Coinsult

Request your audit at coinsult.net

Advanced Manual

Smart Contract Audit

November 16, 2022

Audit requested by

Baby inu

0x50f54fBCD1C6b0C847c72C71fd45032C7206A7E1

Table of Contents

Audit Summary

Project Name	Baby inu
Website	https://www.babyinu.finance/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0x50f54fBCD1C6b0C847c72C71fd45032C7206A7E1
Audit Method	Static Analysis, Manual Review
Start date of audit	November 16, 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Source Code

Coinsult was comissioned by Baby inu to perform an audit based on the following code:

https://bscscan.com/address/0x50f54fBCD1C6b0C847c72C71fd45032C7206A7E1#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- ✓ Slither: Solidity static analysis framework
- ✓ Remix: IDE Developer Tool
- ✓ CWE: Common Weakness Enumeration
- ✓ SWC: Smart Contract Weakness Classification and Test Cases
- ✓ DEX: Testnet Blockchains

Static Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) while overlaying a wide range of weakness variants that are specific to smart contracts.

ID	Description	Status
SWC-100	Function Default Visibility	Passed
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	Passed
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed
SWC-107	Reentrancy	Passed
SWC-108	State Variable Default Visibility	Resolved
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Failed
SWC-116	Block values as a proxy for time	Passed

SWC-117	Signature Malleability	Passed
SWC-118	Incorrect Constructor Name	Passed
SWC-119	Shadowing State Variables	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed
SWC-121	Missing Protection against Signature Replay Attacks	Passed
SWC-122	Lack of Proper Signature Verification	Passed
SWC-123	Requirement Violation	Passed
SWC-124	Write to Arbitrary Storage Location	Passed
SWC-125	Incorrect Inheritance Order	Passed
SWC-126	Insufficient Gas Griefing	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed
SWC-128	DoS With Block Gas Limit	Passed
SWC-129	Does not compromise the functionality of the contract in any way	Passed
SWC-130	Does not compromise the functionality of the contract in any way	Passed
SWC-131	Does not compromise the functionality of the contract in any way	Passed
SWC-132	Does not compromise the functionality of the contract in any way	Passed
SWC-133	Does not compromise the functionality of the contract in any way	Passed
SWC-134	Does not compromise the functionality of the contract in any way	Passed
SWC-135	Does not compromise the functionality of the contract in any way	Passed
SWC-136	Will definitely cause problems, this needs to be adjusted	Passed

Audit Results

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
Informational	Does not compromise the functionality of the contract in any way
Low-Risk	Won't cause any problems, but can be adjusted for improvement
Medium-Risk	Will likely cause problems and it is recommended to adjust
High-Risk	Will definitely cause problems, this needs to be adjusted

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
Informational	0	0	0	0
Low-Risk	11	0	4	7
Medium-Risk	3	0	1	2
High-Risk	1	0	0	1

Error Code	Description	Severity
CNS-016	Initial supply	Low

Initial Supply

When the contract is deployed, the contract deployer receives all of the initially created assets. Since the deployer and/or contract owner can distribute tokens without consulting the community, this could be a problem.

Recommendation

Private keys belonging to the employer and/or contract owner should be stored properly. The initial asset allocation procedure should involve consultation with the community.

Error Code	Description	Severity
CNS-015	Reliance on third-parties	Low

Reliance on third-parties

Interaction between smart contracts with third-party protocols like Uniswap and Pancakeswap. The audit's scope presupposes that third party entities will perform as intended and treats them as if they were black boxes. In the real world, third parties can be hacked and used against you. Additionally, improvements made by third parties may have negative effects, such as higher transaction costs or the deprecation of older routers.

Recommendation

Regularly check third-party dependencies, and when required, reduce severe effects.

Error Code	Description	Severity
SWC-107	CWE-841: Improper Enforcement of Behavioral Workflow	Low

Contract does not use a ReEntrancyGuard

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

Recommendation

The best practices to avoid Reentrancy weaknesses are: Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern, or use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.

Error Code	Description	Severity
SWC-108	CWE-710: Improper Adherence to Coding Standards	Low (17x) Resolved (17x)

Function Default Visibility

Functions that do not have a function visibility type specified are public by default. This can lead to a vulnerability if a developer forgot to set the visibility and a malicious user is able to make unauthorized or unintended state changes.

```
address _token;
IERC20 RWRD = IERC20(0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56);
IRouter02 dexRouter;
address[] shareholders;
mapping (address => uint256) shareholderIndexes;
mapping (address => uint256) shareholderClaims;
uint256 currentIndex;
bool initialized;
address DEV = 0xC726Ac99258552bFa2AbaABA47126a8335400299;
uint256 _totalSupply = 100_000_000_000_000 * 10**_decimals;
mapping (address => uint256) _balances;
mapping (address => mapping (address => uint256)) _allowances;
mapping (address => bool) isFeeExempt;
mapping (address => bool) isDividendExempt;
bool cannotChangeMore;
uint256 distributorGas = 500000;
bool inSwap;
```

Recommendation

Functions can be specified as being external, public, internal or private. It is recommended to make a conscious decision on which visibility type is appropriate for a function. This can dramatically reduce the attack surface of a contract system.

Error Code	Description	Severity
SLT: 078	Conformance to numeric notation best practices	Low Resolved

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 distributorGas = 500000;
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit Scenario

While 1_ether looks like 1 ether, it is 10 ether. As a result, it’s likely to be used incorrectly.

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

Error Code	Description	Severity
CNS-014	Gas optimization	Low Resolved

Duplicate dividend exclusion of pair

You have excluded the pair 2 times from dividends.

```
        isDividendExempt[pair] = true;
        isDividendExempt[address(this)] = true;
        isDividendExempt[DEAD] = true;
        isDividendExempt[marketingFeeReceiver] = true;
        isDividendExempt[devFeeReceiver] = true;
        isDividendExempt[cexFeeReceiver] = true;
        isDividendExempt[pair] = true;
        isDividendExempt[msg.sender] = true;
```

Recommendation

Remove 1 exclusion.

Error Code	Description	Severity
SLT-062	Boolean equality	Low Resolved

Boolean equality

Error Description.

```
function EnableTrading() public onlySafuDeveloper {
    require(isTradingEnabled == false,"Owner cannot stop trading");
    isTradingEnabled = true;
    swapThreshold = balanceOf(pair) / 10000;
}

function lockTax() external onlySafuDeveloper {
    require(cannotChangeMore == false,"Fees already locked");
    cannotChangeMore = true;
}
```

Recommendation

Boolean constants can be used directly and do not need to be compare to true or false.

Error Code	Description	Severity
CNS-015	Gas optimization	Low

Setting allowance of dex

Everytime swapBack is called, the router is getting the max allowance, there is no need to do this on every call.

```
function swapBack() internal swapping {

    uint256 contractBalance = balanceOf(address(this));
    _allowances[address(this)][address(dexRouter)] = type(uint256).max;

    uint256 amountToSwap = contractBalance;

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = dexRouter.WETH();
```

Recommendation

Set the allowance once and reduce gast costs.

Error Code	Description	Severity
SLT-054	Missing events arithmetic	Low Resolved

Missing events arithmetic

Missing events for critical arithmetic parameters.

```
function setFees(uint256 _cexFee, uint256 _reflectionFee, uint256 _marketingFee, uint256 _devFee) external onlySafuDeveloper {
    require(!cannotChangeMore,"Owner has locked fees");
    cexFee = _cexFee;
    reflectionFee = _reflectionFee;
    marketingFee = _marketingFee;
    devFee = _devFee;
    totalFee = _cexFee + (_reflectionFee) + (_marketingFee) + (devFee);
    checkFees(totalFee);
}

function setSwapBackSettings(bool _enabled, uint256 _amount) external onlySafuDeveloper {
    swapEnabled = _enabled;
    swapThreshold = _amount;
}
```

Recommendation

Emit an event for critical parameter changes.

Error Code	Description	Severity
CNS-017	Requirement statement	Low Resolved

Gas can be arbitrary low

If set to an arbitrary low value, the call will fail.

```
function setDistributorSettings(uint256 gas) external onlyOwner {
    require(gas < 750000);
    distributorGas = gas;
}
```

Recommendation

Set a minimum as well instead of only a maximum.

Error Code	Description	Severity
SLT-068	Conformance to Solidity naming conventions	Low Resolved

Conformance to Solidity naming conventions

Solidity defines a naming convention that should be followed.

```
function EnableTrading() public onlySafuDeveloper {
    require(isTradingEnabled == false,"Owner cannot stop trading");
    isTradingEnabled = true;
    swapThreshold = balanceOf(pair) / 10000;
}
```

Recommendation

Use enableTrading() instead of EnableTrading().

Error Code	Description	Severity
SWC-135	CWE-1164: Irrelevant Code	Medium Resolved

Code with no effects

In Solidity, it’s possible to write code that does not produce the intended effects. Currently, the solidity compiler will not return a warning for effect-free code. This can lead to the introduction of “dead” code that does not properly performing an intended action.

For example, it’s easy to miss the trailing parentheses in msg.sender.call.value(address(this).balance)("");, which could lead to a function proceeding without transferring funds to msg.sender. Although, this should be avoided by checking the return value of the call.

```
bool public buyCooldownEnabled = true;
uint8 public cooldownTimerInterval = 5;
mapping (address => uint) private cooldownTimer;

function cooldownEnabled(bool _status, uint8 _interval) public onlyOwner {
    buyCooldownEnabled = _status;
    cooldownTimerInterval = _interval;
}
```

Recommendation

The cooldown system is never used, please consider removing it and all its child functions.

Error Code	Description	Severity
CNS-016	Missing 0 checks	Medium Resolved

Transfer of 0 amount

Error Description.

```
uint256 amountBNBReflection = amountBNB * (reflectionFee) / (totalBNBFee);
uint256 amountBNBMarketing = amountBNB * (marketingFee) / (totalBNBFee);
uint256 amountBNBcexFee = amountBNB * (cexFee) / (totalBNBFee);
uint256 amountBNBDev = amountBNB * (devFee) / (totalBNBFee);

try distributor.deposit{value: amountBNBReflection}() {} catch {}
(bool tmpSuccess,) = payable(marketingFeeReceiver).call{value: amountBNBMarketing, gas: 30000}("");
(tmpSuccess,) = payable(devFeeReceiver).call{value: amountBNBDev, gas: 30000}("");
(tmpSuccess,) = payable(cexFeeReceiver).call{value: amountBNBcexFee, gas: 30000}("");
```

Recommendation

Check if the amounts are bigger than 0 before using .call to send ether.

Error Code	Description	Severity
SWC-115	CWE-477: Use of Obsolete Function	Medium

Use of tx.origin

tx.origin is a global variable in Solidity which returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable if an authorized account calls into a malicious contract. A call could be made to the vulnerable contract that passes the authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.

```
function isLimitedAddress(address ins, address out) internal view returns (bool) {

    bool isLimited = ins != owner()
        && out != owner() && tx.origin != owner() // any transaction with no direct interaction from owner will be accepted
        && msg.sender != owner()
        && !liquidityAdd[ins]  && !liquidityAdd[out] && out != DEAD && out != address(0) && out != address(this);
        return isLimited;
}
```

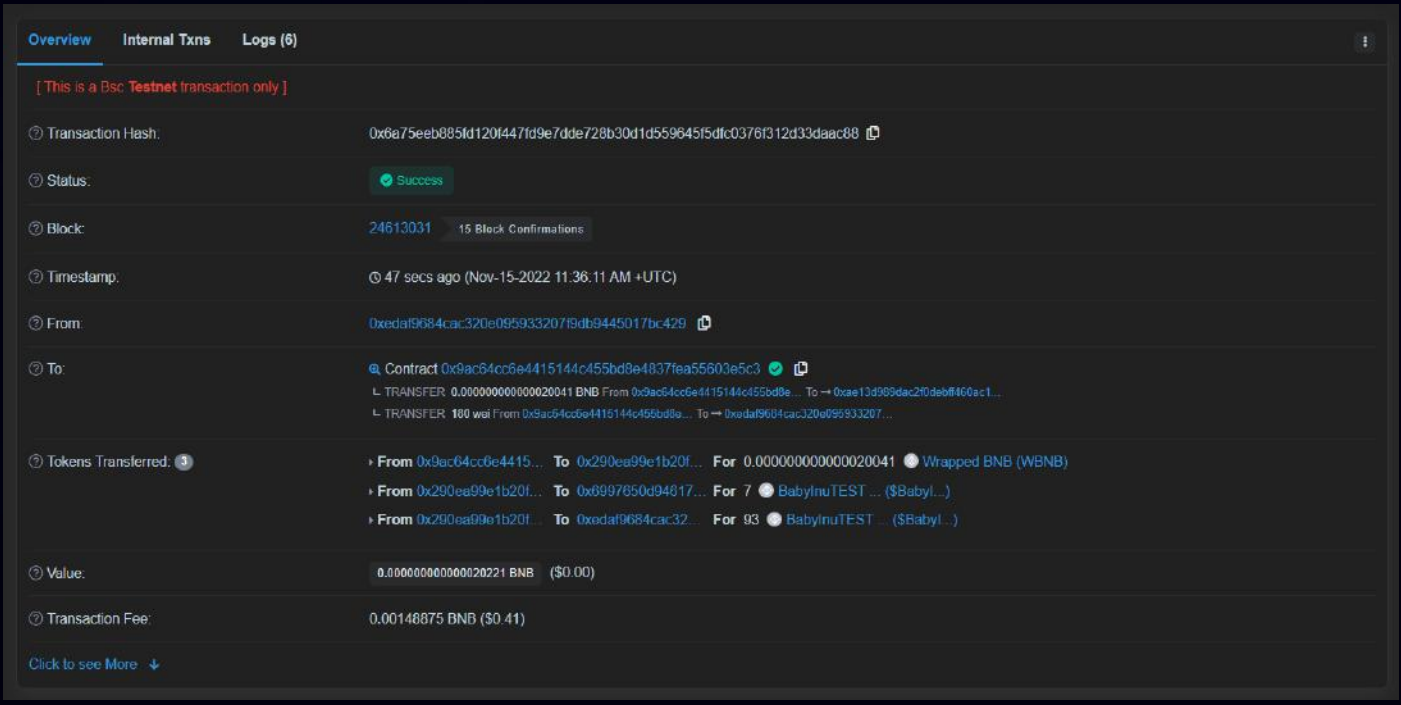
Recommendation

tx.origin should not be used for authorization. Use msg.sender instead.

Error Code	Description	Severity
CNS-001	Exclude from fees not working on buys	High Resolved

Buys still taking fees

Wallet is feeExempt, but still takes fees.



Recommendation

Make the fee system so that exempt users dont pay the fee on buys sells and transfers

Centralization: Trading fee limitation

Error Code	Description
CEN-01	Check if the owner can set fees to an arbitrary high value

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Finding	Status
Owner can **not** set fees to arbitrary high values	🟢 Passed

Type of fee	Maximum fee
Transfer fee	10%
Buy fee	10%
Sell fee	10%

Code highlight

```
function setFees(uint256 _cexFee, uint256 _reflectionFee, uint256 _marketingFee, uint256 _devFee) external onlySafuDeveloper {
    require(!cannotChangeMore,"Owner has locked fees");
    cexFee = _cexFee;
    reflectionFee = _reflectionFee;
    marketingFee = _marketingFee;
    devFee = _devFee;
    totalFee = _cexFee + (_reflectionFee) + (_marketingFee) + (devFee);
    checkFees(totalFee);
}

function checkFees(uint256 _totalFee) pure internal {
    require(_totalFee <= 10,"No more than 10%");
}
```

Centralization: Pause trading

Error Code	Description
CEN-02	Check if the owner of the smart contract can pause / prevent trading

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Finding	Status
Owner can **not** pause trading	🟢 Passed

Centralization: Max transaction amount

Error Code	Description
CEN-03	Check if the contract implemented a system to prevent big transactions

Coinsult tests if the smart contract has a maximum transaction amount. If the transaction amount exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Finding	Status
Contract does **not** contain a maximum transaction amount	Passed

Centralization: Excluding from fees

Error Code	Description
CEN-04	Check if the owner can exclude addresses from trading fees

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Finding	Status
Owner **can** exclude addresses from fees	🟡 Pending

Code to exclude from fees

```
function setIsFeeExempt(address holder, bool exempt) external onlyOwner {
    isFeeExempt[holder] = exempt;
}
```

Centralization: Mint after deployment

Error Code	Description
CEN-05	Check if the owner can mint additional tokens after deployment

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Finding	Status
Owner can **not** mint new tokens after deployment	Passed

Centralization: Ability to blacklist

Error Code	Description
CEN-06	Check if the owner can blacklist addresses from trading

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Finding	Status
Owner can **not** blacklist addresses from trading	Passed

Centralization Privileges

The owner of the contract is allowed to interact with the following functions:

-

Notes

Notes by Baby inu

No notes provided by the team.

Notes by Coinsult

Suggestion to make a public function to check if addresses are fee exempt.

Also suggest to make this improvement:

Before:

```
function setIsFeeExempt(address holder, bool exempt) external onlyOwner {
    isFeeExempt[holder] = exempt;
}
```

After

```
function setIsFeeExempt(address holder, bool exempt) external onlyOwner {
    require(isFeeExempt[holder] != exempt, “Already at this state”);
    isFeeExempt[holder] = exempt;
}
```

Constructor Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

```
constructor () {
    dexRouter = IRouter02(0x10ED43C718714eb63d5aA57B78B54704E256024E); // Mainnet: 0x10ED43C718714eb63d5aA57B78B54704E256024E
    pair = IDEXFactory(dexRouter.factory()).createPair(dexRouter.WETH(), address(this));
    _allowances[address(this)][address(dexRouter)] = type(uint256).max;
    _allowances[address(this)][address(pair)] = type(uint256).max;

    distributor = new DividendDistributor(address(dexRouter));

    isFeeExempt[msg.sender] = true;
    isFeeExempt[address(DEV)] = true;
    isFeeExempt[address(this)] = true;

    marketingFeeReceiver = payable(0x60f5F646daF8E4e837BA6DBf57917d0ca309b5fB);
    devFeeReceiver = payable(0xf0A7854ECd1e89FBD7F27605b61Bd15959040e08);
    cexFeeReceiver = payable(0x357a484EF3751A750A1857c2b3C8C0fCae096722);

    isDividendExempt[pair] = true;
    isDividendExempt[address(this)] = true;
    isDividendExempt[DEAD] = true;
    isDividendExempt[marketingFeeReceiver] = true;
    isDividendExempt[devFeeReceiver] = true;
    isDividendExempt[cexFeeReceiver] = true;
    isDividendExempt[pair] = true;
    isDividendExempt[msg.sender] = true;

    _balances[msg.sender] = _totalSupply;
    emit Transfer(address(0), msg.sender, _totalSupply);
}
```

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

End of report

Smart Contract Audit

November 16, 2022

Audit requested by

Baby inu

0x50f54fBCD1C6b0C847c72C71fd45032C7206A7E1