies that when more than *1024* candidates are registered for the governance it will result in a denial of service.

```
private static ECPoint getCommitteeMemberWithLeastVotes() {
    NeoToken.Candidate[] candidates = NeoToken.getCandidates();
    List<ECPoint> committee = new List<>(NeoToken.getCommittee());
```

The problem increases because this method is called during the receipt of funds in the treasury, so a failure in the `voteCommitteeMemberWithLeastVotes` function will prevent from receiving and storing funds, which is the main function of the contract.

```
if (Runtime.getCallingScriptHash() == NeoToken.getHash()) {
    voteCommitteeMemberWithLeastVotes();
}
```

References

I. src/neo/SmartContract/Native/NeoToken.cs#L340

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L154

Remediation

From Red4Sec, we believe the most fitting solution is to eliminate the automatic vote and reduce it to manual voting; theoretically the core is in charge of executing it, only during the changes of balance.

- src/neo/SmartContract/Native/NeoToken.cs#L73

Fixes Review

The issue has been addressed in the commit:

- ee90461302e42d398043b4d180faea484f55a02e.

Use of ASSERT instead of THROW

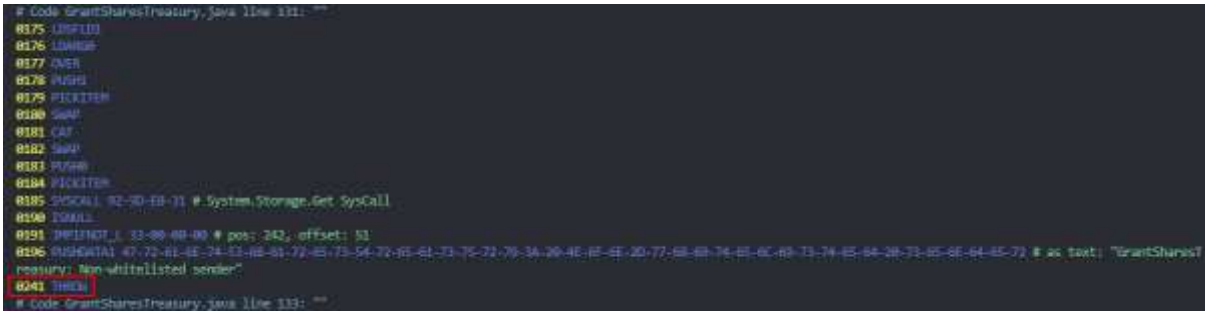
Identifier	Category	Risk	State
GS-02	Business Logic Errors	**High**	**Fixed**

The Java compiler currently used does not properly translate the `assert` call, since it translates it using the `THROW` opcode instead of the `ASSERT`. It is important to mention that the `ASSERT` and `ABORT` opcodes are approximately 9 times less expensive than the `THROW` opcode.

`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 to only certain initial verifications in the methods when none of the values have been updated and even in these cases it is more effective to use `ASSERT`.

Below, it can be seen that the shown statement uses `assert`. However, it has been translated as `THROW`.

```
assert funders.get(sender.toByteString()) != null :
        "GrantSharesTreasury: Non-whitelisted sender";
```



`throw` it is only recommended to be used within a `try` sentence in the same contract, otherwise there is a possibility for it to become captured by an external contract and it will not necessarily end the execution of the transaction.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L131

Remediation

The AxLabs team mentioned that a new version of the neow3j compiler will be released with the proposed fix and the contracts will be adapted to said version. As described in:

I. https://github.com/neow3j/neow3j/issues/744

The problem is considered to be quite critical by the Red4Sec team since the logics are completely different, in one of the logics the execution is automatically cut and in the second logic, the exception is allowed to be captured.

Fixes Review

This vulnerability has been submitted to the Neo blockchain development team and they have applied a fix to the core of the blockchain to fix it natively.

- https://github.com/neo-project/neo/pull/2729

Vulnerable to Proposal Flooding

Identifier	Category	Risk	State
GS-03	Business Logic Errors	**Medium**	**Assumed**

The `createProposal` method does not limit the number or type of proposals created. There is no limitation that prevents a contract from calling the `createProposal` method multiple times. So, an unlimited number of proposals can be created for a very low cost. Although this does not currently trigger a denial of service of the contract, this vector of attack could be exploitable in the future if is not controlled.

Flooding the contract with invalid proposals can cause failures in the dApss that display information of the proposals and can lead to malicious errors during the voting.

In the following image we can see that the `createProposal` method does not set any limits on the `proposer`. Examples of said limits could be: limited number of proposals, minimum balance, etc...

```
public static int createProposal(Hash160 proposer, Intent[] intents, String discussionUrl,
        int linkedProposal, int acceptanceRate, int quorum) {

    assert checkWitness(proposer) : "GrantSharesGov: Not authorised";
    assert acceptanceRate >= parameters.getInt(MIN_ACCEPTANCE_RATE_KEY)
            : "GrantSharesGov: Acceptance rate not allowed";
    assert quorum >= parameters.getInt(MIN_QUORUM_KEY)
            : "GrantSharesGov: Quorum not allowed";
    assert linkedProposal < 0 || proposals.get(linkedProposal) != null
            : "GrantSharesGov: Linked proposal doesn't exist";

    int id = Storage.getInt(ctx, PROPOSALS_COUNT_KEY);
    proposals.put(id, serialize(new Proposal(id)));
    proposalData.put(id, serialize(new ProposalData(proposer, linkedProposal, acceptanceRate,
            quorum, intents, discussionUrl)));
    proposalVotes.put(id, serialize(new ProposalVotes()));
    Storage.put(ctx, PROPOSALS_COUNT_KEY, id + 1);

    // An event can take max 1024 bytes state data. Thus, we're not passing the discussionUrl
    created.fire(id, proposer, acceptanceRate, quorum);
    return id;
}
```

Recommendations

It is advisable to verify the number of proposals created, how many are active at that moment or the time elapsed between proposals in order to avoid creating unlimited proposals.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L307

Fixes Review

The AxLabs team considers it unnecessary to add any control to the smart contract since; "AxLabs restricts the number of new proposals per user time interval in the backend/frontend of the dApp. Proposal creation directly on the GrantShares contract, i.e., without using the dApp's *website, is not restricted (besides transaction GAS costs). However, the dApp website is not prone to a DoS attack through proposal creation. Proposal lookups are all done in a hash table fashion which prevents overload of the dApp through this attack vector. The smart contract itself does also not suffer from the creation of a lot of proposals. Each proposal can be fetched with it's ID independent of the amount of proposals in the contract's storage."*

Lack of Inputs Validation in Treasury Contract

Identifier	Category	Risk	State
GS-04	Improper Input Validation	**Medium**	**Fixed**

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

The general execution of the contract is not currently checking the inputs nor the integrity of most of the `UInt160` types, and in various cases, integers with negative value. This bad practice 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`.

```
/**
 * Checks if the given object is a valid Hash160, i.e., if it is either a ByteString or Buffer
 * and 20 bytes long.
 *
 * @param data The object to check.
 * @return true if the given object is a valid Hash160. False, otherwise.
 */
@Instruction(opcode = OpCode.DUP)
@Instruction(opcode = OpCode.DUP)
@Instruction(opcode = OpCode.ISTYPE, operand = StackItemType.BYTE_STRING_CODE)
@Instruction(opcode = OpCode.SWAP)
@Instruction(opcode = OpCode.ISTYPE, operand = StackItemType.BUFFER_CODE)
@Instruction(opcode = OpCode.BOOLOR)
@Instruction(opcode = OpCode.SWAP)
@Instruction(opcode = OpCode.SIZE)
@Instruction(opcode = OpCode.PUSHINT8, operand = LENGTH) // 20 bytes expected array size
@Instruction(opcode = OpCode.NUMEQUAL)
@Instruction(opcode = OpCode.BOOLAND)
public static native boolean isValid(Object data);
```

```
/**
 * Checks if this {@code Hash160} is zero-valued.
 *
 * @return true if this {@code Hash160} is zero-valued. False, otherwise.
 */
@Instruction(opcode = OpCode.PUSH0)
@Instruction(opcode = OpCode.NUMEQUAL)
public native boolean 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.

References

I. https://javadoc.io/doc/io.neow3j/devpack/latest/io/neow3j/devpack/Hash160.html
II. devpack/src/main/java/io/neow3j/devpack/Hash160.java#L60
III. devpack/src/main/java/io/neow3j/devpack/Hash160.java#L80

Source Code References

Check `IsValid && !IsZero` additionally, it is recommended to verify that it is a valid contract and `GrantSharesGov`.

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L91

Check `accountHash.IsValid && !accountHash.IsZero`

II. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L96

Check `IsValid && !IsZero` and verify that `maxes` variable has a positive value.

III. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L106

Check if the value is positive

IV. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L110

It is not verified that the `value` variable is not greater than zero or if it is less than or equal to the number of funders.

V. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L293

Check `token.IsValid && !token.IsZero`. The `maxFundingAmount`variable is not verified to be positive.

VI. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L344

Check `account.IsValid && !account.IsZero`. Also, the `publicKeys` array size should be greater than zero.

VII. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L312

Fixes Review

The issue has been addressed in the commit:

- 4648d81588809ab9d79f62a8b3fa44e994266c9f

Lack of Inputs Validation in Governor Contract

Identifier	Category	Risk	State
GS-05	Improper Input Validation	**Medium**	**Fixed**

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

The general execution of the contract is not checking the inputs nor the integrity of most of the `UInt160` types, and in various cases, integers with negative value. This bad practice 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`.

```
/**
 * Checks if the given object is a valid Hash160, i.e., if it is either a ByteString or Buffer
 * and 20 bytes long.
 *
 * @param data The object to check.
 * @return true if the given object is a valid Hash160. False, otherwise.
 */
@Instruction(opcode = OpCode.DUP)
@Instruction(opcode = OpCode.DUP)
@Instruction(opcode = OpCode.ISTYPE, operand = StackItemType.BYTE_STRING_CODE)
@Instruction(opcode = OpCode.SWAP)
@Instruction(opcode = OpCode.ISTYPE, operand = StackItemType.BUFFER_CODE)
@Instruction(opcode = OpCode.BOOLOR)
@Instruction(opcode = OpCode.SWAP)
@Instruction(opcode = OpCode.SIZE)
@Instruction(opcode = OpCode.PUSHINT8, operand = LENGTH) // 20 bytes expected array size
@Instruction(opcode = OpCode.NUMEQUAL)
@Instruction(opcode = OpCode.BOOLAND)
public static native boolean isValid(Object data);
```

```
/**
 * Checks if this {@code Hash160} is zero-valued.
 *
 * @return true if this {@code Hash160} is zero-valued. False, otherwise.
 */
@Instruction(opcode = OpCode.PUSH0)
@Instruction(opcode = OpCode.NUMEQUAL)
public native boolean isZero();
```

The same case applies to `ECPoint`, since this class also contains the `isValid` method, therefore, it must be checked.

```
/**
 * Checks if the given object is a valid EC point, i.e., if it is either a ByteString or Buffer
 * and 33 bytes long.
 *
 * @param data The object to check.
 * @return true if the given object is a valid EC point. False, otherwise.
 */
@Instruction(opcode = OpCode.DUP)
@Instruction(opcode = OpCode.DUP)
@Instruction(opcode = OpCode.ISTYPE, operand = StackItemType.BYTE_STRING_CODE)
@Instruction(opcode = OpCode.SWAP)
@Instruction(opcode = OpCode.ISTYPE, operand = StackItemType.BUFFER_CODE)
@Instruction(opcode = OpCode.BOOLOR)
@Instruction(opcode = OpCode.SWAP)
@Instruction(opcode = OpCode.SIZE)
@Instruction(opcode = OpCode.PUSHINT8, operand = LENGTH) // 33 bytes expected array size
@Instruction(opcode = OpCode.NUMEQUAL)
@Instruction(opcode = OpCode.BOOLAND)
public static native boolean isValid(Object data);
```

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.

References

I. https://javadoc.io/doc/io.neow3j/devpack/latest/io/neow3j/devpack/ECPoint.html
II. devpack/src/main/java/io/neow3j/devpack/ECPoint.java#L58

Source Code References

It is not tested that the value of `MULTI_SIG_THRESHOLD_KEY` is previously configured, this value is necessary for the contract to properly work.

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L109

It is not verified that the new member introduced previously exists, which can lead to introducing duplicates.

II. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L112

Check `IsValid`

III. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L113

Check if the value is positive

IV. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L224

The values entered for the variables `acceptanceRate` and `quorum` are not verified, thus the minimum values are verified but not the maximum values allowed, so a proposal might not be executed if the entered values exceed the maximum possible.

V. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L308
VI. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L415-L418

It is not verified that the `paramKey` to modify previously exists, before modifying its value.

VII. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L443

Fixes Review

The issue has been addressed in the commit:

- a3cd2c175bdbf16d37f49b2093b817e6c2debae3

Incomplete OnNEP17Payment logic

Identifier	Category	Risk	State
GS-06	Improper Input Validation	**Medium**	**Fixed**

`The OnNEP17Payment` implementation does not properly perform the desired verifications. For example, if a user mints any token directly to the contract, the action will be allowed.

According to the NEP17 standard:

> "A token contract which creates new tokens MUST trigger a Transfer event with the from address set to null when tokens are created."

For this reason, during the `mint` of a token, the `sender` argument is `NULL`; therefore, the `sender` argument is verified to allow `mint` of the GAS to the contract.

However, it is not verified that the hash of the `sender` is the hash of GAS. So, any token can be minted and the logic allows it.

```
if (sender == null) {
    return; // If the sender is null the transfer is a GAS claim.
}
```

Recommendations

It is convenient to limit said condition exclusively to the contract of GAS.

References

I. https://github.com/neo-project/proposals/blob/master/nep-17.mediawiki#transfer-1

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L126-L130

Fixes Review

The issue has been addressed in the commit:

- 0721b636a36c228647ef500001586aa06ecb7c74

Insecure Multisig Threshold

Identifier	Category	Risk	State
GS-07	Design Weaknesses	**Low**	**Fixed**

The `calcMembersMultiSigAccountThreshold` method does not check for a minimum threshold which allows `calcMembersMultiSigAccount` to generate insecure multisign addresses.

The root of this problem resides in the lack of control of the `MULTI_SIG_THRESHOLD_KEY` value, which should never be less than *1* nor greater than the total number of the committee members. If this value is set to zero, all the methods protected by the multisign are exposed without any protection.

Note that there are several situations that can trigger this condition. First, during `_deploy` it is not verified that this argument has been previously supplied, nor that the value matches the desired range. Second, the `changeParam` method allows to arbitrarily set any value to the `params` of the contract and allows a proposal to alter the values.

Recommendations

- The `calcMembersMultiSigAccountThreshold` method should not return zero.
- The `changeParam` method should always check the format of the arguments before using their value.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L267
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L443

Fixes Review

The issue has been addressed in the commit:

- 0721b636a36c228647ef500001586aa06ecb7c74

Wrong Pausable Logic

Identifier	Category	Risk	State
GS-08	Business Logic Errors	**Low**	**Fixed**

The `GrantSharesGov` contract can be paused by the members of the council, however, there are methods such as `execute`, `vote` or `endorseProposal` that can still be called even if the contract is in a paused state.

The existence of the `pause` method allows members to act in emergency situations and avoid possible problems that have been previously detected. For this reason, it is convenient to apply the requirement non-paused to the methods that the user is going to interact with. In this case we consider that the `execute`, `vote` and `endorseProposal` methods must be limited to a non-paused state.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L516

Fixes Review

The issue has been addressed in the commit:

- 0721b636a36c228647ef500001586aa06ecb7c74

Denial of Service in the Query Methods

Identifier	Category	Risk	State
GS-09	DoS with Failed Call	**Low**	**Assumed**

It has been identified that certain logics of the contract execute loops that can make excessive iterations, which can trigger a Denial of Service (DoS) by GAS exhaustion because it iterates without any limit.

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.

Loops without limits are considered a bad practice in the development of Smart Contracts, since they can trigger a Denial of Service or overly expensive executions, this is the case affecting Network Contract.

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 existing limitation in the virtual machine will therefore limit the maximum number of entries in the `getMembers` and `getWhiteListedTokens` methods.

```
@Safe
public static Map<Hash160, Integer> getWhitelistedTokens() {
    Iterator<Struct<Hash160, Integer>> it = whitelistedTokens.find(RemovePrefix);
    Map<Hash160, Integer> tokens = new Map<>();
    while (it.next()) {
        Struct<Hash160, Integer> i = it.get();
        tokens.put(i.key, i.value);
    }
    return tokens;
}
```

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L256
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L182

Fixes Review

Considering that the criticality of this vulnerability is low risk, since only the members can add whitelisted tokens, the AxLabs team plans to address this issue on future releases of the Smart Contract.

Improvable Intents

Identifier	Category	Risk	State
GS-10	Design Weaknesses	**Low**	**Fixed**

In the verification and control of the actions defined in the proposals, the `Intent` can be improved to increase the coverage.

1. Currently the `Intent` is executed with the `CallFlagsAll`, the `Intent` class should contain the `CallFlags` with which said execution is carried out, so that the executions can be more secure and auditable.

```
for (int i = 0; i < data.intents.length; i++) {
    Intent t = data.intents[i];
    returnVals[i] = Contract.call(t.targetContract, t.method, CallFlags.All, t.params);
}
```

2. It is not verified if the destination is a neo native smart contract, in order to avoid the call to methods such as `Update` or `Destroy`, since for the first case there is already a method designed for its call, but not for the second case.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L307
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L426
III. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L495
IV. src/main/java/com/axlabs/neo/grantshares/Intent.java#L9

Fixes Review

The issue has been addressed in the following commits:

- 13adaa2ed92047fddbde3c4524bef071cb574c06
- 1af6029e7565d310f107f22b900435d73a866e57

Although a property for the `CallFlags` has been included in the `Intent`, they are not used in the `execute` method.

- src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L528

The `containsCallsToContractManagement` method has been created that iterates all the `Intent` in order to know if there is a malicious call. This can be unified in the `areIntentsValid` method using the following code fragment.

```
i.targetContract ==  Hash160.zero() ||
  i.targetContract == ContractManagement.getHash()
```

Abuse of discussionUrl argument

Identifier	Category	Risk	State
GS-11	Improper Input Validation	**Informative**	**Assumed**

During the creation of the proposals in the `createProposal` method, the format of the `discussionUrl` argument is not properly controlled, being a crucial field for decision making, approval and discussion of the proposals.

By allowing the user to arbitrarily establish the `discussionUrl`, a new surface of attack emerges, from discussion url forgery to the possibility of Cross Site Scripting (XSS) attacks over the dApps that display information about our proposals.

```
int id = Storage.getInt(ctx, PROPOSALS_COUNT_KEY);
proposals.put(id, serialize(new Proposal(id)));
proposalData.put(id, serialize(new ProposalData(proposer, linkedProposal, acceptanceRate,
        quorum, intents, discussionUrl)));
proposalVotes.put(id, serialize(new ProposalVotes()));
Storage.put(ctx, PROPOSALS_COUNT_KEY, id + 1);
```

Recommendations

It is convenient that the addresses of the proposals aim to a dapp controlled by the project, and that they are generated based on the id of the proposal; so, the field is not controlled by the user, which will ultimately eliminate this vector of attack.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L321

Fixes Review

The issue has been addressed in the following commits:

- f3e235c8fdccaf5fb89c5031b12666c645eb459f
- bfe16665fb65c2ac2ab7ddf84f69b599ac98b220

The issue has been assumed by the AxLabs team as the applied changes do not resolve the issue. The AxLabs team considers it unnecessary to add any control to the smart contract since; "… The dApp will make sure that the URI is displayed properly. If users interact directly on the smart contract level, they have the responsibility to properly validate/check whether the URI is malicious or not. Moreover, the parameter `discussionUrl` has been renamed to `offchainUri` in order to be more explicit and broad for future GrantShares features."

Safe Storage Access

Identifier	Category	Risk	State
GS-12	Bad Coding Practices	**Informative**	**Fixed**

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

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

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L271
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L282
III. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L202
IV. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L212
V. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L225
VI. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L242
VII. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L266
VIII. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L318

Fixes Review

The issue has been addressed in the following commit:

- 050b76eac5d38f21a671b0a0be2ab64e0fd768f0

Missing NEF Information

Identifier	Category	Risk	State
GS-13	Bad Coding Practices	**Informative**	**Fixed**

The Red4Sec team has detected that the `NEF` file is not being generated with the `source` field.

The `source` field indicates the url of the contract's source code, publishing the contract's source code increases the transparency and decentralization of the project, so it is always beneficial to do so.

```
@SuppressWarnings("unchecked")
@Permission(contract = "0xfffdc93764dbaddd97c48f252a53ea4643faa3fd", methods = "update") // ContractManagement
@Permission(contract = "0xef4073a0f2b305a38ec4050e4d3d28bc40ea63f5", methods = "vote") // NeoToken
@Permission(contract = "*", methods = "transfer")
@DisplayName("GrantSharesTreasury")
@ManifestExtra(key = "author", value = "AxLabs")
```

The NEP-16 standard reserves a field in the NEF for this purpose.

References

I. https://github.com/neo-project/proposals/blob/master/nep-16.mediawiki
II. src/neo/SmartContract/NefFile.cs#L57

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L37
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L38

Fixes Review

GrantShares Team Notes

The issue has been addressed in the following commit:

- 67dd64408db5216a9be9451966b34856119a7829

The client has included the attribute to the `Manifest` and is aware that it must be updated before deploying.

Improper Call Rights

Identifier	Category	Risk	State
GS-14	Design Weaknesses	**Informative**	**Fixed**

In the `drain` method of the `GrantSharesTreasury` contract, a call is made to the `balanceOf` method to obtain the current balance, to subsequently invoke the `transfer` method. However, the `CallFlags` chosen may not be compatible with every token.

In this case, the `CallFlags` defined is not wrong, however, there could be a case of incompatibility if the destination contract uses a proxy or uses calls between contracts. The `ReadStates` state does not allow calls between contracts, however, using `ReadOnly` will provide the same security and allow said possibility.

```
while (it.next()) {
    Hash160 token = new Hash160(it.get());
    int balance = (int) Contract.call(token, "balanceOf", CallFlags.ReadStates,
            new Object[]{selfHash});
    if (balance > 0) {
        Object[] params = new Object[]{selfHash, fundersMultiAddress, balance,
                new Object[]{}};
        Contract.call(token, "transfer", CallFlags.All, params);
    }
}
```

Recommendations

Evaluate the change of scope from ReadStates to ReadOnly in case of wanting to make it compatible with proxy.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L406

Fixes Review

The issue has been addressed in the following commit:

- a9e12717228ec10a4dedcb2cf44b20415e3268dc

GAS Optimization

Identifier	Category	Risk	State
GS-15	Bad Coding Practices	**Informative**	**Partially Fixed**

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.

Since Java does not have constants[1], but only final static, it is convenient to use private methods that return the contexts of the storage, in this way, they will not be initialized unless they are going to be used by the logic of the execution.

The AxLabs team is currently studying possible solutions for this issue:

- https://github.com/AxLabs/grantshares-contracts/issues/21

Logic Optimization

The logic to store the flag that establishes the paused status of the `GrantSharesGov` contract requires an extra process when parsing a hexadecimal value. Using `getBoolean` will obtain the same result at a lower cost.

[1] https://www.javatpoint.com/java-constant

Unnecessary logic

During the execution of the `onNep17Payment` method, a vote is produced for the selected candidate as long as it is the Neo token, this logic is already contemplated in the Core of neo, and when the balance of an account is modified, its vote is altered accordingly.

```
if (Runtime.getCallingScriptHash() == NeoToken.getHash()) {
    voteCommitteeMemberWithLeastVotes();
}
```

As shown below, the balance is increased during the reception of the neo token to the previously voted candidate.

References

I. https://www.javatpoint.com/java-constant
II. src/neo/SmartContract/Native/NeoToken.cs#L73

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L137
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L242

Fixes Review

The issue has been addressed in the following commit:

- daca337bc28eaf5864486d4d5ac213b77194f1f9
- ee90461302e42d398043b4d180faea484f55a02e

Unify storage prefixes

Identifier	Category	Risk	State
GS-16	Bad Coding Practices	**Informative**	**Assumed**

Different `storage` prefixes are used throughout the contract, mixing `String` prefixes and `Byte` prefixes. It is convenient to use the practice of unifying all the prefixes to the same type and of the same length to avoid possible collisions and injections in the use of storage keys.

As can be seen below, a different way of initializing the `StorageMap` has been used between the different contracts, on certain occasions a byte has been used, which is the most optimal way, and on other occasions a string has been used.

```
static final StorageMap proposals = new StorageMap(ctx, 1); // [int id: Proposal proposal]
static final StorageMap proposalData = new StorageMap(ctx, 2); // [int id: ProposalData proposalData]
static final StorageMap proposalVotes = new StorageMap(ctx, 3); // [int id: ProposalVotes proposalVotes]
static final StorageMap parameters = new StorageMap(ctx, 4); // [String param_key: int param_value ]
```

The use of prefixes in string mode is discouraged, and even more so if they are text strings of different sizes amongst them, since this will cause the keys to have a different sizes, if the code is vulnerable, there is a possibility of collisions and/or storage injections, however not in this particular case.

```
static final String FUNDERS_PREFIX = "funders";
static final String WHITELISTED_TOKENS_PREFIX = "whitelistedTokens";
static final String MULTI_SIG_THRESHOLD_KEY = "threshold";

static final StorageContext ctx = Storage.getStorageContext();
static final StorageMap funders = new StorageMap(ctx, FUNDERS_PREFIX); // [hash, List<ECPoint>]
static final StorageMap whitelistedTokens = new StorageMap(ctx, WHITELISTED_TOKENS_PREFIX); // [hash, max_amount]
```

During the review of the accesses to the storage of the audited smart contracts, no incident related to storage injections or key collisions has been found, however, from Red4Sec we discourage the use of this initialization method of the `StorageMap`.

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesTreasury.java#L43-L44

Fixes Review

The issue has been assumed by the AxLabs team and they are currently studying possible solutions for this issue:

- https://github.com/AxLabs/grantshares-contracts/issues/22

Open to-do

Identifier	Category	Risk	State
GS-17	Bad Coding Practices	**Informative**	**Fixed**

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

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.

```
// TODO: `currentIndex` has to be replaced with `getTime`
int time = currentIndex();
assert time >= proposal.reviewEnd && time < proposal.votingEnd
        : "GrantSharesGov: Proposal not active";
```

Source Code References

I. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L346
II. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L373
III. src/main/java/com/axlabs/neo/grantshares/GrantSharesGov.java#L408

Fixes Review

The issue has been addressed in the following commit:

- a070a8a89dcc2051bf784692c15334b2ecbe7c1f

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.

Invest in Security, invest in your future