Cyberscope
A TAC Security Company

Audit Report
XNAP Token

December 2025

Contract	XNAPToken.sol
SHA256	bfb9d5cbbe620bb0cb485eb230afd4fdf7f532bca4861071668e27bee26c5705
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PGA	Potential Griefing Attack	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	MOF	Missing Ownership Functionality	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PAV	Pair Address Validation	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	UAR	Unexcluded Address Restrictions	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved

Table of Contents

Analysis 2
Diagnostics 3
Table of Contents 4
Risk Classification 6
Review 7
Audit Updates 7
Source Files 7
Findings Breakdown 7
ST - Stops Transactions 9
Description 9
Recommendation 11
PGA - Potential Griefing Attack 12
Description 12
Recommendation 12
CCR - Contract Centralization Risk 13
Description 13
Recommendation 13
IDI - Immutable Declaration Improvement 14
Description 14
Recommendation 14
MVN - Misleading Variables Naming 15
Description 15
Recommendation 16
MOF - Missing Ownership Functionality 17
Description 17
Recommendation 17
NWES - Nonconformity with ERC-20 Standard 18
Description 18
Recommendation 18
PAV - Pair Address Validation 19
Description 19
Recommendation 20
PTAI - Potential Transfer Amount Inconsistency 21
Description 21
Recommendation 22
UAR - Unexcluded Address Restrictions 23
Description 23
Recommendation 24
L02 - State Variables could be Declared Constant 25

Description 25
Recommendation 25
Functions Analysis 26
Inheritance Graph 28
Flow Graph 29
Summary 30
Disclaimer 31
About Cyberscope 32

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

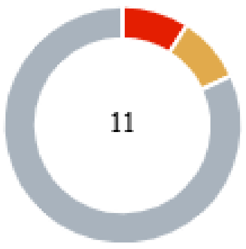
Audit Updates

Initial Audit	06 Oct 2025 https://github.com/cyberscope-io/audits/blob/main/xnap/v1/audit.pdf
Corrected Phase 2	20 Oct 2025 https://github.com/cyberscope-io/audits/blob/main/xnap/v2/audit.pdf
Corrected Phase 3	04 Nov 2025 https://github.com/cyberscope-io/audits/blob/main/xnap/v3/audit.pdf
Corrected Phase 4	31 Dec 2025

Source Files

Filename	SHA256
XNAPToken.sol	bfb9d5cbbe620bb0cb485eb230afd4fdf7f532bca4861071668e27bee26c5705

Findings Breakdown



🟥	Critical	1
🟧	Medium	1
⬜	Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
🟥 Critical	1	0	0	0
🟧 Medium	1	0	0	0
⬜ Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	XNAPToken.sol#L61,66,101
Status	Unresolved

Description

The token transactions are initially disabled. The contract owner has the authority to enable all sales by setting `tradingEnabled` to true and `limitsEnabled` to false.

Shell
```
modifier tradingAllowed(address from, address to, uint256
amount) {
    if (!tradingEnabled) {
        require(from == owner || to == owner, "trading
disabled");

    }
```

Furthermore if the pair address is unset or misconfigured, transaction limits and cooldown checks may be incorrectly applied to the actual liquidity pool address. This misconfiguration can unintentionally block or disrupt buy and sell operations on decentralized exchanges, effectively impairing normal trading activity.

Shell
```
function setPancakePair(address pair) external onlyOwner {
    require(pair != address(0), "zero");
    require(pancakePair == address(0), "already set");
    pancakePair = pair;
    emit PairSet(pair);
}

...

if (limitsEnabled && from != owner && to != owner && to !=
pancakePair) {
    require(amount <= maxTxAmount, "max tx");
    require(_balances[to] + amount <= maxWalletAmount,
"max wallet");

    require(
        block.timestamp >= lastTxTime[from] +
cooldownTime,
        "cooldown"

    );
```

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. The team is advised to ensure that the correct liquidity pair address is properly configured before enabling trading. Implementing additional validation checks (such as verifying that the provided address is a deployed contract) can help prevent misconfiguration.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PGA - Potential Griefing Attack

Criticality	Medium
Location	XNAPToken.sol#L70
Status	Unresolved

Description

The contract includes functionality designed to enforce specific conditions on transactions. However, this design is vulnerable to griefing attacks, where malicious actors can exploit the contract's logic to interfere with legitimate user operations.

In this case, the contract enforces transactional limits based on on-chain activity. A third party could strategically interact with the contract's state to disrupt normal user operations, resulting in failed transactions or unintended behavior.

Such griefing attacks could undermine the contract's usability and obstruct legitimate user operations.

Shell
```
require(block.timestamp >= lastTxTime[from] +
cooldownTime,"cooldown");
lastTxTime[from] = block.timestamp;
```

Recommendation

The team is advised to review the transfer mechanism to ensure that all legitimate operations are processed as intended. This will help maintain the integrity of user activities and strengthen trust in the system.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	XNAPToken.sol#L105,110
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
Shell
function enableTrading() external onlyOwner {
    ...
     }

function removeLimits() external onlyOwner {

    ...
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	XNAPToken.sol#L83,87,88
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

Shell
```
owner
maxTxAmount

maxWalletAmount
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	XNAPToken.sol#L151,153
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.

Specifically, the `BUY_BURN_BP` and `SELL_BURN_BP` are used to calculate the `burnAmount` if the users perform a buy or a sell but they are also applied during the addition and removal of liquidity from the pair.

Shell
```
if (pancakePair != address(0)) {
    if (from == pancakePair) {
        burnAmount = (value * BUY_BURN_BP) /
BP_DENOM;
    } else if (to == pancakePair) {
        burnAmount = (value * SELL_BURN_BP) /
BP_DENOM;

    }
```

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.

MOF - Missing Ownership Functionality

Criticality	Minor / Informative
Location	XNAPToken.sol#L83
Status	Unresolved

Description

The contract uses a custom ownership system where the deployer is permanently set as owner in the constructor, with no functions to transfer or renounce ownership. Critical administrative actions remain under indefinite control of this single address.

```
Shell
owner = msg.sender;
```

Recommendation

It is suggested to add standard ownership management functions (transfer and renounce) or inherit from OpenZeppelin Ownable to allow proper delegation and permanent revocation of administrative privileges if needed.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	XNAPToken.sol#L142
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
Shell
function _transfer(address from, address to,
uint256 value)
    ...
        require(value > 0, "zero");
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	XNAPToken.sol#L98
Status	Unresolved

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

Shell
```
function setPancakePair(address pair) external
onlyOwner {
        require(pair != address(0), "zero");
        require(pancakePair == address(0),
"already set");
        pancakePair = pair;
        emit PairSet(pair);

    }
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	XNAPToken.sol#L66,161
Status	Unresolved

Description

The contract calculates the burn amount, deducts the full `value` from the sender's balance while adding only `value - burnAmount` to the recipient. However, the `maxWalletAmount` checks in the `tradingAllowed` modifier use the full requested amount (pre-burn value) instead of the post-burn amount that will actually be received.

Shell
```
if (limitsEnabled && from != owner && to != owner
&& to != pancakePair) {
    require(amount <= maxTxAmount, "max tx");
    require(_balances[to] + amount <=
maxWalletAmount, "max wallet");

    require(
        block.timestamp >= lastTxTime[from] +
cooldownTime,
        "cooldown"
    );


uint256 sendAmount = value - burnAmount;
        uint256 b = _balances[from];

        require(b >= value, "balance");

        _balances[from] = b - value;

        _balances[to] += sendAmount;
```

Recommendation

The team is advised to take into consideration the actual amount that will be transferred instead of the expected amount during the `maxWalletAmount` restrictions.

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	XNAPToken.sol#L61,149
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

Shell
```
if (limitsEnabled && from != owner && to != owner
&& to != pancakePair) {
   require(amount <= maxTxAmount, "max tx");
   require(_balances[to] + amount <=
maxWalletAmount, "max wallet");
...
if (pancakePair != address(0)) {
   if (from == pancakePair) {
         burnAmount = (value * BUY_BURN_BP) /
BP_DENOM;
   } else if (to == pancakePair) {
         burnAmount = (value * SELL_BURN_BP) /
BP_DENOM;
   } else {
         burnAmount = (value * TRANSFER_BURN_BP) /
BP_DENOM;
   }
}
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	XNAPToken.sol#L35
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.

```Shell
uint256 public cooldownTime = 30 seconds
```

Recommendation

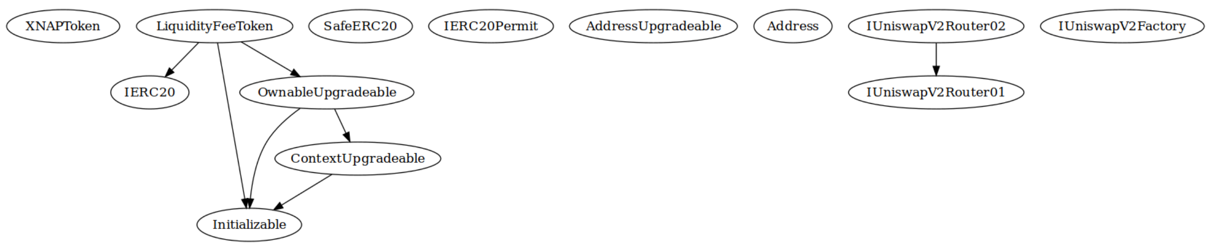
Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times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

Functions Analysis

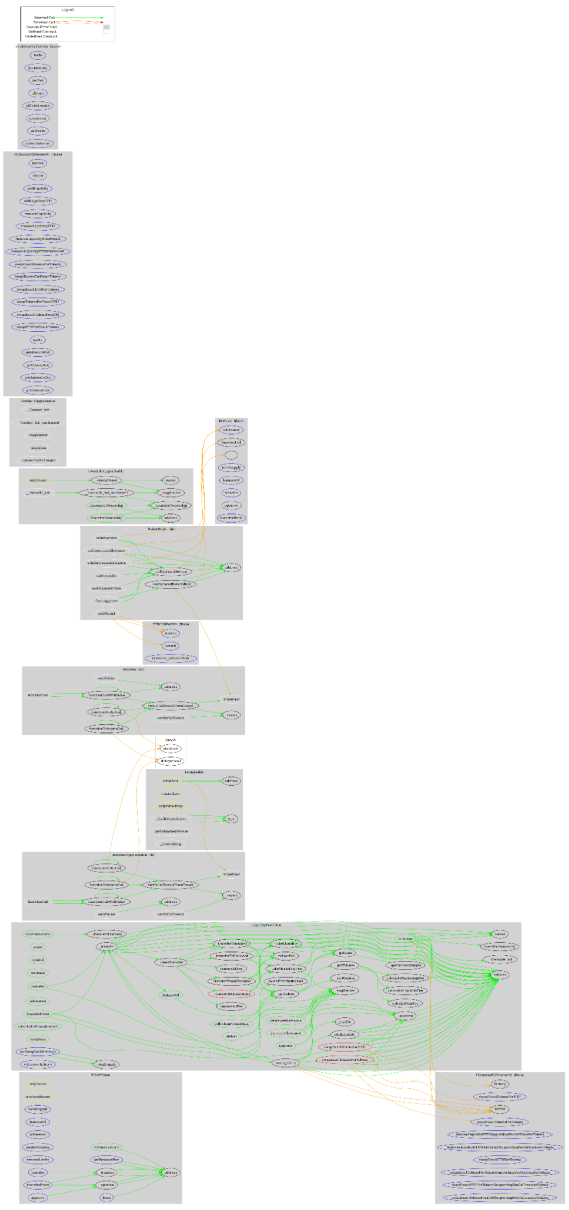
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
XNAPToken	Implementation			
		Public	✓	-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	setPancakePair	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
	_transfer	Internal	✓	tradingAllowed
	_approve	Internal	✓	
	burn	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

Inheritance Graph



Flow Graph



Summary

XNAP Token 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 stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

About Cyberscope

Cyberscope is a TAC 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.

A *TAC Security* Company

The Cyberscope team

cyberscope.io