Cyberscope

A TAC Security Company

Audit Report

AlperBarnes Token

December 2025

Network	BSC
Address	0xa107f5fc5a750151afe7395e113c7dd3d0220b83
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
Minor / Informative	IPI	Inconsistent Permit Implementation	Unresolved
Minor / Informative	RF	Redundant Functionality	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved
Minor / Informative	L20	Succeeded Transfer Check	Unresolved

Table of Contents

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

Contract Name	ALPERBARNES
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa107f5fc5a750151afe7395e113c7dd3d0220b83
Address	0xa107f5fc5a750151afe7395e113c7dd3d0220b83
Network	BSC
Symbol	ABNN
Decimals	18
Total Supply	1,000,000,000

Audit Updates

Initial Audit	29 Dec 2025

Source Files

Filename	SHA256
ALPERBARNES.sol	fa7e6a1b209bda82f3f74554abab0e734e3eb5bb7b6af33e73d1a33e707bd4a1

Findings Breakdown

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	4	0	0	0

IPI - Inconsistent Permit Implementation

Criticality	Minor / Informative
Location	ALPERBARNES.sol#L180
Status	Unresolved

Description

The contract does implement a permit-style approval flow by recovering a signer from a hashed message, verifying it against the token owner, and incrementing a nonce to block signature reuse. However it does not apply the commonly expected cryptographic hardening and parameter validation and also relies on a permanently fixed signing domain value with no ability to recompute it if the chain context changes in the future.

Shell
```
abstract contract ERC20Permit is ERC20 {
    mapping(address => uint256) public nonces;
    bytes32 public immutable DOMAIN_SEPARATOR;
    bytes32 private constant PERMIT_TYPEHASH =
keccak256("Permit(address owner,address spender,uint256
value,uint256 nonce,uint256 deadline)");

    constructor(string memory tokenNameForDomain) {
        uint256 chainId;
        assembly { chainId := chainid() }
        DOMAIN_SEPARATOR =
keccak256(abi.encode(keccak256("EIP712Domain(string
name,string version,uint256 chainId,address
verifyingContract)"),
keccak256(bytes(tokenNameForDomain)),
keccak256(bytes("1")), chainId, address(this)));
    }

    function permit(address ownerAddress, address
spenderAddress, uint256 value, uint256 deadline, uint8 v,
bytes32 r, bytes32 s) external {
        require(block.timestamp <= deadline, "signature
expired");
        bytes32 structHash =
keccak256(abi.encode(PERMIT_TYPEHASH,
ownerAddress,spenderAddress, value,
nonces[ownerAddress]++, deadline));
        bytes32 digest =
keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR,
structHash));
        address signer = ecrecover(digest, v, r, s);
        require(signer == ownerAddress && signer !=
address(0), "invalid signature");
        _approve(ownerAddress, spenderAddress, value);
    }

}
```

Recommendation

It is advised that the team unifies the permit recovery path with standard hardening by rejecting or normalizing invalid signature version bytes, enforcing a non-malleable `s` range, and storing signing domain components in a way that allows recomputation instead of keeping a frozen value.

RF - Redundant Functionality

Criticality	Minor / Informative
Location	ALPERBARNES.sol#L234,254
Status	Unresolved

Description

The contract introduces an immutable `cap` variable and overrides the internal `_mint` function to enforce a maximum mintable supply, but the `constructor` mints the entire `initialSupply` at deployment and the contract contains no mechanism for further minting, making both the `cap` variable and the `_mint` override functionally redundant outside of constructor execution, while also adding unnecessary complexity and slightly increasing deployment gas cost.

```
Shell
uint256 public immutable cap;
...
uint256 initialSupply = 1_000_000_000 * 10 ** decimals();
cap = initialSupply;
_mint(initialRecipient, initialSupply);
...
function _mint(address to, uint256 amount) internal
override {
    if (to == address(0)) revert ZeroAddress();
    if (totalSupply() + amount > cap) revert
CapExceeded();
    super._mint(to, amount);

}
```

Recommendation

Since cap is set equal to `initialSupply` and `_mint` is never callable again after deployment, the `cap` enforcement provides no ongoing security or utility guarantees, and could be removed.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	ALPERBARNES.sol#L184
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

Shell
```
assembly { chainId := chainid() }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ALPERBARNES.sol#L6
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.

Shell

```
pragma solidity ^0.8.24;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	ALPERBARNES.sol#L269
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.

Shell
```
IERC20(tokenAddress).transfer(recipient, amount)
```

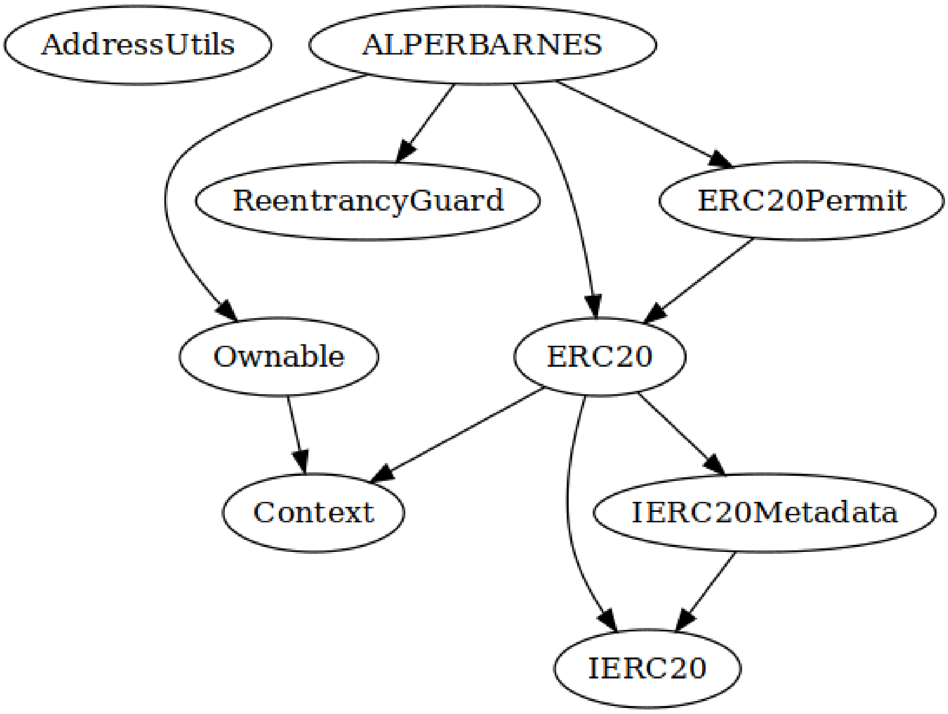
Recommendation

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.

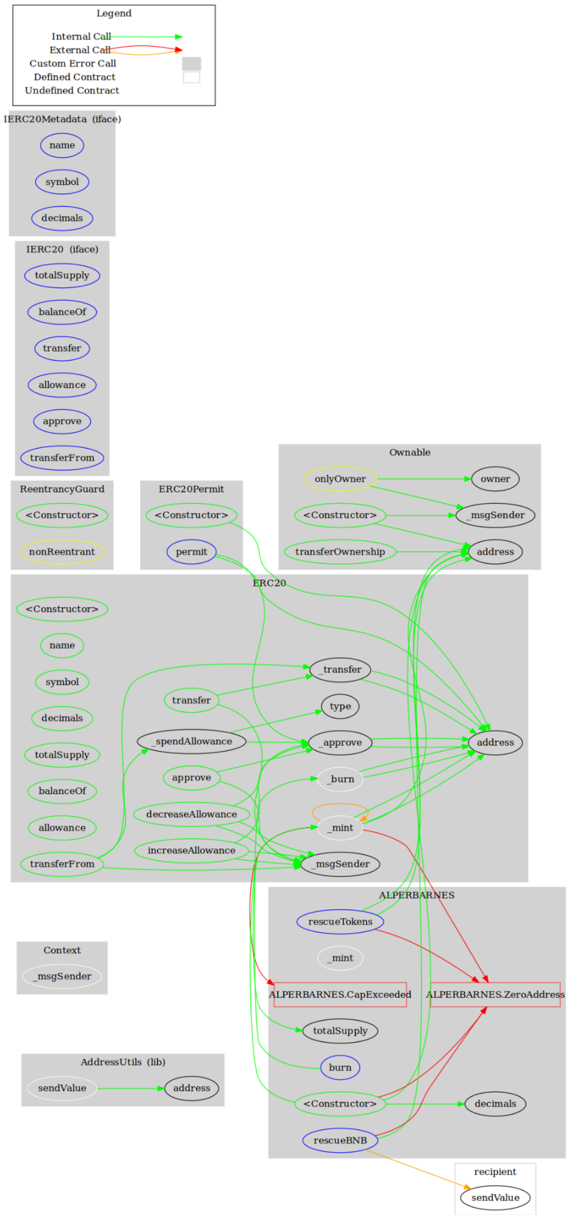
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	transfer	Public	✓	-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
owner
<Constructor>
_msgSender
transferOwnership
address
ReentrancyGuard
<Constructor>
nonReentrant
ERC20Permit
<Constructor>
permit
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
allowance
transferFrom
_transfer
transfer
type
_spendAllowance
_approve
address
approve
_burn
decreaseAllowance
_mint
increaseAllowance
msgSender
ALPERBARNES
rescueTokens
_mint
ALPERBARNES.CapExceeded
ALPERBARNES.ZeroAddress
totalSupply
burn
<Constructor>
decimals
rescueBNB
Context
_msgSender
AddressUtils (lib)
sendValue
address
recipient
sendValue

Summary

Alper Barnes contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Alper Barnes is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users’ transactions.

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 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