Cyberscope

Audit Report
ATM88

May 2023

Network	BSC
Address	0x52c9D415173489625de99dbbE4C7A79e90E9e1D0
Audited by	© cyberscope

Table of Contents

Review

Contract Name	ATM88Upgradeable
Compiler Version	v0.8.2+commit.661d1103
Optimization	150 runs
Explorer	https://bscscan.com/address/0x52c9d415173489625de99dbbe4c7a79e90e9e1d0
Address	0x52c9d415173489625de99dbbe4c7a79e90e9e1d0
Network	BSC
Decimals	18

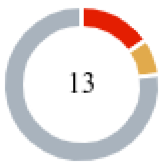
Audit Updates

Initial Audit	18 May 2023

Source Files

Filename	SHA256
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	f0cbb88e6cbc994b565645eabd4320d27d529c7f1f4b3abb5fc263f3961c0a24
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	6e058aaee8c641107b209b62c34d484f2f125a44ecb66f7204a701614dfc1d68
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	8aecaaba0f09bc906c27867246210adfd19230a3e4a209a1909045c633030476
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	a439a162881f7f36131b1fe307aa2a8dc98ac3f01ac121ff92fbbc25d0d216b5
@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol	68bcca423fc72ec9625e219c9e36306c726a347e43f3711467c579bd3f6500c8
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	db1d80b38061ba675444e6ad861a621d99666042950278d6cdeae9a108afdd17
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	44edc4d7099c781d11421cea2d82a52948e738f5f6191c8ad01dfc0f9858549c
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
contracts/coin/ATM88.sol	d6156bfbe4db68bca92cd9f73f82a8d20e25d7a96af65e3c0063391280374fc9
contracts/interface/IFactory.sol	6d4372a8b92ad975c0f89034fea0c2e9ae072e72268c8439ebbd5eca6c8bd149
contracts/interface/IPinkAntiBot.sol	7225b28b58bf1954a21981e01b72ed6e8f88595d1ad52672eabb1011fb7aca8b
contracts/interface/IRouter.sol	1327fa034fffa54c1f44f16fea2847eaaa77fe85fa904591ee1451050534af06

Findings Breakdown



●	Critical	2
●	Medium	1
●	Minor / Informative	10

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	10	0	0	0

Analysis

🔴 Critical 🟠 Medium ⚪ Minor / Informative 🔵 Pass

Severity	Code	Description	Status
🟠	ST	Stops Transactions	Unresolved
🔵	OCTD	Transfers Contract's Tokens	Passed
🔵	OTUT	Transfers User's Tokens	Passed
🔵	ELFM	Exceeds Fees Limit	Passed
🔵	ULTW	Transfers Liquidity to Team Wallet	Passed
🔵	MT	Mints Tokens	Passed
🔵	BT	Burns Tokens	Passed
🔴	BC	Blacklists Addresses	Unresolved

ST - Stops Transactions

Criticality	Medium
Location	contracts/coin/ATM88.sol#L96,106
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the whitelisted addresses. The owner may take advantage of it by setting the `tradingEnabled` to false.

```
if (!tradingEnabled) {
    require(
        isWhiteList[sender] && isWhiteList[recipient],
        "Not allow to trade"
    );
}
```

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the `minATM` to zero.

```
require(balanceOfATM >= minATM, "Hold 1000 ATM to sell");
```

Recommendation

The contract could embody a check for not allowing setting the `minATM` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/coin/ATM88.sol#L184
Status	Unresolved

Description

The contract owner has the authority to massively stop addresses from transactions. The owner may take advantage of it by calling the `bulkBlacklist` function.

```
function bulkBlacklist(
    address[] memory accounts,
    bool state
) external onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        blackList[accounts[i]] = state;
    }
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	FPP	Function Public Permissions	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

FPP - Function Public Permissions

Criticality	Critical
Location	contracts/coin/ATM88.sol#L197
Status	Unresolved

Description

The `rescueERC20` function allows users to claim all the balance of the contract. The function is marked as external and can be accessed by any user. As a result, any user can claim all of the contract's token balance.

```
function rescueERC20(address tokenAdd, uint256 amount) external {
    IERC20Upgradeable(tokenAdd).transfer(devWallet, amount);
}
```

Recommendation

The team is advised to add proper access controls and checks to prevent such vulnerabilities and ensure the security of the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L150
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the whitelisted status of an account even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function updateIsWhileList(address account, bool state) external onlyOwner {
    isWhiteList[account] = state;
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L94
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The variable `feeSwap` stores the fee amount that is added to the `devWallet` . The variable should be named `devFee` .

```
uint256 feeSwap;
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L26,27,28
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public genesisBlock
uint256 private deadline
uint256 private launchtax
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L50,57,124,128,133,138,163
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function __ATM_init() internal initializer {
        __ERC20_init("ATM88", "ATM88");
        __Ownable_init();
        __Pausable_init();
        __ATM_init_unchained();
    }

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, and maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L26,27,28
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256 public genesisBlock
uint256 private deadline
uint256 private launchtax
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L130,135,143
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
sellFee = _fee
buyFee = _fee
minATM = _minATM * 10 ** 18
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L129,134
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

The variable `_fee` is an unsigned integer. Hence, its value will always be greater than or equal to zero. The `MIN_FEE` variable is also set to zero. As a result, this check is redundant.

```
require(_fee <= MAX_FEE && _fee >= MIN_FEE, "Invalid fee");
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L94
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 feeSwap
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L177
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
devWallet = newWallet
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/coin/ATM88.sol#L198
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20Upgradeable(tokenAdd).transfer(devWallet, amount)
```

Recommendation

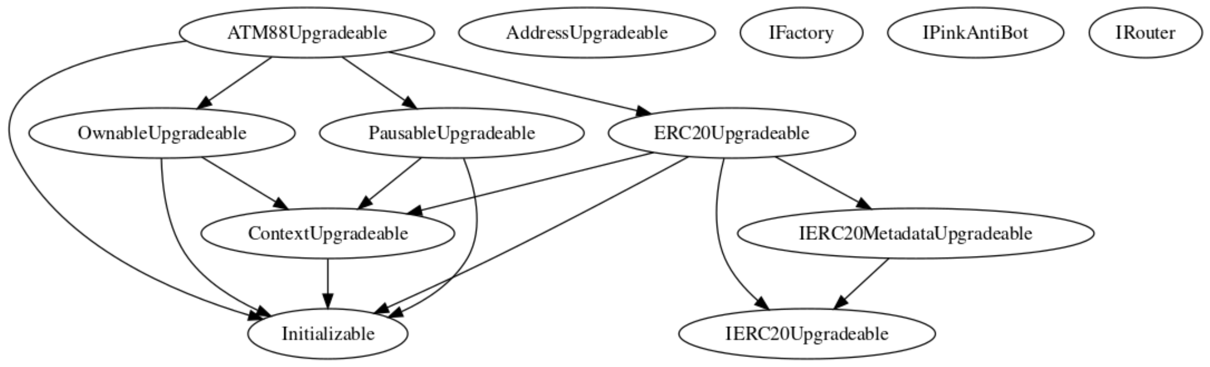
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

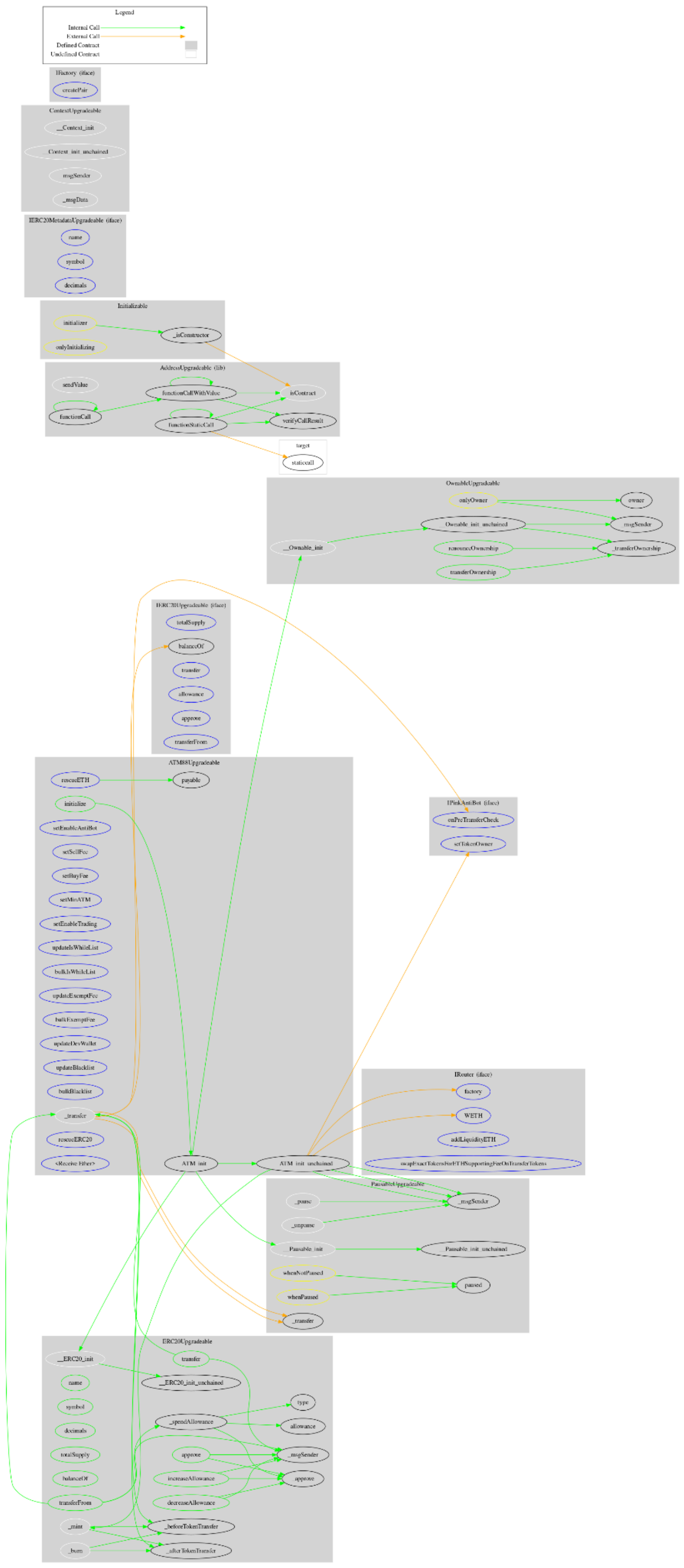
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ATM88Upgradeable	Implementation	Initializable, ERC20Upgradeable, OwnableUpgradeable, PausableUpgradeable		
	initialize	Public	✓	initializer
	__ATM_init	Internal	✓	initializer
	__ATM_init_unchained	Internal	✓	initializer
	_transfer	Internal	✓	
	setEnableAntiBot	External	✓	onlyOwner
	setSellFee	External	✓	onlyOwner
	setBuyFee	External	✓	onlyOwner
	setMinATM	External	✓	onlyOwner
	setEnableTrading	External	✓	onlyOwner
	updateIsWhileList	External	✓	onlyOwner
	bulkIsWhileList	External	✓	onlyOwner
	updateExemptFee	External	✓	onlyOwner
	bulkExemptFee	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	updateBlacklist	External	✓	onlyOwner

	bulkBlacklist	External	✓	onlyOwner
	rescueETH	External	✓	onlyOwner
	rescueERC20	External	✓	-
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IFactory (iface)
createPair
ContextUpgradeable
IERC20MetadataUpgradeable (iface)
Initializable
AddressUpgradeable (lib)
target
OwnableUpgradeable
IERC20Upgradeable (iface)
ATM88Upgradeable
IPinkAntiBot (iface)
IRouter (iface)
PausableUpgradeable
ERC20Upgradeable

Summary

ATM88 - Casino Game Series contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 10% buy and sell fees.

At the time of the audit report, the contract with address 0x52c9d415173489625de99dbbe4c7a79e90e9e1d0 is pointed by the following proxy address: 0xbc0640a9af9048385a241b899bb5184796ae0c98.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io