Cyberscope

Audit Report

Galaxy Fox Bridge

May 2024

Repository https://github.com/humanshield89/galaxy-fox-token

Commit f0fc44d6a19a2cdfffc2fc86b696389ad0b88fca

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/humanshield89/galaxy-fox-token
Commit	f0fc44d6a19a2cdfffc2fc86b696389ad0b88fca

Audit Updates

Initial Audit	14 May 2024

Source Files

Filename	SHA256
GFOXBridgeUpgradable.sol	9b4765c0b9df4324265da19a39ab43cb74b6378884de708f1682884ccc20b529
GFOXBridgeStorage.sol	8a63e68ac34bff0e6a28b2425d13fe9552ef4b0ca6e099528913202961e957ac

Overview

GFOXBridgeUpgradable Contract

The GFOXBridgeUpgradable contract is designed as an upgradable smart contract leveraging the UUPS (Universal Upgradeable Proxy Standard) pattern from OpenZeppelin. It inherits from both `GFOXBridgeStorage`, which manages the bridge functionality, and `UUPSUpgradeable`, a library that provides the mechanisms for upgrading the contract. The constructor initializes the contract with an address for a message forwarder, setting up compatibility with EIP-2771 to handle meta-transactions, thereby enabling transactions to be paid for by a third party. The `_authorizeUpgrade` function is overridden to ensure that only the contract owner can authorize upgrades, adding a layer of security that restricts upgradeability to authorized addresses only. This structure makes the contract both upgradeable and secure, adhering to modern development patterns for long-term flexibility and maintenance.

GFOXBridgeStorage Contract

The GFOXBridgeStorage contract is an integral part of a bridge mechanism that enables the transfer of the `gfoxToken` across different blockchain networks. It utilizes OpenZeppelin's libraries for secure token handling and upgradable contract capabilities. The contract inherits from `GFAccessControlUpgradable` for access control mechanisms and `EIP712Upgradeable` to provide typed structured data hashing and signing capabilities. This contract manages a list of approved signers and tracks processed transactions to prevent double spending, while also allowing token bridging and claim functionalities through cryptographic validations.

setSigner Functionality

The setSigner function allows the contract owner to manage signers dynamically, enabling or disabling their permissions to authorize transactions. This function is crucial for maintaining the security and integrity of the bridge, as it ensures that only trusted entities can approve the bridging of tokens.

initialize Functionality

The initialize function is used to set up the contract upon deployment. It configures the token contract address for `gfoxToken`, defines the contract's owner, and initializes the list of signers and admins. This setup is essential for starting the bridge with the required permissions and roles defined, ensuring that the bridge operates under controlled access from the outset. The function also invokes initializations for access control and EIP712 settings, establishing a strong foundation for secure operations.

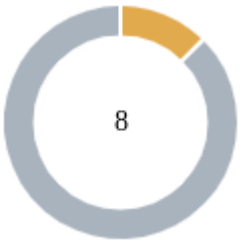
claimTx Functionality

Users have the ability to claim gfoxToken through the `claimTx` function by submitting a transaction hash from the source chain, along with the amount, their address, the source chain's ID, and a valid signature from an authorized signer. This function checks if the transaction has already been processed to prevent duplication, verifies the signature against the list of authorized signers, and then transfers the specified amount of `gfoxToken` to the receiver's address. This mechanism is crucial for enabling cross-chain transactions where users can safely claim their tokens on a different network.

bridgeGfox Functionality

The `bridgeGfox` function enables users to send gfoxToken to the bridge contract. Users specify the amount they wish to bridge, and the function transfers the tokens from the user's address to the bridge contract. This functionality is vital for moving tokens across chains, as it transfers tokens into the contract on the current chain.

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	RTL	Repeat Transaction Limitation	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	PLT	Potential Locked Tokens	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	TSI	Tokens Sufficiency Insurance	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

RTL - Repeat Transaction Limitation

Criticality	Medium
Location	GFOXBridgeStorage.sol#L71
Status	Unresolved

Description

The contract is currently designed such that if a user attempts to transfer the same token amount to the same recipient under the same chain ID, the transaction `hash` must differ from any previous transactions for the transfer to proceed. This design inadvertently increases the risk of centralization because it necessitates administrator intervention to alter the hash for identical subsequent transactions. Specifically, the admin must provide different hashes for the same set of transaction parameters (amount, receiver, chainID) to execute a repeat transaction. This mechanism could centralize control over transaction processing to a few, potentially compromising the decentralized nature of the contract.

```
function claimTx(
        bytes32 hash, // transaction hash from source chain
        uint256 amount, // amount of tokens to transfer
        address receiver, // receiver address
        uint256 chainId, // source chain id
        bytes calldata signature
    ) external {
        require(
            !processedTx[chainId][hash],
            "GFOXBridge: transaction already processed"
        );

        bytes32 digest = _hashTypedDataV4(
            keccak256(
                abi.encode(
                    keccak256(
                        "Claim(address receiver,uint256 amount,uint256
chainId,bytes32 hash)"
                    ),
                    receiver,
                    amount,
                    chainId,
                    hash
                )
            )
        );

        address signer = ECDSA.recover(digest, signature);
    ...
```

Recommendation

It is recommended to incorporate a `nonce` variable within the transaction parameters to ensure each transaction hash is unique, even when the amount, receiver, and chain ID are identical. By integrating a nonce, the hash will naturally differ with each transaction without requiring manual intervention. This approach enhances decentralization and security, maintaining the integrity and trustless nature of the blockchain operations. Adjusting the `claimTx` function to include a `nonce` in its hash calculation would automate this process, ensuring all transactions remain distinct and independently verifiable.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L24
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
    function setSigner(address _signer, bool _status) external
onlyOwner {
        require(_signer != address(0), "GFOXBridge: invalid
signer");
        signers[_signer] = _status;
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PLT - Potential Locked Tokens

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L108
Status	Unresolved

Description

The contract includes the `bridgeGfox` function, which facilitates the transfer of gfoxToken from the user to the contract. This function effectively transfers the specified amount of `gfoxToken` from the message sender to the contract itself. While the function includes an event emission that logs the transaction details, it notably lacks any corresponding functionality to handle or utilize the transferred tokens post-deposit. As a result, the tokens remain indefinitely locked within the contract with no available mechanism for retrieval or further manipulation. This oversight can lead to permanent token loss, reducing user trust and potentially impacting the token's utility and value.

```
function bridgeGfox(uint256 amount) external {
    gfoxToken.safeTransferFrom(_msgSender(), address(this),
amount);
    emit BridgeGfox(_msgSender(), amount, block.chainid);
}
```

Recommendation

It is recommended to thoroughly review the intended functionality of the `bridgeGfox` function. If the design requires that tokens be retrievable or usable after being bridged to the contract, corresponding functionalities must be implemented to allow for such operations. This could include methods for token withdrawal by authorized parties or further processing within the contract. Additionally, if the contract is part of a larger ecosystem where other contracts handle these tokens, which is out of the current audit scope, ensure that there is clear documentation and understanding of the token flow and accessibility. Implementing these changes will enhance the contract's usability and ensure that token handling aligns with the overall system design and user expectations.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L109
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
gfoxToken.safeTransferFrom(_msgSender(), address(this),
amount);
emit BridgeGfox(_msgSender(), amount, block.chainid);
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the

contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L24
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
   function setSigner(address _signer, bool _status) external
onlyOwner {
        require(_signer != address(0), "GFOXBridge: invalid
signer");
        signers[_signer] = _status;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L49,101
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
gfoxToken = IERC20(_gfoxToken);
...
gfoxToken.safeTransfer(receiver, amount);
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the presale tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L24,30,31,32,33,43,44,45,46,47,114
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.

```
bool _status
address _signer
address _gfoxToken
address _owner
address[] memory _signers
address[] memory _admins

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, 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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	GFOXBridgeStorage.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.23;
```

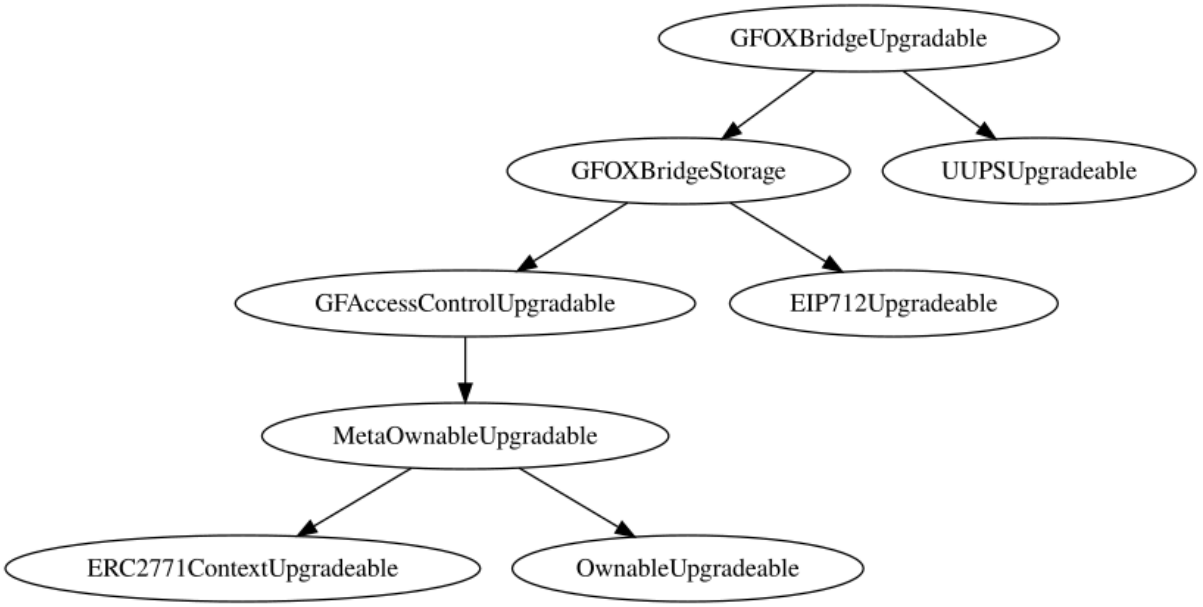
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

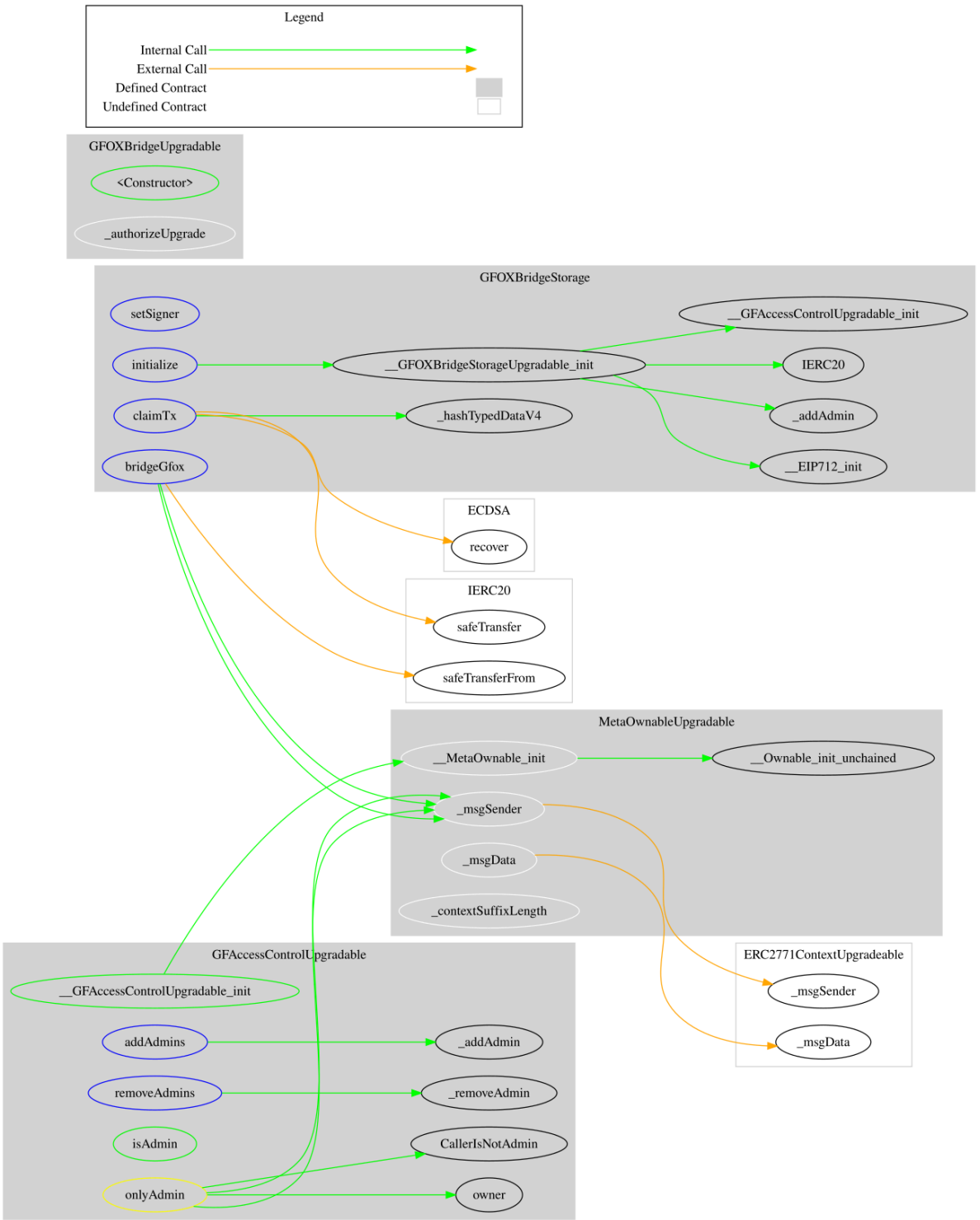
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
GFOXBridgeUpgradable	Implementation	GFOXBridgeStorage, UUPSUpgradeable		
		Public	✓	ERC2771ContextUpgradeable
	_authorizeUpgrade	Internal	✓	onlyOwner
GFOXBridgeStorage	Implementation	GFAccessControlUpgradable, EIP712Upgradeable		
	setSigner	External	✓	onlyOwner
	initialize	External	✓	initializer
	__GFOXBridgeStorageUpgradable_init	Internal	✓	
	claimTx	External	✓	-
	bridgeGfox	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
GFOXBridgeUpgradable
<Constructor>
_authorizeUpgrade
GFOXBridgeStorage
setSigner
initialize
claimTx
bridgeGfox
__GFOXBridgeStorageUpgradable_init
_hashTypedDataV4
__GFAccessControlUpgradable_init
IERC20
_addAdmin
__EIP712_init
ECDSA
recover
IERC20
safeTransfer
safeTransferFrom
MetaOwnableUpgradable
__MetaOwnable_init
__Ownable_init_unchained
_msgSender
_msgData
_contextSuffixLength
GFAccessControlUpgradable
__GFAccessControlUpgradable_init
addAdmins
removeAdmins
isAdmin
onlyAdmin
_addAdmin
_removeAdmin
CallerIsNotAdmin
owner
ERC2771ContextUpgradeable
_msgSender
_msgData

Summary

The Galaxy Fox contract facilitates a bridging mechanism between different blockchain networks. This audit investigates security issues, business logic concerns, and potential improvements.

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