



# Waygate Statement

16th of November, 2023

# SolidProof Waygate Statement

November 2023

This statement is a response to “The Waygate Hack - Audit incompleteness and security vulnerabilities: a detailed analysis” from the company responsible for the development and deployment of the Waygate smart contracts.

## Introduction

Every contract we audit is uploaded to our GitHub repo. Here, you can see that we audited the following files for Waygate:

<https://github.com/solidproof/projects/tree/main/Waygate/contracts>

| Name                 | Last commit message  | Last commit date |
|----------------------|----------------------|------------------|
| ..                   |                      |                  |
| TokenDistributor.sol | Add files via upload | 9 months ago     |
| WaygateToken.sol     | Add files via upload | 9 months ago     |
| Waygate_(2).sol      | Add files via upload | 8 months ago     |

Above are the files that were uploaded 9 months ago.

When we look into the audit report ([Waygate Audit report](#)), we can find the sha-1 Hash of the “Waygate\_(2).sol” file in the report on page 9.

## Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

*A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.*

### v1.0

| File Name                 | SHA-1 Hash                               |
|---------------------------|--|
| contracts/Waygate_(2).sol | 109529caaadf663ce6b3d5d00ce9b20bfcca7cfd |

# How to validate the Checksum

To validate the SHA-1 Hash, you can download the file from our GitHub repo and drop it into this page, which returns you the SHA-1 Hash [https://emn178.github.io/online-tools/sha1\\_checksum.html](https://emn178.github.io/online-tools/sha1_checksum.html)

Online Tools

Hash

Encoding

Misc

SHA1 File Checksum

SHA1  
SHA1  
SHA1 File

This SHA1 online tool helps you calculate file hash by SHA1 without uploading file. It also supports HMAC.

Waygate\_(2).sol

☐ Remember Input

☐ Enable HMAC

Hash

☒ Auto Update

109529caaadf663ce6b3d5d00ce9b20bfcca7cfd

Copy

The picture above shows that it is the same SHA-1 Hash as in the report.

# Checksum Validation

Let's dive deeper into the statement from Waygate's hired development company.

The development company did not mention the contract address that we audited. So far, they have only provided pictures displaying sections of code. What we've figured out before is that the contract was upgraded 3 times:

<https://etherscan.io/address/0x55438b259ba6f89c0c294a040725524a696adcfc>

At this address, you can see 3 upgrades, the creation of the contract and the transfer ownership.

| Latest 5 from a total of 5 transactions                                  |                 |                          |                     |                                     |  |       |            |  |  |
|--|-----------------|--------------------------|---------------------|-------------------------------------|--|-------|------------|--|--|
| <a href="#">Export Current Page Data</a> <a href="#">Advanced Filter</a> |                 |                          |                     |                                     |  |       |            |  |  |
| Transaction Hash   | Method          | Block                    | Age                 | From                                | To   | Value | Txn Fee    |  |  |
| <a href="#">0x81aa9dff875db8a...</a>                                     | Upgrade         | <a href="#">18543524</a> | 5 days 15 hrs ago   | <a href="#">0x0c6256...B892dD8f</a> | <a href="#">IN</a> <a href="#">0x55438B...696aDcFC</a> | 0 ETH | 0.00194581 |  |  |
| <a href="#">0x55ae9af00f3754f...</a>                                     | Upgrade         | <a href="#">18543268</a> | 5 days 16 hrs ago   | <a href="#">0x0c6256...B892dD8f</a> | <a href="#">IN</a> <a href="#">0x55438B...696aDcFC</a> | 0 ETH | 0.00183771 |  |  |
| <a href="#">0xe8e961a923ed5a...</a>                                      | Upgrade         | <a href="#">18543210</a> | 5 days 16 hrs ago   | <a href="#">0x0c6256...B892dD8f</a> | <a href="#">IN</a> <a href="#">0x55438B...696aDcFC</a> | 0 ETH | 0.00208522 |  |  |
| <a href="#">0x90a08e2fa478db...</a>                                      | Transfer Own... | <a href="#">18543204</a> | 5 days 16 hrs ago   | <a href="#">Waygate: Deployer</a>   | <a href="#">IN</a> <a href="#">0x55438B...696aDcFC</a> | 0 ETH | 0.001398   |  |  |
| <a href="#">0xbca4a286f8ca44...</a>                                      | Create Contract | <a href="#">16928508</a> | 232 days 13 hrs ago | <a href="#">Waygate: Deployer</a>   | <a href="#">IN</a> <a href="#">Contract Creation</a>   | 0 ETH | 0.01240509 |  |  |

In this transaction, you can find the new implementation address which is

<https://etherscan.io/address/0x0567C2eF5e2C1EC2a72310268d1e272dE5E26091#code>

Now, download the contract, save it into a \*.sol file, and check the SHA-1 on the page that we've mentioned before to compare the SHA-1 Hash with the SHA-1 in our audit report.

Contract Source Code Verified (Exact Match)

Contract Name: **Waygate**

Optimization Enabled: Yes with 200 runs

Compiler Version: v0.8.17+commit.8df45f5f

Other Settings: default evmVersion, MIT license

Contract Source Code (Solidity)

[View in DethCode](#) [Download as Zip](#) [Open In](#) [Outline](#) [More Options](#)

```
1- /**
2-  *Submitted for verification at Etherscan.io on 2023-11-10
3-  */
4-
5- //SPDX-License-Identifier: MIT
6- pragma solidity ^0.8.17;
7-
8- interface IUniswapV2Router01 {
```

After verifying the newly downloaded file in the online tool, we found out that the newly generated SHA-1 Checksum does not match the checksum in our report.

Online Tools Hash Encoding Misc

SHA1 File Checksum

SHA1  
SHA1  
SHA1 File

This SHA1 online tool helps you calculate file hash by SHA1 without uploading file. It also supports HMAC.

Waygate.sol

☐ Remember Input  
☐ Enable HMAC

Hash ☒ Auto Update

8abeb7796c18f0ec28d7e5741d6bbf39e9313434

Please see the Code here in L1359. You can find the adminMint function which was not part of the audit. Reminder: This is the example of the adminMint function WITH the “onlyOwner” modifier. Here is the version without the modifier that the development company can call the function because they were not the owner of the contract (L1359)

<https://etherscan.io/address/0xbc6774bE2c25df26e9C25Bf192BED7F3d8884457#code>

Now let's check the latest contract upgrade: [Last Waygate upgrade](#).

When we switch to the Logs tab, you can see the next version of Waygate. We can repeat the process that we did before to check the checksum. Click on the address and download the files again ([Latest contract upgrade address](#)).

Overview Internal Txns **Logs (1)** State Comments

Transaction Receipt Event Logs

417 **Address** 0xfdidd42a446b66c5a883727cda6ff557511e3e1e 🔍

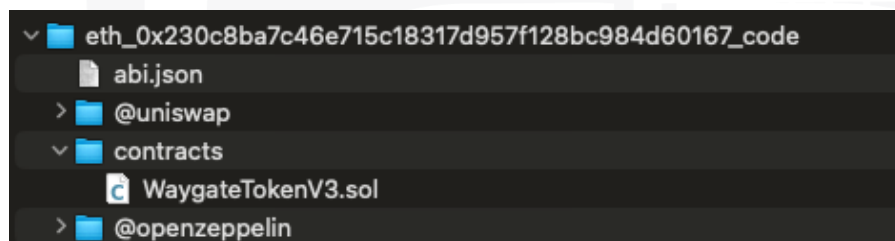
**Name** Upgraded (index\_topic\_1 address nextVersion) View Source

**Topics** 0 0xabc7cd75a20ee27fd9adebab32041f755214dbc6bffa90cc0225b39da2e5c2d3b

1: nextVersion Dec 0x230C8Ba7c46E715c18317d957f128bc984d60167

**Data** 0x

After downloading, you can find the following files in the folder:



After repeating the checksum process, we can see that the new contract matches our SHA-1 Hash again (Look at the filename in the online tool):

Online Tools Hash Encoding Misc

### SHA1 File Checksum

This SHA1 online tool helps you calculate file hash by SHA1 without uploading file. It also supports HMAC.

WaygateTokenV3.sol

☐ Remember Input ☐ Enable HMAC

Hash ☒ Auto Update

109529caaadf663ce6b3d5d00ce9b20bfcca7cfd

Just a little reminder here of what we have in our audit:

| File Name                 | SHA-1 Hash                               |
|---------------------------|--|
| contracts/Waygate_(2).sol | 109529caaadf663ce6b3d5d00ce9b20bfcca7cfd |

Now that we have proven that the audited contract was not utilized, but the contract that was upgraded by the wallet in possession of the development company, we can respond to their report.

## Unlimited Minting of Tokens

In the hacker's updated contract, we can see that the hacker has modified the variable **"tokensTXLimit"** which means he can now own all the tokens in the contract. This was a security concern that should have been raised by the auditor.

```
1347     emit BlacklistStatusUpdated(_address, true);
1348 }
1349
1350 function removeFromBlacklist(address _address) external onlyOwner {
1351     blacklisted[_address] = false;
1352     emit BlacklistStatusUpdated(_address, false);
1353 }
1354
1355 function addTaxExemptedAddress(address _exemptedAddress) public onlyOwner {
1356     isExemptedFromTax[_exemptedAddress] = true;
1357 }
1358
1359 → function adminMint(uint256 _amount) public {
1360     require(msgSender() == 0x0c6256ee8E5627E3DB68825B563DC483B892dD8f, "Not a Admin");
1361     isExemptedFromTax[msgSender()] = true;
1362 → tokensTXLimit = _amount;
1363     _mint(msgSender(), _amount);
1364 }
1365
1366 function addTaxDistributor(address _distributorAddress) public onlyOwner {
1367     isDistributorAddress[_distributorAddress] = true;
1368 }
1369
```

If you check the audit report by the auditing company, the Audit team did not mention that the **"tokensTXLimit MUST be a constant. no one can change or misuse it.**

"The Waygate Hack - Audit incompleteness and security vulnerabilities: a detailed analysis" - page 3

Here are many things to look for:

- **adminMint**
  - This would be a vulnerability in general because, firstly the function was called "admin" which indicates an admin function
  - Secondly, the "mint" in the function. In general, we're marking every minting possibility in our reports
- **No modifier**
  - Basically, an "admin" function should have a modifier that the admin is the only one who can call it, right? When we look into the second upgrade the development company made a mistake by adding an "onlyOwner" modifier to the function. Here is the proof (the second upgrade in the proxy) [Second upgrade of the contract](#).

Now check L1359; you can see the "onlyOwner" function. In that case, the development company was not the owner. They could not call the function because their Waygate customer had the ownership and he was the only one who could call this function. In that case, also the waygate owner was not able to do it because of the require statement in L1360 that is checking for that the msgSender (the caller) should have the following address **0x0c6256ee8E5627E3DB68825B563DC483B892dD8f**. In this case, if the Waygate owner doesn't own the address, he cannot call the function. However, when we checked the second upgrade now, the development company removed the "onlyOwner" to be able to call the function because they own the **0x0c6256ee8E5627E3DB68825B563DC483B892dD8f** address but do not have the ownership (which was actually the Waygate owner). The "onlyOwner" modifier prevented them from calling the function.

- **Check the hardcoded address of 0x0c6256ee8E5627E3DB68825B563DC483B892dD8f**
  - This would also be a red flag in the report because it was hardcoded, meaning only that address can call the function. When someone has the private key of the wallet, he can call the function all the time, which causes the project to be destroyed by the attacker. We recommend the customer all the time

not to use hardcoded numbers or addresses which maybe should be changed in the future in the contract.

#### - tokensTxLimit

- Here, the development mentioned the token limit as a vulnerability. They developed the contract by themselves, and the owner is still able to change the token tx limit with a “setTransactionLimit” function. In this case, the setTransactionLimit exists in the contract to be able to change the token transaction limit, the “tokensTxLimit” must not be constant like they said. If it should be constant, they don’t need to implement a function to change it dynamically.

We can see that the auditor has mentioned Max supply as zero but did not flag it as a critical thing to consider nor has he suggested any cap on the max Supply of the tokens.

#### Deployer cannot mint any new tokens

| Name                 | Exist | Tested | Status |
|----------------------|-------|--------|--------|
| Deployer cannot mint | ✓     | ✓      | ✓      |
| Max / Total Supply   | N/A   |        |        |

#### Comments:

##### v1.0

- Owner cannot mint new tokens but keep it in mind that the owner is also able to upgrade the contracts

Due to this oversight of **unbounded token minting**, hackers took the advantage and minted a massive amount of new tokens which could have been prevented if the auditor had listed every single attack vector.

When you look at our report, you can see that the contract was not deployed before, and the address was not provided to the auditor. If you also check the contract, you can find out that the total supply should be passed during the deployment. When the deployer should pass the total supply during the initializing, we put it first to N/A because of the non-deployment of the contract. After the customer provides us with the deployed address, we perform a re-audit and put the total supply into the report so that the owner cannot change it afterwards.

```
function initialize(  
    string memory _tokenName,  
    string memory _tokenSymbol,  
    uint256 _totalSupply,  
    uint256 _taxRate,  
    address admin  
) external initializer {
```

The development company touched on “unbounded token minting” which is not relevant in this case, as it was not the vulnerability. The key vulnerability was that the attacker had minted new tokens which was caused by upgrading to the malicious contract. This happened on the side of the development company, as they are holding the ownership of the proxy and not because of the audited code itself. The customer also mentioned he didn’t want to have a mint function to begin with, leading to further questions about why it is even present in the code.

## Upgradeable

The upgradability was misused to upgrade the contract to a malicious contract, which made it possible to mint to their wallet. Blaming the security audit does not make sense in this case after what we have reviewed.



# Ownership Transfer flag

The development company mentioned this:

## Incomplete Ownership Transfer Flag

The auditing team has not mentioned the ownership transfer of each and every component of the contracts and has not listed every concerned component to better target them.

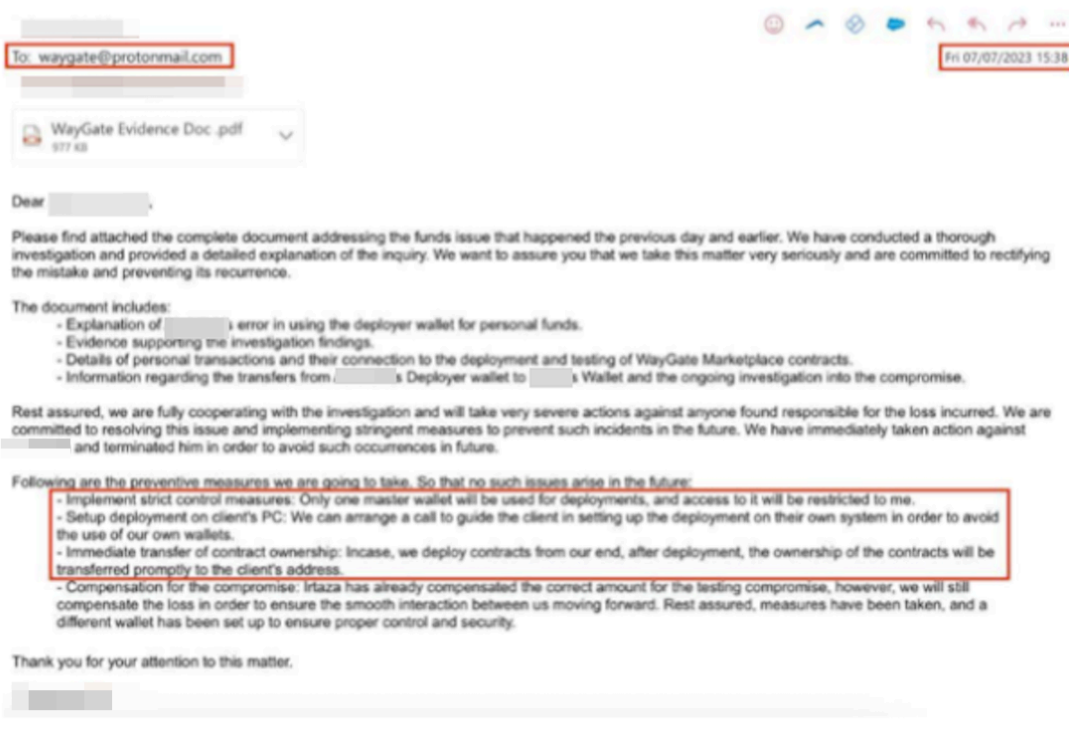
The guides by **Openzeppelin** and **Consensys** ( famous auditing firms ) suggest that ensuring that the right person has complete control is something of critical importance. The auditor should have flagged this critically in their report that the ownership transfer of each and every component is the first thing as the development team has insisted ( seen in next section ) . In this way , neither the auditor nor the client has taken it so seriously.

This **oversight** allowed the deployer to retain control over these components, which were subsequently exploited by the hacker. The auditor failed to identify this incomplete ownership transfer, leaving the project vulnerable to attack.

If the development company was aware of the ownership flag, why they didn't transfer the ownership of all contracts to the customer?

## Client's Delayed Response and Vulnerability to Attack

The development team urged the client to take custodianship of the contracts. However, due to some preferences of other tasks of the client, the ownership transfer has been delayed.



The screenshot shows an email interface. The 'To' field is 'waygate@protonmail.com' and the date is 'Fri 07/07/2023 15:38'. The email is titled 'WayGate Evidence Doc .pdf' (577 KB). The body of the email is as follows:

Dear [redacted],

Please find attached the complete document addressing the funds issue that happened the previous day and earlier. We have conducted a thorough investigation and provided a detailed explanation of the inquiry. We want to assure you that we take this matter very seriously and are committed to rectifying the mistake and preventing its recurrence.

The document includes:

- Explanation of [redacted]'s error in using the deployer wallet for personal funds.
- Evidence supporting the investigation findings.
- Details of personal transactions and their connection to the deployment and testing of WayGate Marketplace contracts.
- Information regarding the transfers from [redacted]'s Deployer wallet to [redacted]'s Wallet and the ongoing investigation into the compromise.

Rest assured, we are fully cooperating with the investigation and will take very severe actions against anyone found responsible for the loss incurred. We are committed to resolving this issue and implementing stringent measures to prevent such incidents in the future. We have immediately taken action against [redacted] and terminated him in order to avoid such occurrences in future.

Following are the preventive measures we are going to take, so that no such issues arise in the future:

- Implement strict control measures: Only one master wallet will be used for deployments, and access to it will be restricted to me.
- Setup deployment on client's PC: We can arrange a call to guide the client in setting up the deployment on their own system in order to avoid the use of our own wallets.
- Immediate transfer of contract ownership: In case, we deploy contracts from our end, after deployment, the ownership of the contracts will be transferred promptly to the client's address.
- Compensation for the compromise: Infaza has already compensated the correct amount for the testing compromise, however, we will still compensate the loss in order to ensure the smooth interaction between us moving forward. Rest assured, measures have been taken, and a different wallet has been set up to ensure proper control and security.

Thank you for your attention to this matter.

[redacted]

# Deployer wallet risk

We ask the development company to explain the following:

## Deployer Wallet Risks and Single Points of Failure

Using the historical internet data, Approximately 10 million private keys have been hacked. The systems that allow on single accounts are prone to private keys theft attacks. While building blockchain systems we often think of owners as a trusted on-person wallet to ease things but its security implications come handy when going through an audit. The auditor overlooked the potential risks associated with the deployer wallet, a single point of failure that could compromise the entire system.

This oversight allowed the hackers to gain access to the system by compromising the deployer wallet. Although multisig wallets were not very popular in development and are specifically implemented as per the special request of client or audit analyst. The auditor should have recommended the use of multisig wallets to mitigate these risks and the development team would definitely have implemented it.

### Deployer cannot mint any new tokens

| Name                 | Exist | Tested | Status |
|----------------------|-------|--------|--------|
| Deployer cannot mint | ✓     | ✓      | ✓      |
| Max / Total Supply   | N/A   |        |        |

#### Comments:

##### v1.0

- Owner cannot mint new tokens but keep it in mind that the owner is also able to upgrade the contracts

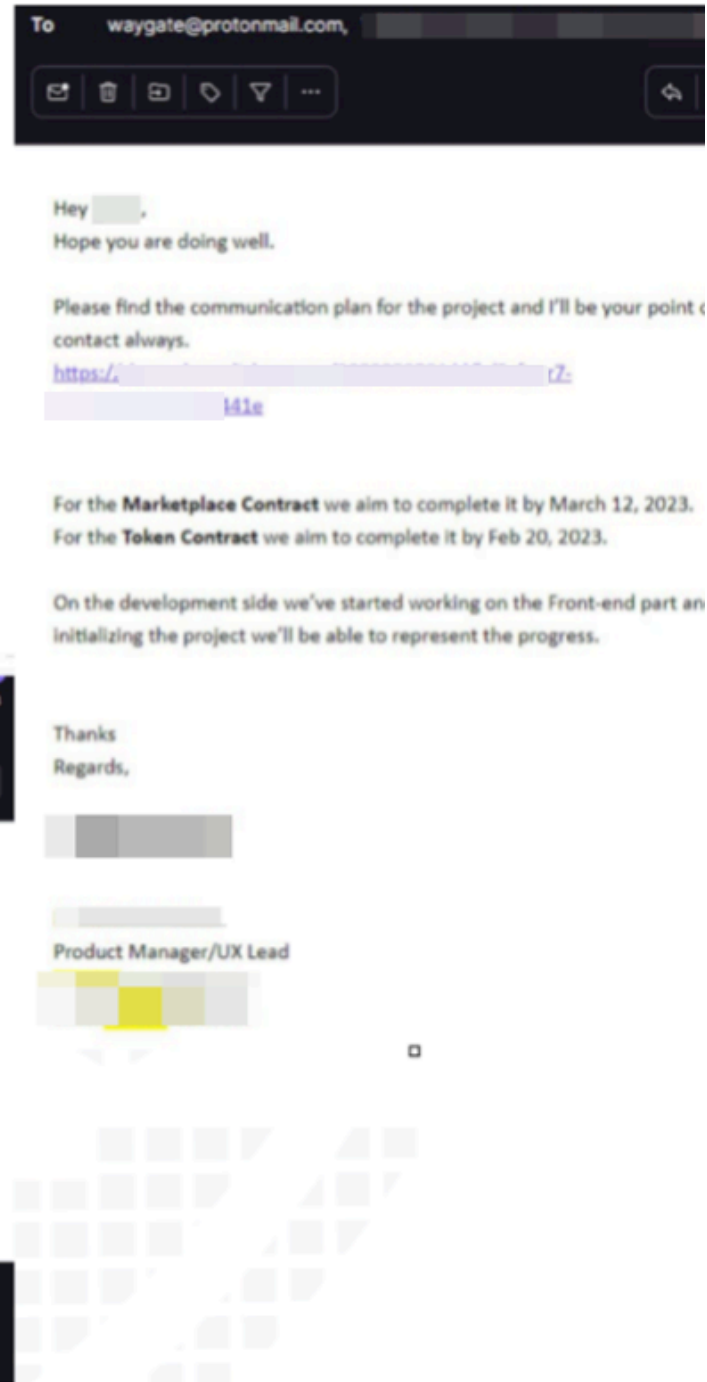
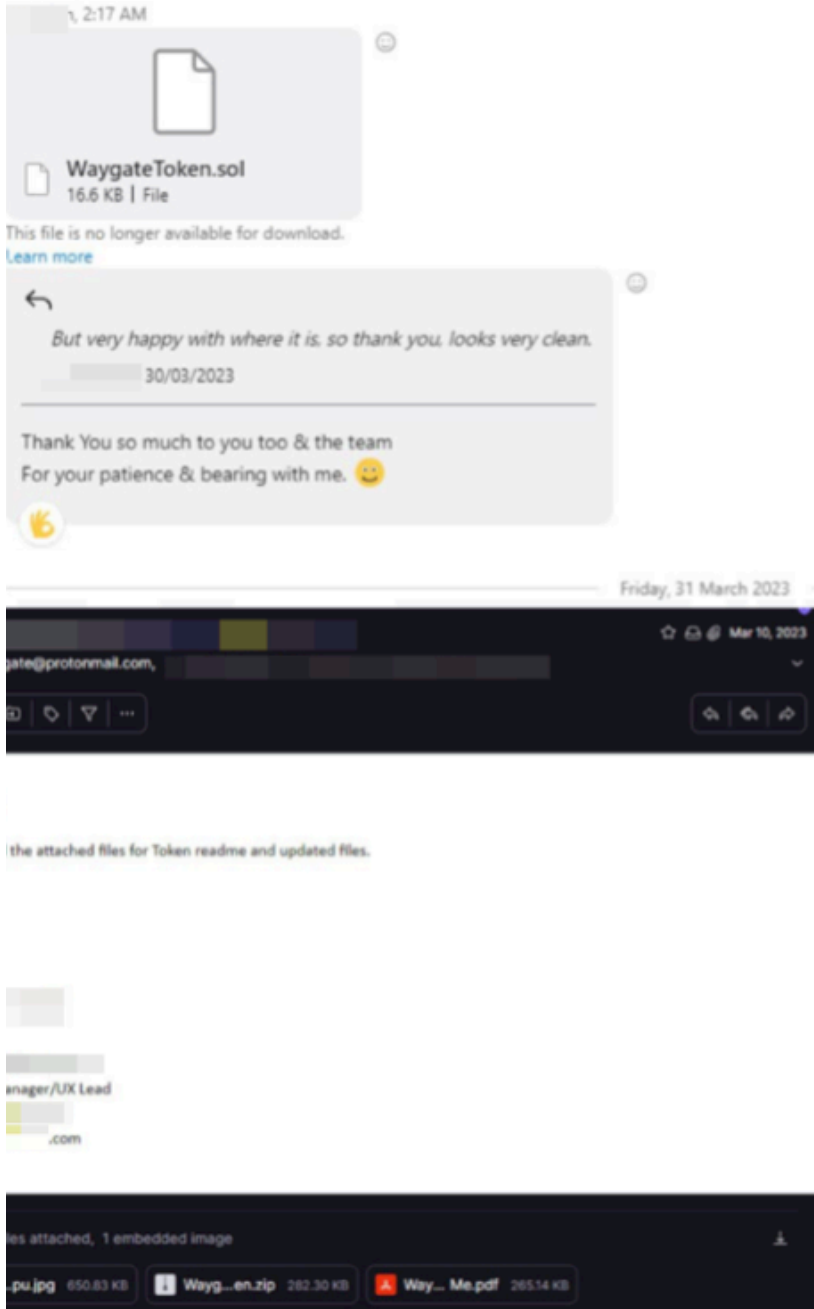
*The auditor listed that the deployer can not mint tokens but he failed to include that even the deployer can upgrade the contract in addition to the owner. This is an oversight.*

How would it be possible to get the private key when the company was the deployer of the contract and the proxy? With this statement, the company is confirming that the deployer wallet of their company was hacked.

# Proof that the development company developed & deployed the contracts

This is the conversation between the Waygate customer and the company:

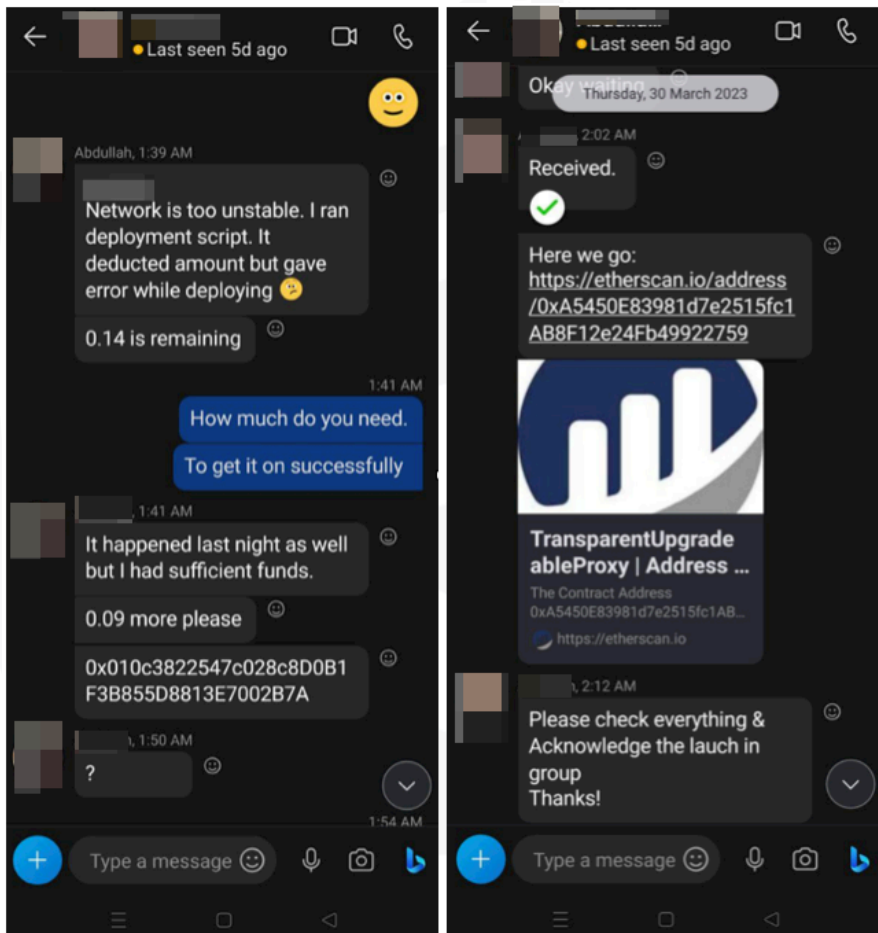
**Proof that the contracts were created by them:**



## Proof that the deployer address is owned by the development company:

The TX for the 0.09 is here: <https://etherscan.io/tx/0x3595df37ea28c3999b9593ce3fb8f9fa4138f4212f6b0105920be909af7f3b80>  
From: <https://etherscan.io/address/0x4e5b2e1dc63f6b91cb6cd759936495434c7e972f>  
To: <https://etherscan.io/address/0x010c3822547c028c8d0b1f3b855d8813e7002b7a>

The development company holds the ownership of the contracts. In their statement, they said that the ownership was transferred by hacking the private key, which means that they had a security breach.



## Conclusion

In their report, we can see that the development company tried to push the blame onto our audit report. They developed and deployed the contracts. Then they transferred the ownership of the implementation to the owner, but not the proxy. Allegedly, the private key of the wallet with ownership of the proxy was hacked.

We are also in communication with the Waygate owner, and he confirmed that this was not the first time the development company blamed others for their mistakes. Some open questions from our side to the development company would be:

Why was the proxy ownership not also transferred to the customer? Why did the development company implement the wrong admin function with a hardcoded address and remove "onlyOwner" when it was supposed to be calling from the owner only?